

UNIVERSITÀ DEGLI STUDI DI CATANIA

**DOTTORATO DI RICERCA IN MATEMATICA APPLICATA
ALL'INGEGNERIA
XXIV CICLO**

GABRIELE MANNO



**RELIABILITY MODELLING OF COMPLEX
SYSTEMS: AN ADAPTIVE TRANSITION SYSTEM
APPROACH TO MATCH ACCURACY AND
EFFICIENCY**

TESI DI DOTTORATO

COORDINATOR:

PROF. ANTONIO SCALIA

SUPERVISOR:

PROF. NATALIA TRAPANI

An undertaking such as this can never be accomplished in isolation; it is only through the support and advice of many others, both technical and moral, that it can be carried out.

I would like to acknowledge the CTI² research group of the Department of Industrial and Mechanical Engineering of the University of Catania and the Centre for Software Reliability team of the London City University. Their collective technical guidance added much to the content of this work.

In particular I would like to mention Professor Natalia Trapani, my thesis advisor, and Professor Peter Popov, my supervisor during the time I spent at the City University as a visiting researcher, for their continuous support and significant advices.

Lastly, but by no means lesser in importance, I would like to thank my family and friends for their moral support during the time of this effort. While technical advice is necessary, it is by no means sufficient. Without their encouragement, none of this work would have been possible.

INDEX

INDEX	I
INTRODUCTION	VII
 CHAPTER 1: CONCEPTS OF RELIABILITY ENGINEERING	
1.1 INTRODUCTION	1
1.2 SYSTEM DEFINITION	2
1.3 DEPENDABILITY CONCEPTS AND PERFORMANCE INDEXES	4
1.4 HAZARD RATE	7
1.5 RELIABILITY MODELLING AND EVALUATION METHODS	10
1.5.1 NON STATE-SPACE MODELS	12
1.5.2 STATE-SPACE MODELS	13
1.5.3 HYBRID MODELS	15
1.5.4 ANALYTICAL SOLUTION VS SIMULATION	16
1.6 CONCLUSION	17
BIBLIOGRAPHY	18
 CHAPTER 2: CLASSIC RELIABILITY MODELS	
2.1 INTRODUCTION	22
2.2 NON STATE-SPACE MODELS	22

2.2.1	RELIABILITY BLOCK DIAGRAMS	23
2.2.1.1	RBD EVALUATION METHODS	24
2.2.1.2	ADVANTAGES AND DISADVANTAGES OF RBD	26
2.2.2	FAULT TREE ANALYSIS	28
2.2.2.1	STRUCTURE FUNCTION	29
2.2.2.2	QUALITATIVE ANALYSIS	29
2.2.2.3	QUANTITATIVE ANALYSIS	30
2.2.2.4	COMPARISON WITH RBD	31
2.2.2.5	ADVANTAGES AND DISADVANTAGES OF FT	32
2.3	STATE-SPACE MODELS	33
2.4	CONCLUSION	37
	BIBLIOGRAPHY	39

CHAPTER 3: HYBRID MODELS

3.1	INTRODUCTION	44
3.2	DYNAMIC FAULT TREE	46
3.2.1	DFT OBJECTS AND ASSUMPTIONS	47
3.2.2	DYNAMIC GATES	48
3.2.2.1	FUNCTIONAL DEPENDENCY GATE	48
3.2.2.2	SPARE GATE	49
3.2.2.3	PRIORITY AND GATE	50
3.2.2.4	SEQUENCING ENFORCING GATE	50
3.2.3	SOLUTION METHODS	50
3.2.4	RELATED WORK	51

3.2.4.1	DYNAMIC RELIABILITY BLOCK DIAGRAM.....	51
3.2.4.2	BOOLEAN DRIVEN MARKOV PROCESS.....	52
3.2.5	ADVANTAGES AND DISADVANTAGES OF DFT.....	52
3.3	STOCHASTIC EXTENSIONS OF PETRI NETS.....	53
3.3.1	STOCHASTIC PETRI NET CLASSIFICATION.....	55
3.3.1.1	STOCHASTIC PETRI NET.....	56
3.3.1.2	GENERALIZED STOCHASTIC PETRI NET.....	56
3.3.1.3	STOCHASTIC PETRI NET WITH GENERAL DISTRIBUTION.....	57
3.3.2	STOCHASTIC ACTIVITY NETWORKS.....	58
3.4	CONCLUSION.....	66
	BIBLIOGRAPHY.....	67

CHAPTER 4: MATCARLORE

4.1	INTRODUCTION.....	71
4.2	MONTE CARLO SIMULATION OF DFT.....	72
4.3	SIMULINK® LIBRARY.....	75
4.3.1	THE BE BLOCK.....	76
4.3.2	THE PAND BLOCK.....	77
4.3.3	THE SPARE BLOCK.....	79
4.3.4	THE SEQ BLOCK.....	86
4.3.5	THE FDEP BLOCK.....	87
4.4	A COMPARATIVE EXAMPLE.....	87
4.5	JDFTDES: A JAVA-MATLAB INTEGRATED TOOL.....	90
4.5.1	JAVA INTERFACE.....	93

4.6 CONCLUSION	94
BIBLIOGRAPHY	95
 CHAPTER 5: ADAPTIVE TRANSITION SYSTEM	
5.1 INTRODUCTION	100
5.2 ADAPTIVE TRANSITION SYSTEMS	108
5.2.1 SYSTEM VARIABLES	109
5.2.2 REWARD VARIABLES	111
5.2.3 TRANSITION VARIABLES	112
5.3 ORDERED-ATS	116
5.4 THE HEAT-POWER SYSTEM	118
5.5 EXECUTION OF ATS	121
5.6 MARKOV-OATS	124
5.6.1 MOATS WITHOUT TRANSITION TIMING VARIABLES	125
5.6.2 MOATS WITH TRANSITION TIMING VARIABLES	131
5.7 DISCRETE EVENT SIMULATION OF ATS	135
5.7.1 THE ROME POWER-TELCO SYSTEM	136
5.7.1.1 ATS MODEL OF POWER NETWORK ELEMENTS	137
5.7.1.2 ATS MODEL OF BATTERIES	142
5.7.1.3 THE ONLINE RISK ESTIMATOR	143
5.8 CONCLUSION	144
BIBLIOGRAPHY	148

CHAPTER 6: SAN REPRESENTATION OF ATS MODELS

6.1 INTRODUCTION	152
6.2 MOBIUS MODELLING TOOL	152
6.3 SAN REPRESENTATION OF ATS	153
6.3.1 SYSTEM AND REWARD VARIABLES DATA STRUCTURE	153
6.3.2 SAN REPRESENTATION OF ATS TRANSITIONS	155
6.3.3 ATS-SAN UPDATE STRUCTURE	156
6.4 ATS-SAN MODEL OF THE HEAT-POWER SYSTEM	157
6.5 CONCLUSION	163
BIBLIOGRAPHY	165

CHAPTER 7: ATS MODEL OF REPAIRABLE DFT

7.1 INTRODUCTION	166
7.2 STATIC REPRESENTATION OF DFT	167
7.2.1 STATIC REPRESENTATION OF SPARE GATES	169
7.2.2 STATIC REPRESENTATION OF FDEP GATES	169
7.2.3 STATIC REPRESENTATION OF SEQ GATES	170
7.2.4 PAND GATES	171
7.2.5 STATIC DFT	171
7.3 ATS MODEL OF DFT COMPONENTS	172
7.3.1 PRIMARY COMPONENTS	172
7.3.2 SPARE COMPONENTS	174
7.3.3 REWARD FUNCTION SPECIFICATION	181

7.4 SAN IMPLEMENTATION AND EVALUATION	181
7.5 MATCARLOAV: AN EXTENSION TO SOLVE RDFT	183
7.6 CONCLUSION	183
BIBLIOGRAPHY	184
CONCLUSION AND FUTURE WORK	186
APPENDIXES: PUBLICATIONS	189
A.1 DFT RESOLUTION: A CONSCIOUS TRADE-OFF BETWEEN ANALYTICS AND SIMULATIVE APPROACHES	190
A.2 MATCARLORE: AN INTEGRATED FT AND MONTE CARLO SIMULINK TOOL FOR THE RELIABILITY ASSESSMENT OF DYNAMIC FAULT TREES	202
A.3 THE EFFECT OF CORRELATED FAILURE RATES ON RELIABILITY OF CONTINUOUS TIME 1-OUT-OF-2 SOFTWARE	218
A.4 AN OPEN SOURCE APPLICATION TO MODEL AND SOLVE DYNAMIC FAULT TREES OF REAL INDUSTRIAL SYSTEMS	232
A.5 MATCARLOAV, AN EXTENSIBLE MATLAB LIBRARY FOR THE SIMULATIVE EVALUATION OF DYNAMIC FAULT TREES	240

INTRODUCTION

A model is a simplification of a real system so that we can better understand the system we are developing or analysing. Modelling is an essential aspect of any engineering field. Engineers, managers and domain experts use models for many practical reasons. Just to name a few: a model helps us to visualize a system as it is or as we want it to be; allows to specify the structure and the behaviour of a system; gives us a template that guides in constructing a system and documents the decision we have made. In most circumstances a model must be supported by appropriate evaluation methods that allow us to retrieve some specified measures of interest of the system.

The main objective of this thesis is to define a modelling formalism, provided by some type of evaluation methods, applicable in reliability engineering for modelling and evaluate reliability measure of interest complex systems. In the last years reliability engineering has been concerned with systems subjected to complex interdependencies between their parts. Since the behaviour of such systems is driven by these interdependencies, the appropriate way of capturing them in a model of the system has gained interest from academic and industrial practitioners. For instance, the typical classes of interdependencies existing in reliability engineering are those involving redundancy management, share load, maintenance management in presence of limited resource, stochastic associations between parts, shared load, etc.

The aspect of capturing interdependencies, however, has not been always the main driver in reliability modelling. Early works in reliability engineering made use of the assumption of stochastic independence between the system parts and did not consider any kind of functional dependency deriving from redundancy and maintenance management. Fault Trees (FTs) and Reliability Block Diagrams (RBDs) are an example of modelling formalisms where any kind of possible dependency is neglected.

As a matter of fact, these formalisms do not consider temporal dependencies in the logic that brings to the failure at the system level, too. For instance, the possible extension of a fire could be avoided if the fire alarm works properly at the time the fire starts. Thus, the common AND condition of a FT “alarm failed-fire” can have two meanings depending on which of the two events occurred before. However a FT cannot handle the difference between the two scenarios.

In order to deal with stochastic associations and functional and temporal dependencies many practitioners switched their attentions to Markov models. A Markov model represents a system by a set of states and transitions between states. States represent different conditions the system can be in. For instance, in a Markov chain is possible to distinguish the state “alarm failed-fire” depending on which of the two events occurred before. Transitions, on the other hand, represent possible events that may occur given the system is in a specific state.

However, practitioners almost immediately realized that Markov chains present some limitations that impact the resolution, construction and fidelity in the representation of the true behaviour of the system. As a matter of fact, Markov chains results in very large state spaces, i.e., high number of states representing the possible system conditions. Moreover, Markov chains support only exponential distribution for the time occur of events. Extensions taking into account general distributions have been considered. This has lead to semi Markov process, regenerative semi Markov processes and to some extend also to generalized semi Markov process. However, analytical solution for these models presents still many limitation such as the maximum number of simultaneously enabled transitions with general distributions, the kind of general distribution itself and again the size of the state-space.

Driven by these conditions discrete event simulation and Monte Carlo simulation have been extensively used to solve system of general complexity. There is a large literature on simulation methods and applications to complex systems for reliability studies. The main drawback concerns the time that is needed to obtain a reliable result, thus, researchers have been exploiting the use of appropriate techniques to reduce it, e.g., variance reducing techniques.

Both Markov models and discrete event simulation models however present another main disadvantage: they are not provided by an high level language of description. As a matter of fact, the manual construction of a Markov chain is a tedious and error prone work. Such can be a discrete event simulation model if the constructing procedure is not well formalized. Generalized semi Markov process aid in the construction of simulation models, but still they lack of an higher level of abstraction that would be beneficial to easily capture the structure and the behaviour of the system at hand.

High level formalisms, both using analytical or simulation evaluation methods, have been proposed. In this thesis we shall refer to these formalism as hybrid, since they are characterized by a non state-space model, e.g., FTs, but, at the moment of their birth were supported by an algorithmic conversion of the model into a Markov chain.

There are several kinds of hybrid formalisms: some are an extension of methodologies like FTs and RDBs, e.g., Dynamic Fault Trees (DFTs), Driven Boolean Markov Process (DBMP), Dynamic Reliability Block Diagrams (DRBDs); some are extensions of earlier works on distributed systems, e.g., Stochastic Petri Nets (SPNs), Stochastic Reward Nets (SRNs), Stochastic Activity Networks (SANs), Stochastic Process Algebra (SPA); etc.

With the aid of these formalisms a wide class of dependencies existing between system parts can be captured in a model of the system. Moreover, at the state of the art, both analytical and simulation techniques are provided for the evaluation of specific measures of interest.

Driven by experience with hybrid formalisms and the tools that implement them we have found some limitations concerning their applications in reliability studies. While extensions of classical reliability models, e.g., FT and RBD, impose still some limitation of the kind of behaviour that is possible to represent, extensions of models like Petri nets and Process Algebra result too general to represent the kind of systems we deal with.

For instance, Dynamic Fault Trees introduce dynamic gates to the set of Boolean gates of classic Fault Tree. With the introduction of dynamic gates some class of functional and temporal dependencies and redundancy management have become representable in a FT-like model. However, many other kind of dependencies cannot be considered and, for instance, there is still a lack of applications to evaluate measure of interest in case of repairable components.

On the other hand, stochastic extension of Petri nets allow to model “any kind of system” by a net representation of the system. While concerns about the “flat” structure of the net has been addressed and resolved in formalism like Stochastic Reward Nets, Stochastic Activity Networks and even Stochastic Process Algebra, some issues remains about the fact that, due to the generality of the formalism, a model is often subjected to the specific modeller preference. Therefore, debugging and maintenance of the model result often difficult tasks. Moreover, for instance, Stochastic Activity Networks introduce a set of predicates and functions that are not supported by graphical means.

Thus, while extensions of formalism like Dynamic Fault Trees in order to increase their generality are desirable, one would like to limit the modelling capabilities of stochastic extensions of Petri nets maintaining however a as more general language as possible for the kind of studies that one is concerned about.

Driven by these considerations we present a modelling formalism based on interdependent transition systems, i.e., a set of interdependent state-space models, supported by a set of additional variables. These variables take the form of inputs and outputs of a set of communication functions, that while allowing different transition systems to communicate, are the mean to model dependencies between the modelled

elements of a system. Moreover, in most circumstances these functions can be expressed graphically by a tree structure (a Fault Tree-like structure) that maintains, together with the state-space models, an high (graphical) level language of description of the system.

We call the formalism Adaptive Transition System (ATS), since, in our vision, transition systems adapt their behaviour on the basis of the evolution of other transition systems as time progresses. This adaption is represented by an opportune model of transitions that is able to capture the effects of changes in the system.

Our model of transition takes on from formalism like Stochastic Activity Network. We model a transition by a set of attributes that determine the time to complete of a transition. On the basis of these attributes the supporting mathematical law together with its parameter can change as time progresses. Moreover a transition can exist or not and can be restarted or not on the basis of some condition.

The conditions that define the behaviour of a transitions are specified by a set of transition functions whose inputs are a set of variables that register some condition of the evolution of the system. At every change of these variables, transition functions are evaluated in order to adapt to the new reached conditions.

Variables responsible for registering the state of the system must be set in such a way that the class of information used to define the interdependencies existing in the system can be taken into consideration. We will show that all the class of dependencies seen above can be considered by the introduction of three kind of variables that register: the state, the last completed transition and the time at which transitions most recently completed.

Moreover, in evaluating some kind of performance measure, one could be interested in the effect that the conditions specified by these measure have on the behaviour of the system itself. For instance if one want to evaluate the reliability with repair of a system, one should force all restoring activities to cease when a system level failure condition is met.

Finally, ATS can be solved via simulation or, under some constrain, through conversion into a Markov chain. From a simulation point of view ATS results very

powerful since their execution logic is specified in terms of event as in generalized semi Markov processes. The definition of the updating structure of variables is also well specified. From an analytical point of view the main advantage of ATS lays on the class of systems that are representable under the Markov assumption, i.e., dependencies on previous states are taken into account by an extended definition of state that consider the ordering of occurrence of events. However, for this reason the resulting state space is very large in size. Both evaluation techniques can be improved by considering only the subset of variables that are considered in the definition of the dependencies and measures of interest of the system.

The remainder of this dissertation is organized as follows:

- In Chapter 1 we present an introduction of the basic concepts of reliability engineering. We give the definition of a system; present some development that have been introduced in the reliability community and classical measures of interest; and illustrate the main difference between classical reliability models and state-space models and between analytical and simulation evaluation methods.
- In Chapter 2 classical reliability models and state-space models will be described in more details. Thus we give the definition of FT, RBD and Markov chains and their extensions.
- In Chapter 3 we introduce some of the hybrid formalism introduced above. Special attention will be given to Dynamic Fault Trees and Stochastic Activity Networks.
- In Chapter 4 we present a tool implemented in Simulink for the evaluation of Dynamic Fault Tree via simulation that we call MatCarloRe, i.e., Matlab Monte Carlo Reliability. Moreover, an extension of the tool in Matlab with a Java interface is also presented. The extension was developed in order to overcome the limitation of the Simulink tool to handle shared spare components.
- In Chapter 5 we present Adaptive Transition Systems in detail. We define the variables and functions of the model, the execution logic and the solution

techniques. Finally, a simulation ATS model for a real case study of Power-Telco critical infrastructures is shown.

- In Chapter 6 we show how to convert an ATS model into a Stochastic Activity Network model. We show the benefit of using ATS as an high level formalism to build SAN models with a standardized structure.
- In Chapter 7 an ATS application to solve Repairable DFT, a new concept of Dynamic Fault Trees that has not yet been introduced. We show how ATS are capable to extend the kind of dependencies handled by DFTs.

Finally, we report some conclusion and future work. Moreover, Appendix A is a collection of accepted publications in international journals and proceedings of international conferences; in Appendix B we report the developed MatCarloRe scripts and in Appendix C we give some more insights and report the Matlab scripts of the case study introduced in Chapter 5.

CHAPTER 1

1.1 INTRODUCTION

Reliability Engineering concerns the study and evaluation of a set of performance indexes that express the “ability” of a system to perform a required task during its life-cycle. Reliability engineering provides the theoretical and practical tools whereby these measures of performance can be specified, designed, predicted, tested and demonstrated [1].

Reliability is concerned with failures. The effects of product failures rang from those that cause minor nuisances to catastrophic failures involving loss of life and property. Reliability engineering was born out of the necessity to avoid such catastrophic events.

Nowadays, however, increasing attention is put also in those products whose failure does not have any major life and death consequences to the consumer. Due to increasing product-awareness of modern consumer, products that do not perform in a reliable fashion are no longer tolerated. Therefore, customer dissatisfaction can have disastrous financial consequences to the manufacturer. Moreover, it is essential for a company to know the reliability of its product and to be able to control it. Moreover, to succeed, a company must produce products that work successfully for the desired

period of time, but also must avoid the design of products that operate longer than the required life.

In this chapter we give the basic definition employed in Reliability Engineering. We start, in Section 1.2, by giving the definition of system as a ensemble of hierarchical interacting parts. Successively in Section 1.3 we give the formal definition of dependability and its performance indexes. Section 1.4 concerns with the definition of the hazard rate of a system. In Section 1.5 we give a classification of the common modelling formalism and evaluation methods in Reliability Engineering and finally in Section 1.6 we report some conclusion.

1.2 SYSTEM DEFINITION

The word “system” has a very wide connotation. There is a wide variety of systems around us. Several of them have been created by man to satisfy his needs while others exist in nature. With system we may connote anything ranging from simple, artificial or composite, physical systems to conceptual, static and dynamic systems or even organizational and information systems.

We define a system as an aggregation of parts or elements, connected in some form of interaction or interdependence to form a complex or unitary whole with a specific scope. Misra [2] defines a system as: “*a system is a set of mutually related elements or parts assembled together in some specified order to perform an intended function*”. This is a very broad definition and allows anything from a power system down to an incandescent lamp to be classified as a system provided that a system must have an objective or a function to perform.

A system has basically three levels of hierarchy [3], i.e., *systems*, *subsystems* and *components*. In such a hierarchy, a component is defined as the lowest level of hierarchy in a system and it is a basic functional unit of a system. Components, in the system definition should be regarded as those units of the system, which can be assumed indivisible in the context of the problem being considered at hand. The assembly of components connected to produce a functional unit is designated as a subsystem. This is the next higher level of hierarchy in a system, above the

component. Finally, an assembly of subsystems connected functionally to achieve an objective is called “system”.

Laprie [4] makes use of the concept of *atomic system*, i.e. a system where any internal structure cannot be discerned, or is not of interest to discern (a component, in the terminology of Misra). An atomic system has defined boundaries that distinguish it from its external environment. The external environment provides one or more inputs to the atomic system that is used to process a service(s). The service that an atomic system provides is thus dependent on the fact that the correct input is received and on the fact that it is correctly processed. Therefore, a system can be considered as a set of interrelated atomic systems working together to accomplish some common objective, purpose or goal.

Most of the engineering systems today belong to the category of complex systems. Although such a distinction between simple and complex systems is totally arbitrary, the degree of complexity of a system relates to the number of elements, their physical dimensions, multiplicity of links or connections of the constituent elements within the system, multiple functions, etc. The complexity of a system can be best defined on the basis of its structural complexity and the functions performed by the system.

A system can be modelled in terms of a set of states that describe its internal status on the basis of its input(s) and its intrinsic capability to process the input to provide the output. As an example let consider an electronic equipment. In order to provide the service the first requirement is that the power is provided to the equipment by the external environment. Given that the external input is correct the ability of processing the output depend on a series of internal characteristics like the fact that the equipment works properly, i.e., no internal failures. In a state-space characterization of such a system the generic state the system can be in is given by a two-dimensional vector where the first entry represent the correctness of the input and the second the correctness of the internal status. As we will see, state-space based approaches are widely used to describe system behaviour.

In a large variety of natural or man-made systems, the inputs, the processes and the outputs are described mostly in statistical terms and uncertainty exists in both the number of inputs and their distribution over time. Therefore, these features can be best described in terms of probability distributions and the system operation is known to be probabilistic. This is the class of system we are concerned with. *Complex systems with stochastic behaviour*. It is required that an engineering system must be trustworthy and dependable otherwise it cannot serve the purpose it was intended.

1.3 DEPENDABILITY CONCEPTS AND PERFORMANCE INDEXES

Reliability engineering is an engineering field which aims to retrieve measure of interest of a modelled system such that reliance can be placed on the service it delivers. This field of engineering has undertaken many changes in the last years since new systems and new properties of these systems have been introduced. With the introduction of the term Dependability [4] the various characteristics of reliability engineering were lined out. In this dissertation we consider only few aspect of the concept of dependability that are of interest for our purposes.

Dependability may be viewed according to different, but complementary, properties, which enable the attributes of dependability to be defined. In the context of this dissertation we are interested in the following properties:

- the *readiness* for usage leads to **availability**;
- the *continuity* of service leads to **reliability**;
- the *non-occurrence of catastrophic consequences* on the environment leads to **safety**.

Availability is concerned to the fact that a system is ready to operate when requested and is a measure that includes repairs of the system also after the system failure. Reliability is concerned with the operation of a system during its life-cycle without system failures. Repairs can be included only at component and subsystem

levels, but not at the system level. Safety is related to reliability in that the service interruption is seen as the occurrence of a catastrophic event. Another difference is the level of impact of failures on society and the control of governments. Although safety requirements can lower system reliability, we do not distinguish between safety and reliability assuming that safety can be regarded as a special type of reliability.

We now give the definition of reliability and availability as proposed by the International Telecommunications Union (ITU-T) and the mathematical formulation that leads to the evaluation of the two measure of interest.

Definition 1.1 (Reliability): Reliability is defined in International Telecommunications Union (ITU-T) recommendations E.800 as follows:

“The ability of an item to perform a required function under given conditions for a given time interval.”

Reliability can be expressed as the *probability of a system being up throughout an interval without system-level repairs*. With system level repairs we refer to the possibility of system components to be repaired when the service provided by the system is “true” and we do not allow repairs when the system service is “false”. This brings to the distinction between *reliability with repairs* and classical reliability where components are assumed not repairable.

Definition 1.2 (Probabilistic measure of Reliability). Let X be the random variable that represents the time to failure of a system and $F(t)$ the *distribution of the system life time*, we define the reliability of the systems as:

$$R(t) = P(X > t) = 1 - F(t). \quad (1.1)$$

Another important quantity in the context of reliability is the *Mean Time to Failure* (MTTF). It is the mean time between 0 and the time to failure of the system.

Definition 1.3 (Mean Time to Failure). Let X be the random variable that represents the time to failure of a system and $f(t)$ the *probability density function of the system life time*, we define the *MTTF* as:

$$MTTF = E[X] = \int_0^{\infty} t f(t) dt = \int_0^{\infty} R(t) dt. \quad (1.2)$$

Definition 1.4 (Availability). Availability is closely related to Reliability, and is also defined in ITU-T Recommendation E.800 as follows:

"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."

An important difference between reliability and availability is that reliability refers to failure-free operation during an interval, while availability refers to failure-free operation at a given instant of time, usually the time when a device or system is first accessed to provide a required function or service.

From a mathematical point of view we can describe the availability as the probability of a system being UP (i.e., providing the service) at a specific instant of time t .

Definition 1.5 (probabilistic measure of Availability). Given the stochastic process $\{I(t), t \geq 0\}$, where $I(t)$ is a Bernoulli random variable that takes on value 1 when the system is UP and 0 when DOWN, we define the availability at time t (or point availability) as:

$$A(t) = P\{I(t) = 1\}. \quad (1.3)$$

Definition 1.6 (Steady State Availability). Given the Definition 1.5 we define the steady state availability (or inherent availability) as:

$$A_{ss} = \lim_{t \rightarrow \infty} A(t). \quad (1.4)$$

Definition 1.7 (Average Availability). Given the definition of availability in eq. (1.3), we define the average availability (or interval availability) in $[0, t]$ as:

$$A_I(t) = \frac{\int_0^t A(\tau) d\tau}{t}. \quad (1.5)$$

Definition 1.8 (Mean Time to Repair). Let Y be the random variable that represents the *time to repair* of a system and $g(t)$ the *density function of the system repair time*, we define the **MTTR** as:

$$MTTR = E[Y] = \int_0^{\infty} t g(t) dt. \quad (1.6)$$

Definition 1.9 (Mean Time between Failures). Given Definitions 1.3 and 1.8 we define the **MTBF** as:

$$MTBF = MTTR + MTTR. \quad (1.7)$$

1.4 HAZARD RATE

Let us consider a sample of N_0 identical elements and let us assume that at time $t = 0$ all the N_0 are in the working state. Let us define $N_w(t)$ and $N_f(t)$ the number of working and failed elements at time t , respectively. Since we can express the probability of an event as the ratio between the number of successes and the total number of trials, we introduce:

- *Unreliability* or probability of an element to be failed at t ,

$$F(t) = \frac{N_f(t)}{N_0}. \quad (1.8)$$

- *Reliability* or probability of an element to be working at time t ,

$$R(t) = \frac{N_w(t)}{N_0}. \quad (1.9)$$

Obviously $R(t) + F(t) = 1$.

The derivative of $F(t)$, $f(t)$ is defined as the density function of the element lifetime. The quantity $f(t)dt$, the differential of $F(t)$, represents the infinitesimal probability of failure in $[t, t + dt]$. It can be shown that $f(t)$ is a probability density function, i.e., the integral of the function over the real axis is equal to 1, and that the following relation holds $f(t) = \frac{dF(t)}{dt} = -\frac{dR(t)}{dt}$. The instant of time t subdivides the area under the curve in the two zones which areas measure $R(t)$ and $F(t)$.

We can rewrite $f(t)$ as:

$$f(t) = \frac{1}{N_0} \frac{dN_f(t)}{dt}. \quad (1.10)$$

Substituting $N_w(t)$ to N_0 in 1.10 we obtain the instantaneous *hazard rate* $h(t)$. We have:

$$h(t) = \frac{1}{N_w(t)} \frac{dN_f(t)}{dt}. \quad (1.11)$$

$h(t)$ represents the ratio of population that experience a failure in the time interval $[t, t + dt]$ given the number of remaining working elements at time t .

It can be shown that the following relations hold:

$$h(t) = - \frac{R'(t)}{R(t)} = \frac{f(t)}{R(t)}, \quad (1.12)$$

$$f(t) = - \frac{dR(t)}{dt}, \quad (1.13)$$

$$R(t) = e^{-\int_0^t h(\tau) d\tau}, \quad (1.14)$$

given $R(0) = 1$.

Equation 1.12 states that it is possible to retrieve the failure rate from the knowledge of $f(t)$ since $R(t) = \int_t^\infty f(\tau) d\tau$ (see eq. 1.13). Equation 1.14 is also important because allows to retrieve the reliability function on the basis of the hazard rate function $h(t)$.

The hazard rate is a very important quantity in reliability engineering. It is common to represent the hazard rate function as a decreasing function during the burn-in period of the life of a component. It is considered constant during the normal lifetime of the component, i.e., random failures, and increasing during the wear-out period. Generally in reliability engineering this behaviour of the failure rate is represented by the well known *bathtub-curve*. It is common to represent the failure rate by the following function:

$$h(t) = a \cdot t^b, \quad (1.15)$$

where a and b are two constants greater than 0 and depending on the value of b ($b \geq 1, b < 1, b = 0$) the function results increasing, decreasing and constant, respectively. Another common way to represent $\mathbf{h}(t)$ is given by:

$$\mathbf{h}(t) = \frac{\beta}{\eta} \left(\frac{t}{\eta}\right)^{\beta-1}, \quad (1.16)$$

where $\eta > 0, \beta > 0, t \geq 0$.

Substituting 1.16 in 1.14 we obtain:

$$\mathbf{R}(t) = e^{-\left(\frac{t}{\eta}\right)^\beta}, \quad (1.17)$$

and substituting 1.17 in 1.13 we have:

$$\mathbf{f}(t) = \frac{\beta}{\eta} \left(\frac{t}{\eta}\right)^{\beta-1} e^{-\left(\frac{t}{\eta}\right)^\beta}, \quad (1.18)$$

that is the Weibull distribution function. In the case $\beta = 1$ we obtain the exponential distribution with parameter $\frac{1}{\eta}$. The exponential distribution, in fact, is the distribution for which the hazard rate is constant.

It can be shown that if \mathbf{X} is the random variable that represents the time to failure of a component the following relation holds (probabilistic interpretation of 1.12):

$$\mathbf{h}(t) = P\{t < \mathbf{X} \leq t + dt | \mathbf{X} > t\}. \quad (1.18)$$

Eq. 1.18 highlights the meaning of the hazard rate, that is: the conditional probability of a component failing in $(t, t + dt]$ given that it has not failed in $[0, t]$. Moreover from 1.18 is possible to show that the exponential distribution has a constant (with respect to time) failure rate (and it is the only distribution with this characteristic). This is directly related to the memory-less property of the exponential distribution. In fact the failure is not related to some deterioration mechanism but is the result of some suddenly appearing failure. For the sake of completeness we give the definition of the memory-less property:

$$P\{t < \mathbf{X} \leq t + \Delta t | \mathbf{X} > t\} = P\{\mathbf{X} \leq \Delta t\}. \quad (1.19)$$

1.5 RELIABILITY MODELLING AND EVALUATION METHODS

Reliability modelling is the first step in reliability evaluation. A system is usually decomposed into its constituent parts. The model must highlight the relationship between individual components, or subsystems failures with the system failures for the known system objective.

To this end the state of the system is specified in terms of the states of the various components (white-box approach). Two approaches are possible:

- *forward*, in which starting with the failure events at the component level, the system level failure is assessed as the consequence of such failures, e.g., Failure Mode and Effect Analysis (FMEA) [5].
- *bottom-up*, where starting at the system level, system performance are linked to the failures of components, proceeding downwards to the component level, e.g., Fault Tree Analysis (FTA) [6].

System performance can be linked to the component performances qualitatively and quantitatively. Qualitative analysis brings to the evaluation of the logical relationship between components, while in quantitative analysis one obtains a measure of system performance.

Logical relationship can be expressed graphically. Reliability Block Diagrams (RBDs) and Fault Trees (FTs) are the two most know formalism in this context [6,7]. In RBD logical relationships highlight the conditions for which the system succeed in fulfil its requirements, while FTs highlight the conditions for which a failure of the system is related to the failures of system constituents. Other graphical relationships are possible by which the same task can be achieved, e.g., events trees, binary decision diagrams (BDD), causal trees or diagraphs [8] *etc.* Petri nets [9-14] have shown their capabilities in system reliability assessment or fault diagnosis programs.

The underlying assumption in reliability modelling is that each component or system can have only two states, working or failed. Thus the model is a two-state model. Multi-state systems have been considered in [15,16]. They are useful in

considering degraded states in addition to working and failed states. Multi-state systems cannot be handled by FT and RBD, thus new formalisms like Dynamic Fault Trees (DFTs) and Dynamic Reliability Block Diagram (DRBD) have been introduced [17,18].

In Reliability Engineering, redundancy techniques are used to improve system reliability [1]. Model like FT and RBD can capture redundancy only in the case of independence across the components within the system. In other words, it is assumed that the failure of a component does not affect the failure properties (failure rates) of the remaining components. We can distinguish two cases of redundancy where the independence assumptions do not hold: stand-by redundancy and load-sharing. Again FTs and RBDs cannot cope with these scenarios. DFTs and DRBDs can tackle stand-by redundancy, but cannot be used to model load-sharing redundant systems. In these cases one can use formalisms like stochastic extensions of Petri Nets or Markov models.

Therefore, it is important to develop reliability models that incorporate stochastic dependencies among the system's components. Another class of dependency that is common in Reliability Engineering are shocks. In this case the system is exposed to shocks that cause random amounts of damage [1]. The shocks themselves can occur according to a random process. The intensity and occurrence frequency of the shocks may vary with time. Generally, the occurrences of shocks are modelled using homogeneous or non-homogeneous Poisson processes. The additional damage to the system at a given shock may depend on the intensity of the shock, the damage already experienced by the system, and the age of the system. The system fails when the cumulative damage exceeds a certain level. Another class of shock models includes common cause failures. For example, the bivariate shock model introduced by Marshall and Olkin [19] analyzes component dependencies by incorporating latent variables to allow simultaneous component failures.

There are models [20,21] that help to compute system reliability with the assumption of dependency of failures to approach realistic situations where the analyst cannot ignore the dependency of failures. Markov chains provide a modelling

procedure for the availability or reliability modelling of maintained systems under various assumptions of practical importance.

Reliability models has been distinguished into two main groups: non state-space models and state space model. Classically non state-space models are solved by the mean of combinatorial techniques while state-space models recall the use of Markov Chains and their generalizations. Another class of models, that we define as hybrid models, use an high level formalism of description that is more convenient to use because it reduces modelling efforts. This high level formalism of description resemble the one used by non state-space models. However, the solution of these models is achieved by their conversion into an isomorphic state-space model. We will consider hybrid approaches in the Chapter 3, while in Chapter 2 we present non state-space and state-space models.

1.5.1 NON STATE-SPACE MODELS

Classical non state-space models are:

- Series/Parallel Block Diagram (or Reliability Block Diagram, RBD);
- Non Series-Parallel Block Diagram (or Reliability Graphs);
- Fault Trees without repeated events;
- Fault Trees with repeated events.

All the models are similar in that they capture condition that make the system fail in terms of the structural relationship between the system components. In fact, these models are usually solved by retrieving the *structure function* of the system; that is that combination of events that leads to the system failure. The model is solved analytically without generating the state-space using techniques like Boolean Algebra, order statistics and convolution. Given a set of components that make up the system and given for each of them a quantity like a probability of failure, a failure rate, a distribution of time to failure or the steady state or instantaneous availability and, assumed statistical independence, the resulting model is very easy to use and able to evaluate measures like reliability (without repair), point and steady state availability and the system MTTF. In the case of repairable components is also

assumed that the repair units are as many as needed. Relatively good algorithms are available to solve these models so that large number of component can be handled, e.g., Sum of Disjoint Product (SDP) algorithms, Binary Decision Diagram (BDD) algorithms, Factoring and Series-Parallel composition algorithms.

The main drawback of these methodologies is that they cannot easily handle failure/repair dependencies, e.g., shared repair, warm/cold spares, imperfect coverage, non-zero switching time, travel to repair person, reliability with repair.

1.5.2 STATE-SPACE MODELS

In the early approaches, components of a system were assumed to be independent, whereas in practice, the times to failure and recover from a fault depend, broadly speaking, on the state of the system. As a result, combinatorial models, such as FTs and RBDs, cannot be used to accurately model the system behaviour. Further, non state-space (combinatorial) models cannot adequately model the sequence-dependent failure mechanisms associated with spares management, changing working conditions, and so on. Demands for increased accuracy in reliability estimation quickly forced the development of more elaborate system models without the vastly simplifying independence assumptions [22].

For this reason, many modellers turned to Markov chains for reliability assessment of fault tolerant systems. Markov chains are extremely flexible and they can capture the fault coverage mechanism quite well. They belong to the group of state-space methods. For instance a state can keep track of the number of functioning resource of each type, the recovery state for each failed resource, the allocation of resource to tasks, etc. Transitions are directed from one state to another and represent the change of the system state due to the occurrence of an event. Transition are labelled and the kind of label depends on the kind of state-space model used. In fact we can distinguish state space model in: Markovian, i.e., Discrete Time Markov Chain (DTMC), Continuous Time Markov Chain (CTMC), Markov Reward Models (MRC); non Markovian, i.e., Semi Markov Processes (SMP), Markov Regenerative Processes (MRGP) and Generalized Semi Markov Process (GSMP) [23-28].

Thus the label of a transition can be a probability in the case of DTMC, a rate in the case of CTMC, a distribution function in the case of SMP or two distribution function in the case of MRGP or GSMP. Finally time dependent rates are handled by the mean of Non Homogeneous Continuous Time Markov Chains (NHCTMC).

Markov Models are effective in handling fault-tolerance and recovery/repair, dependencies, contention for resources, concurrency and timeliness, etc. The use of MRGP and NHCTMC allows to model general distributed event times and Weibull failure distribution, respectively.

The main drawback of these methodologies is the large state space, i.e. exponential in the number of components. This brings to the problems of specification, storage and solution of the model. Storage problems can be handle with sparse matrices and with methods of truncation and lumping of the state space; the solution problem with “sparsity preserving” solution methods, e.g., successive overrelaxation, Gauss-Seidel, uniformization, ODE solution methods. Finally the specification problem can be handle by a higher level formalism like Generalized Stochastic Petri Nets (GSPN) that furnishes aid in the generation of the chain using a formalism that allow hierarchical composition, concurrency, contention and conditional branching and thus facilitating the construction of the model.

In addition to computational complexity, a major disadvantage of Markov chains (state-space models) is that it is difficult to determine the correct Markov model for a given system. This is because the modeller must specify each operational configuration of the system explicitly and determine the rate at which the system changes from one state to another. However, the relative advantages of combinatorial models (fault trees and RBDs) and Markov models have been exploited by using two key techniques: a) behavioural decomposition [29], and b) automatic conversion of a non state-space model to an equivalent Markov model [30-32].

Several different Markov chain based methods are available for reliability analysis. The basic idea is to construct a single Markov chain to represent the failure behaviour of the entire system. Solving the Markov chain models yields the

probability of the system being in each state. The system unreliability is obtained by summing all the failure state probabilities.

Instead of generating and solving an overall Markov chain one could generate and solve separate Markov chains for each independent subsystem. Apparently, each individual Markov chain is much smaller than the overall Markov chain. The reliability (or unreliability) of the system can be computed merging the results of the independent Markov chains.

Both state-space and non state-space models will be briefly discussed in Chapter 2. They represent the background for any further develop in reliability modelling and thus will be considered as a comparative mean in the following of this dissertation.

1.5.3 HYBRID MODELS

With hybrid models we refer to the class of formalisms that use non state-space representation of the system, but whose quantitative evaluation has been usually developed generating a state space model. In Chapter 3 we will introduce two kind of these formalisms: extensions of the non state-space models described above; and stochastic extension to Petri Nets.

Among the former we will introduce Dynamic Fault Trees (DFT) and give a brief description of Dynamic Reliability Block Diagrams (DRBD) and Boolean Driven Markov Processes (BDMP) [3]. Among the second we will introduce Stochastic Petri nets (SPN) and their generalization with particular attention to Stochastic Activity Networks (SAN) [34]. We will see that hybrid formalism overcome many of the limitation of state-space and non state-space models.

To the class of hybrid models belong also Adaptive Transition Systems (ATS), a formalism developed during this doctoral course in order to overcome two main problems observed when using DFTs and SANs. In particular ATS deal with the limited modelling capabilities of DFT and the “generality” of SAN models.

1.5.4 ANALYTIC SOLUTION VS SIMULATION

Beside the modelling approach used to describe a system, another important aspect concerns the evaluation methods. We can distinguish between two kinds of evaluation methods: Discrete Event Simulation and Analytic (or numerical).

Analytic solutions aim to obtain a closed form solution of the measure of interest, or when the model results too complex they aim to retrieve the solution by evaluating it numerically by the assistance of a tool. Analytic solutions are generally of two kinds: combinatorial, when a non state-space model is used; or determined by the definition of the (integro)-differential equations that describe the evolution of a stochastic process, in the case of state-space model.

Limitations of combinatorial approaches are the difficulty to consider the effect of time on the system behaviour. On the other hand, the definition of a stochastic process requires the definition of the state-space of the system that may be too large to be solved. Moreover, both approaches suffer when general distributions of the time of events are used to describe the system behaviour.

On the other hand Discrete Event Simulation (DES) has proven to be an effective approach to retrieve measure of interest of large systems that present a complex behaviour not easily caught by state-space models [35-40]. The methodology does not suffer from the state explosion problem and can be effectively used in presence of general distributions. The use of DES implies statistical analysis of the output like design of experiments, hypothesis testing, statistical inference, analysis of variance and regression models. The main question when using DES is how many simulation runs are sufficient, a question that can be resolved depending on the level of accuracy required. DES can represent in detail the system behaviour. However the main drawback of the methodology is that the simulation time can be very costly (although in some case variance reduction methods can be applied).

1.6 CONCLUSION

In this chapter we have presented the principal concepts of Reliability Engineering with particular focus on the modelling task and modelling methodologies. We have introduced the three classes of models that will be described in this dissertation: non state-space methods, state-space methods and hybrid methods. Each of these methods present advantages and disadvantages. It is, then, clear that the choice of the “right” technique is dictated several parameters concerning the outputs of the analysis. Among these parameters we have:

- the nature of the measure of interest that we want to evaluate;
- the level of detail of description of the system behaviour that we want to achieve;
- the convenience of the model specification and solution;
- the representation power of the model; and
- the access to suitable tools.

Following this parameterization, one should use non state-space models when the measure of interest is not too “complex”, when the level of detail or the behaviour of the system are tractable by combinatorial techniques, etc.. State-space models should be used when the system is complex, when dependencies are present and the level of detail is elevated.

The use of hybrid formalisms depend on the formalism itself. DFTs should be used when we want to capture temporal and functional dependencies but their power is limited when considering complex measure of interest or higher level of details. In these case, therefore, one should use stochastic extensions of Petri nets. The choice is furthermore dictated by the available tools.

Finally, regarding evaluation methods, DES should be used in the case of very complex systems with very complex interactions, e.g., power networks, or in the case when analytical models cannot solve systems too large or systems with general distributions of the time of events.

As we will see, however, in Chapter 5 a possible composition of more formalism is possible, allowing to exploit the advantages of single modelling and evaluation methods.

BIBLIOGRAPHY

- [1] Kececioglu, D., 1991. *Reliability Engineering Handbook*, Prentice Hall, Inc., Englewood Cliffs, New Jersey.
- [2] Misra, K.B., 2008. *The Handbook of Performability Engineering*. Springer.
- [3] Misra, K.B., 1992. *Reliability Analysis and Prediction: A Methodology Oriented Treatment*. Elsevier Science, Amsterdam.
- [4] Laprie., J.C., 1985. *Dependable computing and fault tolerance: concepts and terminology*. In Digest of FTCS-15; 2-11.
- [5] Langford, J. W., 1995. *Logistics: Principles and Applications*. McGraw Hill.
- [6] Vesely, W.E., Goldberg F.F., Roberts N.H. & Haasl D.F., 1981. *Fault tree handbook*. U.S. Nuclear Regulatory Commission, Washington DC.
- [7] Murphy, K.E. & Carter, C.M., 2003. *Reliability Block Diagram Construction Techniques: Secrets to Real-Life Diagramming Woes*. In Proceedings Annual Reliability and Maintainability Symposium-Tutorial Notes. Tampa, Florida.
- [8] Chang, Y.R., Amari, S.V. & Kuo, S.Y., 2005. *OBDD-based evaluation of reliability and importance measures for multistate systems subject to imperfect fault coverage*. IEEE Transactions Dependable and Secure Computing; 2(4): 336-347.
- [9] Malhotra, M. & Trivedi, K.S., 1995. *Dependability modelling using Petri nets*. In IEEE Trans. on Reliability; 44:428-440.
- [10] Sifakis, J., 1977. *Use of timed Petri nets for performance evaluation*. 3rd Int. Symp.. Measuring, Modeling and Evaluating Computer System. Beliner and Gelenbe, Editors; 75-95.

- [11] Hura, G.S., 1982. *Petri nets as a modeling tool*. Microelectronics and Rel; 22(3): 433-439.
- [12] Molloy, M.K., 1982. *Performance analysis using stochastic Petri nets*. IEEE Trans. on Rel.; 31(9): 913–917.
- [13] Hura, G.S. & Etessami, F.S., 1988. *The use of Petri nets to analysis coherent fault trees*. IEEE Trans. on Rel.; 37(5):469-474.
- [14] Hura, G.S., 1993. *Use of Petri nets for system reliability evaluation*. In Misra K.B., (Ed.) *New Trends in System Reliability Evaluation*. Elsevier; 339-364.
- [15] Levitin, G., 2003. *Reliability evaluation for acyclic transmission networks of multi-state elements with delays*. IEEE Transactions on Rel.;52(2):231-237.
- [16] Levitin, G., 2003. *Reliability of multi-state systems with two failure-modes*. IEEE Trans. on Rel.; 52(3):340-348.
- [17] Amari, S., Dill, G., & Howald, E., 2003. *A new approach to solve dynamic fault trees*. Annual Reliability and maintainability symposium; 374–379.
- [18] Distefano, S., & Puliafito, A., 2007. *Dynamic reliability block diagrams vs dynamic fault trees*. In Proceedings Annual Reliability and Maintainability Symposium RAMS '07; 71-76.
- [19] Marshall, A.W. & Olkin, I., 1967. *A multivariate exponential distribution*. Journal American Statistical Association; 62:30-44.
- [20] Balagurusamy, E. & Misra, K.B., 1976. *Failure rate operating chart for parallel redundant units with dependent failures*. IEEE Trans. on Rel.; 25(2):122.
- [21] Misra, K.B., 1992. *Reliability analysis and prediction: A methodology oriented treatment*. Elsevier Science Publishers BV, Amsterdam.
- [22] Geist, R. & Trivedi, K.S., 1990. *Reliability estimation of fault-tolerant systems: Tools and techniques*. IEEE Computer, Special Issue on Fault-tolerant Computing; 23:52-61.

- [23] Nielsen, S.F., 2009. *Continuous-time homogeneous Markov chains*. Copenhagen: University of Copenhagen - Department of Mathematical Sciences.
- [24] Boudewijn, R. Haverkort, 2002. *Markovian Models for Performance and Dependability Evaluation*. Lectures on formal methods and performance analysis. Springer Verlag, New York.
- [25] Bause, F. & Kritzinger, S.P., 1996. *Stochastic Petri nets - an introduction to the theory*. Advanced Studies in Computer Science. Vieweg Verlagsgesellschaft.
- [26] William J. Stewart., 1994. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 41 William Street, Princeton, New Jersey 08540.
- [27] Cinlar, E., 1969. *Markov Renewal Theory*. Advances in Applied Probability, 1(2): 123-187
- [28] Glynn, P. W., 1989. *A GSMP formalism for discrete-event systems*. Proceedings of the IEEE; 77:14-23.
- [29] Trivedi, K.S. & Geist, R., 1983. *Decomposition in reliability analysis of fault tolerant systems*. IEEE Trans. on Reliability; 32:463-468.
- [30] Bavuso, S.J., Dugan, J.B., Trivedi, K.S., Rothmann, E.M. & Smith, W.E., 1987. *Analysis of typical fault-tolerant architectures using HARP*. IEEE Trans. on Reliability; 36:176-185.
- [31] Boyd, M.A., Veeraraghavan, M., Dugan, J.B. & Trivedi, K.S., 1988. *An approach to solving large reliability models*. Proc. of IEEE/AIAA 8th Embedded Digital Avionics Conf.; 243-250.
- [32] Trivedi, K.S., Dugan, J.B., Geist, R. & Smotherman, M., 1984. *Modeling imperfect coverage in fault-tolerant systems*. Fault-Tolerant Computing Symp. (FTCS) IEEE Computer Society; 77-82.
- [33] Bouissou, M. & Bon, J.L., 2003. *A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes*. RESS; 149-163.

- [34] Sanders, W.H., & Meyer, J.F., 2002. *Stochastic activity networks: Formal definitions and concepts*. Lectures on Formal Methods and Performance Analysis. SpringerVerlag.
- [35] Durga Rao, K., Gopika, V., Sanyasi Rao, V.V.S., Kushwaha, H.S., Verma, A.K., & Srividya, A., 2009. *Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment*. Reliability Engineering and System Safety; 94:872-883.
- [36] Marsaguerra, M., Zio, E., Devooght, J., & Labeau, P.E., 1998. *A concept paper on dynamic reliability via Monte Carlo simulation*. Mathematics and Computers in simulation; 47:371-382.
- [37] Marquez, A.C., Heguedas, A.S., & Iung, B., 2005. *Monte Carlo-based assessment of system availability. A case study for cogeneration plants*. RESS;88:273-289.
- [38] Zio, E., Marella, M., & Podollini, L., 2007. *A Monte Carlo simulation approach to the availability assessment of multi-state systems with operational dependencies*. RESS; 92:871-882.
- [39] Zio, E., Podofillini, L., & Levitin, G., 2004. *Estimation of the importance measures of multi-state elements by Monte Carlo simulation*. RESS; 86:191-204.
- [40] Marseguerra, M., & Zio, E., 2004. *Monte Carlo estimation of the differential importance measure: application to the protection system of a nuclear reactor*. RESS; 86:11-24.

CHAPTER 2

2.1 INTRODUCTION

In Chapter 1 we introduced two classes of reliability models: state-space and non state-space models. In this chapter we present a review of the most known models for reliability evaluation that belong to these classes. In Section 2.2 we will present two of the most used non state-space models in Reliability Engineering: Reliability Block Diagrams (RBD) and Fault Trees (FTs). In Section 2.3, among state-space models, we will present the classes of stochastic processes mostly used in Reliability Engineering; Markov Chains and generalizations. Advantages and disadvantages of the revised modelling methodologies are discussed and finally in Section 2.4 we will report some conclusions.

2.2 NON STATE-SPACE MODELS

In this section we describe the following models introduced in Chapter 1: Reliability Block Diagrams (RBDs), Reliability Graphs (RG) and Fault Trees (FTs). The modelling language is introduced together with the resolution techniques. Finally we address the main limitations of these models.

2.2.1 RELIABILITY BLOCK DIAGRAMS

Reliability Block Diagram (RBD) is a well known modelling methodology in Reliability Engineering. RBD belongs to the class of non state-space models. RBDs are usually solved by the mean of combinatorial evaluation methods [1,2]. They make use of a very intuitive representation that results easy to implement and analyze.

In RBDs each system component is represented by a block. Blocks are connected following two kind of configurations: series and parallel.

- In a series configuration, the failure of one component causes the failure of the system; while
- in a parallel configuration all the N components belonging to the parallel structure must be failed for the system to fail.

Mixed architectures are allowed combining together series and parallel structures. In the case of more complex architecture, e.g., bridge connections, we refer to Reliability Diagrams. In the latter case the topology of the system is said complex [1] and it is not resolvable with the traditional methods used in RBDs.

Another possible configuration in RBDs is the k/N structure; that is a parallel configuration where k out of N block are required to be operating.

In RBD a block can be viewed as a switch that is closed when the block is operating and open when the block has failed. In fact only two states, i.e., failed and operating, are used to describe the state of a component. The system is operational if a path of closed switches is found from the input to the output of the diagram.

Definition 2.1 (Reliability Block Diagram). A RBD is a 4-tuple $M = (C, L, N, J)$ where:

- C , is the set of blocks;
- L , is the set of connections existing between blocks;
- N , is the set of nodes. For each diagram is given at least an input and an output node.

- J , is the set of connection relation such that:
 - i. $N \times L \times C$, is the connection relation with respect to the input node;
 - ii. $C \times L \times N$, is the connection relation with respect to the output node;
 - iii. $C \times L \times C$, is the connection relation between blocks.

2.2.1.1 RBD EVALUATION METHODS

To solve a RBD by the mean of combinatorial or order statistics the following hypothesis are made [3]:

- failure of individual components are assumed to be independent;
- at the initial time all the components are assumed to be operating;
- any block can be either failed or operating;
- only active redundancies are allowed, e.g., no warm/cold stand-by.

Serial-parallel reduction algorithms are used to solve a diagram with series parallel structures. For a series system made up of independent components we can use the product law of reliabilities to retrieve the reliability of the system. Let E_i denote that the i -th component of the RBD is operating and $R_i = P(E_i)$ its reliability. Then the series system reliability is given by (under the assumption of independency):

$$R_S = P(\cap_i E_i) = \prod_i P(E_i) = \prod_i R_i(E_i). \quad (2.1)$$

For a parallel system configuration we have (product law of unreliabilities $P(\bar{E}_i)$):

$$R_P = 1 - P(\cap_i \bar{E}_i) = 1 - \prod_i P(\bar{E}_i) = 1 - \prod_i (1 - R_i). \quad (2.2)$$

In 2.1 and 2.2 reliability is treated as a probability instead of a function of time (Figure 2.1). In fact 2.1 and 2.2 are a special case of order statistic.

Order statistics is useful when considering k/N configurations in the case that all the time to failure of the components in the structure are independent and identically distributed random variables (although generalization is also possible).

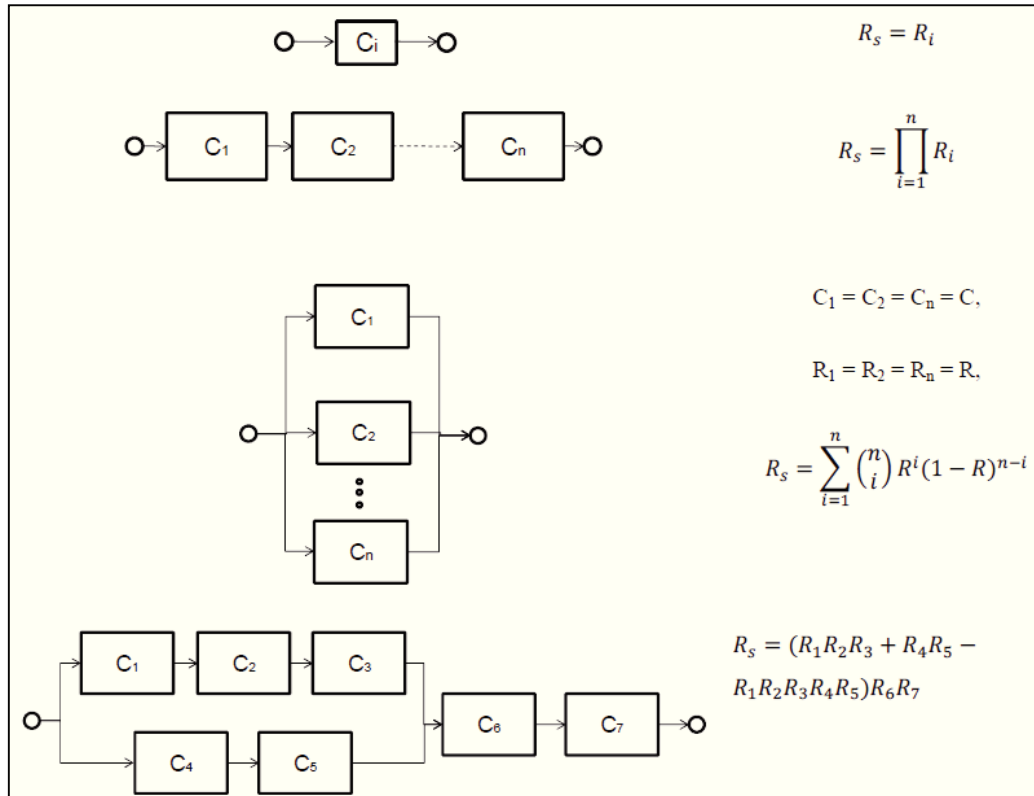


Fig. 2.1. Reliability of common RBD structures.

Let X_1, X_2, \dots, X_N be N independent and identically distributed random variables with the same distribution function F and the same density f . Let Y_1, Y_2, \dots, Y_N be N independent random variables obtained by permuting the set X_1, X_2, \dots, X_N so that they are in increasing order. The random variable Y_k is called the k -th order statistic. To derive the distribution of Y_k we first note that the probability that a given X_j is less than or equal to y is $F(y)$. So the probability that exactly j of X_j 's lie in $(-\infty, y]$ and $(n-j)$ in $(y, +\infty)$ is given by the binomial probability mass function $\binom{N}{j} F^j(y) [1 - F(y)]^{N-j}$. Hence, Y_k is less than or equal to y , if k of the X_i 's are less than or equal to y . Thus, the cumulative density function of Y_k is given by $F_{Y_k}(y) = \sum_{i=k}^N \binom{N}{i} F^i(y) [1 - F(y)]^{N-i}$ and the reliability of a k/N parallel RBD is given by:

$$R_{k|N}(t) = \sum_{i=k}^N \binom{N}{i} R^i(t) [1 - R(t)]^{N-k}. \quad (2.3)$$

Function of random variables can be used to retrieve a closed form for the above relations. For instance to evaluate the reliability at time t in the case of a series configuration we can introduce the random variable $Z = \min_i(X_i)$, where X_i is the random variable that represents the time to failure of the i -th component of the series RBD. Then the reliability of the systems is given by $P\{Z > t\}$.

At the same way for a parallel system, introduced the random variable $W = \max_i(X_i)$, we can evaluate the reliability of the system by $P\{W > t\}$.

Assuming independence we retrieve again solution of the kind of 2.1 and 2.2. However this methodology is very expensive and leads to very complicated formulations even in the case of exponential distribution functions. For instance in the case of the k/N configuration made of independent and identically distributed exponential random variables these leads to a hypo-exponential distribution with parameters $(n\lambda, (n-1)\lambda, \dots, k\lambda)$ while in the case the random variables are not identically distributed the resulting distribution is very complex.

Joint distributions, and convolution in the case of sum of independent random variables, can be used to model warm, cold and hot stand-by redundancy although the use of state space models, where possible, would facilitate the evaluation of the measure of interest.

2.2.1.2 ADVANTAGES AND DISADVANTAGES OF RBD

The main advantages of RBDs are the easy implementation and resolution as well as the peculiarity of representing the reliability structure of the system. However, when more complex structures are considered, i.e., Reliability Graphs, techniques like state enumeration (Boolean true table), factoring and conditioning and Binary Decision Diagrams (BDDs) should be used.

- Boolean true table requires to enumerate all the states and for each of them evaluate the operability of the system. For those state where the system is operating probabilities are evaluated and summed together using Boolean

algebra. BDDs can be seen as the graphical representation of the truth table and give an efficient way to build it.

- Factoring and conditioning requires the definition of a condition, i.e., working or failed, on a specific block so that it can be eliminated from the graph. In this way the resulting graph is made of only series and parallel configuration. By the mean of the theorem of the total probability the reliability of the system is evaluated taking the reliabilities evaluated with the imposition of the conditions.

Reliability Graphs represent the generalization of RBDs and many combinatorial models can be converted into RGs (Figure 2.2). As RBDs consist of nodes and edges, where edges represent components that can fail. There is always a source (input node of the diagram) and a sink (output node of the diagram). They are often referred as a non-series-parallel RBDs and are used in network reliability problems.

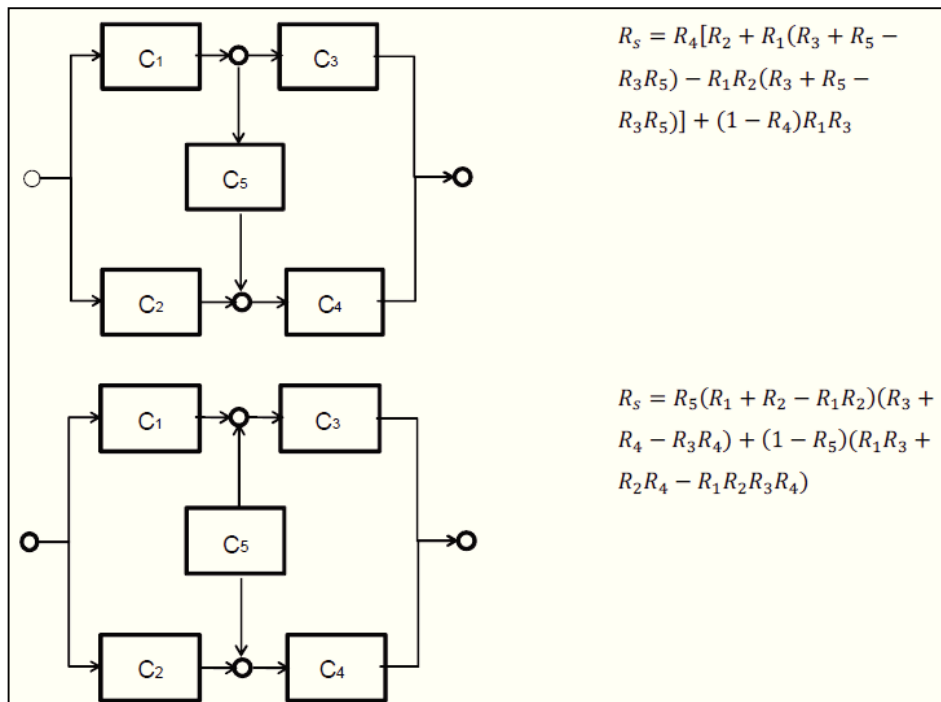


Figure 2.2. Example of Reliability Graphs and solutions.

The main drawback of RBDs is that the methodology cannot tackle dependencies between the modelled elements. As said before, the independence assumption of the time to failure of components must be respected for the evaluation methods to be

applied. These constraints are overcome in Dynamic Reliability Block Diagram (DRBD). We will discuss them shortly in Chapter 3 among the hybrid reliability models.

2.2.2 FAULT TREE ANALYSIS

Fault Tree Analysis (FTA) was developed in 1962 at Bell Telephone Laboratories in the analysis of the launch control system of the intercontinental Minuteman missile [4]. It was later adopted, improved, and extensively applied to many different contexts, so that, FTA has become one of the most widely used techniques for system reliability and safety assessments. Therefore, different forms of FTs, including static, dynamic, parametric and extended have been proposed [5,6,7].

FTA is an analytical technique, where on the basis of an undesired event (the system failure) all combinations of basic events (BE) that will lead to the occurrence of the predefined event (or top event TE) are defined [8].

BEs represent basic causes for the TE; examples are: component failures, human errors, environmental conditions, etc. In first instance a FT is a graphical representation of logical relationships between the TE and the BEs. More generally can be defined as a framework for the assessment of system properties such as reliability, availability and other measures of interest.

To build a FT we start with the failure scenario being considered, and decompose the failure symptom into its possible causes. Each possible cause is then investigated and further refined until the basic causes of the failure are understood. For more details, one can refer to [9,10]. The failure scenario to be analyzed is normally called the TOP event of the fault tree. The basic causes are the basic events of the fault tree. The fault tree should be completed in levels, and they should be built from top to bottom. However, various branches of a fault tree can be built to achieve different levels of granularity.

Definition 2.2 (Fault Tree). A FT is a 4-tuple $M = (TE, BE, G, R)$ where:

- TE , is the top event or the top node of the FT;

- BE , is the set of basic events or lower level nodes;
- G , is the set of Boolean gates, i.e., a function of Boolean variables that return a Boolean value;
- R , is the set of connection relations between the elements of BE and G , the elements of G and the elements of G and TE .

More details of these elements and their graphical representation can be found in [11,12,13].

2.2.2.1 STRUCTURE FUNCTION

To solve FTs we can make use of the concept of the complement to the structure function and of the polynomial algorithm presented for RBDs. The structure function defines whether the system is operating or less on the basis of the state of its components. In the case of a FT the structure function is represented in term of the logical gates that propagate the failure of the BEs up to the TE. Given the state vector $X = \{X_1, X_2, \dots, X_N\}$ where X_i takes on value 1 if the i -th component is working and 0 if failed, the structure function is defined as

$$\Phi(X) = \begin{cases} 1, & \text{if system working} \\ 0, & \text{otherwise} \end{cases} \quad (2.4)$$

In case of repeated events Boolean algebra can be used to retrieve the set of minimal cut set (MCS), i.e., the minimum set of events that leads to the occurrence of the TE, and use the method of the sum of disjoint products to evaluate the unreliability. This is necessary since the terms in the MCS are not mutually exclusive. While a FT without repeated events can be solved in nearly polynomial time, the complexity of a FT with repeated events is exponential with the number of BEs.

2.2.2.2 QUALITATIVE ANALYSIS

Qualitative analysis is usually based on the analysis of Minimal Cut-Sets (MCSs). A Cut-Set (CS) is a set of BEs whose occurrence leads to the occurrence of the TE. A MCS is a CS without redundancy. MCSs are evaluated applying a top-down

approach. Starting from the top gate (the gate connected to the TOP event of the fault tree), CSs are built by considering the gates at each lower level. AND gates are replaced by a list of all its inputs (i.e., intersection or product of lower level elements). In OR gates the occurrence of any input can activate the gate. In this case the CS is split into several CSs, one containing each input to the OR gate (i.e., union or sum of lower level elements). Possible results from the qualitative analysis based on MCSs include:

- combinations of component failures that may result in a critical event in the system (the individual minimal cut set);
- single components whose failure leads to the system failure (singleton CS);
- focus on specific components (considering MCSs that contain the component of interest).

Qualitative results are thus useful to identify conditions that might lead to the system failure. In this way one can take proper *preventive* measures can plan specific *reactive* measures.

2.2.2.3 QUANTITATIVE ANALYSIS

Based on the evaluation of the MCSs from a FT the probability of occurrence of the TE can be simply retrieved as the probability that all the basic events in one or more MCSs will occur [9,10,14].

Let C_i ($i = 1, 2, \dots, N$) denote the N MCSs derived from a FT. The probability of occurrence of the TE, P_{TE} is given by:

$$P_{TE} = \Pr (\cup_{i=1}^N C_i). \quad (2.5)$$

Generally, MCSs are not disjoint. Thus, the probability of the union in (2.5) is not equal to the sum of the probabilities of the individual MCSs. Several methods exist for the evaluation of (2.5) [9,10, 14]. Among them we describe the sum of disjoint products (SPD), where the solution is retrieved making disjoint MCSs using Boolean algebra. We have:

$$\bigcup_{i=1}^N C_i = C_1 \cup (\overline{C_1}C_2) \cup (\overline{C_1}\overline{C_2}C_3) \cup \dots \cup (\overline{C_1}\overline{C_2}\overline{C_3} \dots \overline{C_{N-1}}C_N), \quad (2.6)$$

where $\overline{C_i}$ represents the negation of C_i . Thus P_{TE} can be evaluated substituting (2.6) into (2.5) applying the property of the probability of the union of disjoint events.

Another solution technique, efficient for the evaluation of large FTs are Binary Decision Diagrams (BDD). The reader may refer to [15,16,17,18,19,20,21] for a complete definition of BDD and their application to FTs.

In some circumstance FTs are solved via Discrete Event Simulation. This can be due to very large FTs (where the evaluation of MCSs can be troublesome) or because to the use of “exotic” gates like in the case of non-coherent FTs. In this case, the time to failure of each BE is first sampled and compared with the mission time. If the time to failure of the component is less than the mission time, the component is regarded as failed. Once that all the components state have been defined the tree is evaluated bottom-up in order to assess the occurrence of the TE.

2.2.2.4 COMPARISON WITH RBD

The most fundamental difference between FTs and RBDs is that RBDs are a success-oriented, while FTs are failure-oriented. Specifically, in an RBD, one works in the “success space” and thus looks at system success combinations, whereas in a fault tree one works in the “failure space” and thus looks at system failure combinations. In most cases, we may convert a fault tree to an RBD or *vice versa*. Particularly, the conversion is possible for all static coherent structures. In the conversion from a fault tree to an RBD, we start from the TOP event of the fault tree and replace the gates successively. A logic AND-gate is replaced by a parallel structure of the inputs of the gate, and an OR gate is replaced by a series structure of the inputs of the gate. In the conversion from an RBD to a fault tree, a parallel structure is represented as a fault tree where all the input events are connected through an AND-gate, and a series structure is represented as a fault tree where all the input events are connected through an OR-gate.

The modelling capabilities of FTs and RBDs have been enhanced in order to support a wide range of scenarios. In non-coherent FTs new gates were introduced, e.g., XOR, NOR etc.. A non-coherent FT is characterized by inverse gates besides logic gates used in coherent fault trees. In particular, it may have Exclusive-OR and NOT gates. A non-coherent fault tree is used to describe failure behaviour of a non-coherent system, which can transit from a failed state to a good state by the failure of a component, or transit from a good state to a failed state by the repair of a component. Non-coherent systems are typically prevalent in systems with limited resources, multi-tasking and safety control applications. They are often used to accurately analyze disjoint events [19], dependent events [20], and event trees [6].

These relationships cannot be modelled by a RBD. Thus, FT are capable to capture more complex relations than RBDs. Similarly, there are some other enhancements to RBDs that are not available in FTA, i.e., Reliability Graphs. Hence, it is not always possible to convert all fault trees into equivalent RBDs and *vice versa*.

2.2.2.5 ADVANTAGES AND DISADVANTAGES OF FT

The main advantages of FTs rely on the graphical representation of the causes of occurrence of the TE, the possibility of investigating the weakness in the system and the possibility of using the methodology in the design phase in order to support the decision of the best system configuration.

As for RBD, disadvantages of FT rely on the fact that is not possible to tackle dependencies between the modelled elements, temporal logics, finite maintenance resources, etc.. Moreover measure of interest like reliability with repairs cannot be evaluated. In these case state-space methods should be used. Availability can be evaluated in the case the availability of components are first derived. However, also the availability of components is often evaluated by the mean of state-space methods.

2.3 STATE-SPACE MODELS

State-space methods are based on the stochastic process that define the behaviour of a system. In this section we give the definition of the most common stochastic processes used in Reliability Engineering. A more complete argumentation of the concepts described in this section can be found in [22-26].

Definition 2.3 (Stochastic Process). A stochastic process is a family of random variables $\{X(t)|t \in T\}$, defined on a given probability space.

The set of all possible values that random variables can take is called the *state space*. If the state space of a stochastic process is discrete, it is called a *discrete state process*; otherwise it is said a *continuous state-space*. A stochastic process can be continuous or discrete with respect to the parameter set T . It is continuous if T is continuous, otherwise it is said discrete.

Denoted with $F_n(x_1, x_2, \dots, x_n)$ the finite dimensional joint distribution of a stochastic process $\{X(t)|t \in T\}$, we have that if $F_n(x_1, x_2, \dots, x_n)$ satisfies:

$$F_n(x_1, x_2, \dots, x_n) = P\{X(t_1) \leq x_1, X(t_2) \leq x_2, \dots, X(t_n) \leq x_n\} = \prod_{i=1}^n P\{X(t_i) \leq x_i\}, \quad (2.7)$$

for $0 \leq t_1 \leq t_2 \leq \dots \leq t_n$, the stochastic process is called an independent process.

Although it is easy to study, most real life processes do have some dependencies among these random variables. The most important and most common one is the first-order dependency, which is known as Markov dependency.

Definition 2.4 (Markov Process). A stochastic process $\{X(t)|t \in T\}$ is called a Markov process if for any $t_0 < t_1 < t_2 < \dots < t_n < t$, the conditional distribution of $X(t)$ for given values of $X(t_0), X(t_1), \dots, X(t_n)$ depends only on $X(t_n)$; that is

$$P\{X(t) \leq x|X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \dots, X(t_0) = x_0\} = P\{X(t) \leq x|X(t_n)\}. \quad (2.8)$$

The next state of a Markov process may only depend on the current state. No information about the prior sequence of states visited could affect the next transition. If the state space is discrete, we call such a stochastic process a *Markov chain*. If the parameter t is continuous it is said a *Continuous time Markov chains* (CTMC), if discrete a *Discrete Time Markov Chain* (DTMC).

The transition behaviours are characterized by transition rates or transition probabilities for CTMC and DTMC, respectively.

In many practical problems, the time origin does not matter, i.e., only the time elapsed decides the chain behaviour. Such kinds of Markov chains are *time homogeneous*, which means:

$$P\{X(t) \leq x | X(t_n)\} = P\{X(t - t_n) \leq x | X(0) = x_n\}. \quad (2.9)$$

It is shown [27] that the sojourn time of a homogeneous continuous time Markov chain is exponentially distributed (*memoryless property*). The probability that the process stays in state i at time $t > t_n$ given it was in state i at time t_n only depends on state i , but does not depend on how much time it has spent in state i .

An extension of Markov process is that transition rates not only depend on the current state, but also the duration the process spends in that state may depend on the particular transition. The Markov property still holds at the time of entry (exit) to (from) a states. Such a process is called a *semi-Markov process*. Before defining SMP we need to introduce some fundamental stochastic processes.

Definition 2.5 (Renewal Sequence and Renewal Process). $\{S_n, n \geq 0\}$ is said to be a renewal sequence and $\{N(t), t \geq 0\}$ a renewal process generated by $\{X_n, n \geq 0\}$ if $\{X_n, n \geq 0\}$ is a sequence of independently and identically distributed non-negative random variables, where:

$$S_n = X_1 + X_2 + \dots + X_n, n \geq 1, \quad (2.10)$$

$$S_0 = 0, \quad (2.11)$$

$$N(t) = \sup \{n \geq 0: S_n \leq t\}. \quad (2.12)$$

The random variable X_n is the time interval between the successive arrivals $n - 1$ and n . The same (probabilistically exact) process is repeated at each time epoch S_n . S_n is the absolute time of the n -th arrival. $N(t)$ is the total number of arrivals at time t .

Definition 2.6 (Markov Renewal Sequence and Markov Renewal Process).

$\{(Y_n, S_n), n \geq 0\}$ is said to be a Markov renewal sequence with state space I if for all $n \geq 0$ and $i, j \in I$, the following property holds:

$$\begin{aligned} P\{Y_{n+1} = j, S_{n+1} - S_n \leq x | Y_n = i, S_n, Y_{n-1}, S_{n-1}, Y_0, S_0\} &= \\ &= P\{Y_{n+1} = j, S_{n+1} - S_n \leq x | Y_n = i\} = \\ &= P\{Y_1 = j, S_1 \leq x | Y_0 = i\}. \end{aligned} \tag{2.13}$$

The stochastic process $N(t) = (N_j(t), j \in I)$ is a Markov renewal process, where:

$$N_j(t) = \sum_{n=1}^{N(t)} Z_j(n), \tag{2.14}$$

$$Z_j(n) = \begin{cases} 1, & \text{if } Y_n = j \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

$$N(t) = \sup \{n \geq 0 : S_n \leq t\}. \tag{2.16}$$

$N_j(t)$ is the number of times state j is visited by time t , $N(t)$ is total number of state changes at time t . In Markov renewal process the processes over each interval are not independent, but have one-order dependency. In this case the future evolution of the stochastic process depends on the current state of the process at Markov renewal points. Thus, in Markov renewal process, we are only interested in the states changes at time epochs S_n 's.

Definition 2.7 (Semi-Markov Process). Given a Markov renewal sequence $\{Y_n, S_n\}$ with state space I , the stochastic process $\{Z(t), t \geq 0\}$ is called a semi-Markov process with state space I if $Z(t) = Y_n$ for $t \in [S_n, S_{n+1})$. In a SMP the sample path is piecewise constant and right continuous. Jumps only happen at the Markov renewal points. The inter-arrival times $(S_n - S_{n-1})$ are generally distributed.

Definition 2.8 (Markov Regenerative Process). A stochastic process $\{Z(t), t \geq 0\}$ is called a Markov regenerative process (MRGP) if there exists a Markov renewal sequence $\{(Y_n, S_n), n \geq 0\}$ of random variables such that all the conditional finite dimensional distributions of $\{Z(S_n + t), t \geq 0\}$ given $\{Z(u), 0 \leq u \leq S_n, Y_n = j\}$ are the same as those of $\{Z(t), t \geq 0\}$ given $Y_n = i$.

A Markov regenerative process is also constructed from the Markov renewal process (as SMP). Each S_n is a Markov regenerative point because the stochastic process evolution from that point on is independent of the history before it.

The difference between SMP and MRGP is that in SMP no state change occurs between successive Markov regenerative points, but for Markov regenerative process, the stochastic process between S_n and S_{n+1} could be any continuous-time stochastic process. Hence, in MRGP the sample paths are no longer piecewise constant.

Like semi-Markov processes, Markov regenerative process allows non-exponentially distributed firing time transitions, but it is an even more general process. If a transition t is the only transition out of state i , t is called an *exclusive* transition. A transition t is said to be *competitive* with respect to another transition t' if both t and t' can occur in state i and the firing of t disables t' . If the firing of t does not disable t' , t is said to be *concurrent* with t' . Semi-Markov chains do not allow concurrent transitions, while the MRGP just defined can have all these types of transitions.

Definition 2.9 (Generalized Semi Markov Process). A stochastic process $\{Z(t), t \geq 0\}$ is called a Generalized Semi Markov Process (GSMP) if there is not any restriction on the kind of distribution associated with the sojourn time in a state and there can be more than one concurrent transition enabled in a state.

Strictly speaking, GSMPs are not Markov chains because they lack the *memoryless* property. In [28] each state of a GSMP is characterised as a set of *active elements*, each of which has an associated *lifetime*. When an active element completes a state change occurs but the residual lifetime of all the interrupted elements, if still enabled, are maintained.

Analytical treatment of GSMP becomes viable under restrictions on number of concurrent enabled non-exponential transitions. Under the *enabling restriction*, which assumes that at most one generally distributed transition is enabled in any state, activity cycles of generally distributed transitions never overlap. In this case, the model underlies a Markov Regenerative Process (MRGP) which regenerates at

every change in the enabling status of non-exponential timed transitions [29,30,31,32].

The analysis of a model with multiple concurrent generally distributed transitions has been formulated with the introduction of *supplementary variables* [32,33,34]. In this theory the logical state is extended. It is a vector that record of ages of generally distributed enabled transitions. However, practical solution is limited to one or two concurrently enabled non-exponential distributions, thus falling again within the limits of the enabling restriction typical of MRGP [35]. Some work where the limit of the enabling restriction is overcome can be found in [36,37] but only for the case where all timed transitions are either exponential or deterministic. In [38,39,40] state classes are used to manage the case of multiple concurrent generally distributed transitions. In [41] the approach is extended in order to support derivation of continuous time transient probabilities. However, in these works only expolynomial distributions are considered.

In the most general settings GSMP are used as a mean to define Discrete Event Simulation (DES) models [42]. In fact, the above mentioned methods, beside the limitations of concurrently enabled transitions as well as the limitations of the kind of supported distributions, suffer from computational limits that make the solution possible only for small systems.

2.4 CONCLUSION

In this chapter we have described some of the most known methodologies used in Reliability Engineering. Non-state space models as RBD and FT are probably the most well known methodologies in reliability evaluation. They are simple to use and allow to specify a system in term of its requirements. As we have seen, however, their application is limited to systems with the following characteristics:

- components must be independent;
- components can be only in two possible states, working or failed;

- stand-by policies, shared loads, finite resource maintenance, detection systems cannot be modelled;
- failure logics are limited to the kind of relations admitted by the formalisms;
- it is not possible to evaluate measure of interest like reliability with repair and availability;
- time of events and temporal ordering are not explicated in the formalism.

State-space models, on the other hand, overcome the limits of non state-space models. They allow to model complex systems and evaluate various kind of measure of interest.

However, Markov Chains and generalizations are *flat* structures that are difficult to design and understand. Many complicated systems result in Markov chains that are over a few million states in size; it is impossible to derive a Markov chain by hand. Therefore, Markovian models are typically constructed from some other high-level formalism.

The hybrid formalism presented in the next sections try to alleviate this problem. We will describe two kind of formalism. The ones derived from non state-space models like Dynamic Fault Tree (DFT) extend the modelling capabilities of FTs and use a state-space based low level representation for the solution of the model.

On the other hand, methodologies like stochastic extensions of Petri nets use an approach similar to the one used in state-space models but abstract these models by the mean of objects that allow the composition of reduced models obtained for the different parts of the system.

Moreover, since Petri nets allow to solve a system via simulation, they are able to overcome problems of state-space models as the know issue of the explosion of the state-space and the possibility to solve systems whose underlying process fall in the class of GSMP.

BIBLIOGRAPHY

- [1] Murphy, K.E. & Carter, C.M., 2003. *Reliability Block Diagram Construction Techniques: Secrets to Real-Life Diagramming Woes*. In Proceedings Annual Reliability and Maintainability Symposium-Tutorial Notes. Tampa, Florida.
- [2] Sahinoglu, M., Ramamoorthy, C.V., Smith, A.E. & Dengiz, B., 2004. *A Reliability Block Diagramming Tool to Describe Networks*. In IEEE, ed. Reliability and Maintainability Symposium. Los Angeles.
- [3] Birolini, A., 2003. *Reliability Engineering*. Springer.
- [4] Watson, H.A., 1962. *Launch Control Safety Study*. BELL Telephone Laboratories. WG 10.4.
- [5] Dugan, J.B., Bavuso, S.J. & Boyd M.A., 1990. *Fault Trees and sequence dependencies*. In: Proceedings of the annual reliability and maintainability symposium; 286-93.
- [6] Codetta-Raiteri, D., 2011. *Integrating several formalisms in order to increase Fault Trees' modeling power*. Reliab Eng Syst Safety.
- [7] Codetta-Raiteri, D. & Portinale, L., 2010. *ARPHA: an FDIR architecture for Autonomous Spacecrafts based on Dynamic Probabilistic Graphical Models*. TECHNICAL REPORT TR-INF-2010-12-04-UNIPMN.
- [8] Lee, W.S., Grosh, D.L., Tillman, F.A. & Lie, C.H., 1985. *Fault Tree Analysis, Methods, and Applications - A Review*. IEEE Transactions on Reliability; 194-203.
- [9] Misra, K.B., 1992. *Reliability analysis and prediction: a methodology oriented treatment*. Elsevier, Amsterdam.
- [10] Shooman, M.L., 1990. *Probabilistic reliability: an engineering approach (2nd Edition)*. McGraw- Hill, New York.
- [11] Vesely, W.E., Goldberg F.F., Roberts N.H. & Haasl D.F., 1981. *Fault tree handbook*. U.S. Nuclear Regulatory Commission, Washington DC.

- [12] Dugan, J.B. & Doyle, S.A., 1997. *New results in fault-tree analysis*. Tutorial Notes of the Annual Reliability and Maintainability Symposium.
- [13] NASA, 2002. *Fault tree handbook with aerospace applications*. NASA Office of Safety and Mission Assurance, Washington DC.
- [14] Henley, E.J. & Kumamoto, H., 1992. *Probabilistic risk assessment*. IEEE Press, New York.
- [15] Chang, Y.R., Amari, S.V. & Kuo, S.Y., 2005. *OBDD-based evaluation of reliability and importance measures for multistate systems subject to imperfect fault coverage*. IEEE Transactions Dependable and Secure Computing; 2(4): 336-347.
- [16] Kuo, S., Lu, S. & Yeh, F., 1999. *Determining terminal-pair reliability based on edge expansion diagrams using OBDD*. IEEE Transactions on Reliability; 48(3): 234-246.
- [17] Xing, L. & Dugan, J.B., 2002. *Analysis of generalized phased-mission systems reliability, performance and sensitivity*. IEEE Transactions on Reliability; 51(2): 199-211.
- [18] Xing, L., 2004. *Fault-tolerant network reliability and importance analysis using binary decision diagrams*. Proceedings of the 50th Annual Reliability and Maintainability Symposium, Los Angeles, CA.
- [19] Yeh, F., Lu, S. & Kuo, S., 2002. *OBDD-based evaluation of k-terminal network reliability*. IEEE Transactions on Reliability; 51(4): 443-451.
- [20] Zang, X., Sun, H. & Trivedi, K.S., 1999. *A BDD-based algorithm for reliability analysis of phased mission systems*. IEEE Transactions on Reliability; 48(1): 50-60.
- [21] Zang, X., Wang, D., Sun, H. & Trivedi, K.S., 2003. *A BDD based algorithm for analysis of multistate systems with multistate components*. IEEE Transactions on Computers; 52(12): 1608-1618.

- [22] Nielsen, S.F., 2009. *Continuous-time homogeneous Markov chains*. Copenhagen: University of Copenhagen - Department of Mathematical Sciences.
- [23] Boudewijn, R. Haverkort, 2002. *Markovian Models for Performance and Dependability Evaluation*, Lectures on formal methods and performance analysis. Springer Verlag, New York.
- [24] Bause, F. & Kritzinger, S.P., 1996. *Stochastic Petri nets - an introduction to the theory*. Advanced Studies in Computer Science. Vieweg Verlagsgesellschaft.
- [25] William J. Stewart., 1994. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 41 William Street, Princeton, New Jersey 08540.
- [26] Cinlar, E., 1969. *Markov Renewal Theory*. Advances in Applied Probability, 1(2): 123-187.
- [27] Sahner, R.A., Trivedi, K.S. & Puliafito, A., 1995. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers.
- [28] Hillston, J. & Pooley, R. *Stochastic Process Algebras and their Application to Performance Modelling*. URL:
<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.819>
- [29] Choi, H, Kulkarni, V. G. & Trivedi, K.S., 1994. *Markov Regenerative Stochastic Petri Nets*. Perf. Eval.; 20: 337-357.
- [30] Ciardo, G., German, R. & Lindemann, C., 1994. *A characterization of the stochastic process underlying a Stochastic Petri Net*. IEEE Trans. Software. Eng.: 20(7): 506-515.
- [31] Bobbio, A. & Telek, M., 1995. *Markov Regenerative SPN with non-overlapping activity cycles*. Int. Computer Performance and Dependability Symp. - IPDS95; 124-133.

- [32] German, R. & Lindemann, C., 1994. *Analysis of Stochastic Petri Nets by the method of supplementary variables*. In Performance Evaluation; 20:317-335.
- [33] Cox, D., 1955. *The analysis of non-markovian stochastic processes by the inclusion of supplementary variables*. Proceedings of the Cambridge Philosophical Society; 51:433-440.
- [34] Telek, M. & Horvath, A., 1998. *Supplementary variable approach applied to the transient analysis of age-MRSPNs*. In Proc. Int. Performance and Dependability Symp.; 44-51.
- [35] German, R. & Telek, M., 1999. *Formal relation of Markov renewal theory and supplementary variables in the analysis of Stochastic Petri Nets*. In Proc. 8th International Workshop on Petri Nets and Performance Models; 64-73.
- [36] Lindemann, C. & Schedler, G.S., 1996. *Numerical analysis of Deterministic and Stochastic Petri Nets with concurrent deterministic transitions*. Performance Evaluation, vol. 27-28: 565-582.
- [37] Lindemann, C. & Thuemmler, A., 1999. *Transient analysis of Deterministic and Stochastic Petri Nets with concurrent deterministic transitions*. Performance Evaluation, vol. 36-37:35-54.
- [38] Bucci, G., Piovosi, R., Sassoli, L. & Vicario, E., 2005. *Introducing probability within state class analysis of dense time dependent systems*. Proc. of the 2nd Int. Conf. on the Quant. Evaluation of Sys.(QEST).
- [39] Carnevali, L., Sassoli, L. & Vicario, E., 2009. *Using stochastic state classes in quantitative evaluation of dense-time reactive systems*. IEEE Trans. on Soft. Eng.
- [40] Horvath, A. & Vicario, E., 2009. *Aggregated stochastic state classes in quantitative evaluation of non-markovian stochastic Petri nets*. In Proc. of 6th International Conference on the Quantitative Evaluation of Systems (QEST), Budapest, Hungary.
- [41] Horvath, A. Ridi, L., & Vicario, E., 2010. *Transient analysis of generalised semi-Markov processes using transient stochastic state classes*. Seventh

International Conference on the Quantitative Evaluation of Systems
Williamsburg, VA, USA.

- [42] Glynn, P. W., 1989. *A GSMP formalism for discrete-event systems*. Proceedings of the IEEE; 77:14–23.

CHAPTER 3

3.1 INTRODUCTION

The two formalism introduced in Chapter 2, RBD and FT, although well known and widely used in Reliability Engineering, are limited in their modelling capabilities. They cannot be developed to model systems with interactions, stochastic associations or sequential relationships. Examples are: load-sharing, standby redundancy, interferences, dependencies, common cause failures, etc.

These lacks in system reliability modelling notations have awakened the scientific community to the need of new formalisms. An approach that has been adopted is to extend the existing formalisms with new elements to model the (uncovered) aspects. This resulted in the creation of hybrid formalisms that make use of a high level model of description that is then converted in a state-space based model.

In this chapter we review two class of hybrid formalisms: the ones that are an extension of non state-space models, i.e., FT and RBD, and the ones that are defined as stochastic extensions of Petri nets.

Dynamic Fault Trees (DFTs) were one of the first methodology that was developed in the context of hybrid modelling [1,2,3]. DFT extend FT to enable modelling of time dependent failures by introducing new dynamic gates and elements. DFTs were introduced due to the fact that the traditional static fault trees

with AND, OR, and Voting (k-out-of-N) gates cannot capture the dynamic behaviour of system failure mechanisms. In order to include these kind of behaviours in the FT terminology, dynamic gates were introduced, generating the DFT formalism. In their first formulation DFT were solved via conversion in CTMCs, but in recent literature other approaches have been reported [4-12].

In the same way RBD were also extended into Dynamic RBD (DRBD) [13]. DRBD formalize the concepts of state, event and dependence, providing a logic infrastructure to define several dynamic reliability behaviours. Many similarities link DFT to DRBD, but, at the same time, one of the aims of DRBD is to extend the DFT capabilities in dynamic behaviour modelling. In fact, many conditions modelled by DRBD elements do not have a correspondent in the DFT domain.

Another methodology that has been increasing its popularity is Boolean Driven Markov Process (BDMP) [14]. This approach makes use of an extended FT at the high level that include new kind of objects called triggers. Moreover the leafs of the tree are modelled by CTMC extending the modelling capabilities of the modes of failure of components.

The direct specification of a CTMC at the level of individual states and state-to-state transitions is tedious and error prone and therefore only feasible for very small models. This motivated researchers to develop high level specification formalisms for defining Markovian models at a level of abstraction which is more convenient for the human modeller. The most popular of these formalisms are stochastic Petri nets [15].

Stochastic Petri nets (SPN) were developed in the 1980s for modelling complex synchronisation schemes. The modelling primitives of Petri nets (places, transitions, markings) are very basic and do not carry any application specific semantics. For that reason Petri nets are universally applicable and very flexible as it is reflected by their successful application in different areas.

In the class of generalised SPNs (GSPN) transitions are either timed or immediate [16]. Timed transitions are associated with an exponentially distributed firing time, while immediate transitions fire as soon as they are enabled. During the analysis of a

GSPN the “Reachability Graph” is generated and the so called “vanishing markings”, which are due to the firing of immediate transitions, are eliminated. The result is a CTMC whose analysis yields steady-state or transient-state probabilities, i.e., the probabilities of the individual net markings from which high level measures can be computed.

With the help of high-level model specification formalisms considered so far it is possible to specify larger CTMCs than at the state level, but these formalisms do not support the concepts of modularity hierarchy or composition of sub-models. As a result the models are monolithic and may be difficult to understand and debug. Moreover state space generation and numerical analysis of very large monolithic CTMCs is often not feasible in practice due to memory and CPU time limitations which is referred to as the notorious state space explosion problem.

In the basic GSPN formalism a model consists of a single net which covers the whole system to be studied. Therefore GSPN models of complex systems tend to become very large and confused and suffer from the state space explosion problem. Stochastic activity networks (SAN) constitute an approach to the structuring of GSPNs through the sharing of places between different subnets [17]. In the presence of symmetric sub-models they tackle the state space explosion problem by directly generating a reduced Reachability Graph in which all mutually symmetric markings are collapsed into one.

The remainder of this chapter is organized as follow. In Section 3.2 we give a brief introduction of Dynamic Fault Trees and related methodologies like DRBD and BDMP. In Section 3.3 stochastic extension of Petri nets are introduced with particular attention to the description of the SAN formalism. Advantages and disadvantages of the revised methodologies are discussed and finally in Section 3.4 we report some conclusion.

3.2 DYNAMIC FAULT TREES

Traditional FT cannot capture the dynamic behaviour of the system failure mechanisms associated with sequence-dependent events, spares and dynamic

redundancy management, and priorities of failure events. For this reason, many modellers turned to Markov chains for reliability assessment of safety-critical systems. In addition to their computational complexity, a major disadvantage of Markov chains is that the correct Markov model for a given system is difficult to determine. In order to overcome this difficulty, the concept of Dynamic Fault Trees is introduced by adding sequential notion to the traditional fault tree approach [1,2,3].

In practice, the failure criteria of a system may depend on both the combinations of fault events and sequence of occurrence of input events. This is done by introducing dynamic gates into fault trees. With the help of dynamic gates, modellers can specify the system sequence-dependent failure behaviour using dynamic fault trees that are compact and easily understood.

The modelling power of DFT has gained the attention of reliability engineers working on safety critical systems. Therefore, these gates are not only used in research-oriented projects, but also recently introduced in commercial tools for fault tree evaluation, *eg*, Relex [18].

3.2.1 DFT OBJECTS AND ASSUMPTIONS

A DFT is a stochastic model for the reliability evaluation that synthesizes the ways how an undesired and time dependent event can occur. As a Fault Tree (FT), a DFT is composed by a top gate which represents the most undesired event (TE, top event) and a certain number of lower level gates and basic events (BEs) that, combined according with the logic of the fault scenario, cause the occurrence of the TE.

The main hypotheses for the use of the DFT are that (i) events are binary and (ii), according to many authors [1,2,3], components are not repairable. Thus, the main difference with FT is that DFT were not conceived to evaluate availability but are appropriate to evaluate the reliability of model characterized by complex stochastic dependencies. The possibility to model complex interactions with the graphical symbolism of the FT has encouraged the development of dynamic models but, in point of the fact, DFT has shown many issues for what concern their resolution.

The reason of this anomaly has to be traced in the lack of a rigorous semantic language that has caused the proliferation of several and variegated techniques of resolution that resort to an equivalent stochastic model [1-12]. At the state of the art, an analytical solution exists only if another hypothesis is added to the previous ones: BEs have to be described by the exponential distribution. In this way, it is possible to convert a DFT into a state-space model and solve it within the domain of the Markov processes.

Unfortunately the mentioned hypotheses can result too restrictive, especially for real industrial applications characterized not only by exponentially distributed time to fail but also by Weibull, Gaussian or lognormal probability distribution. Therefore, the reliability evaluation of systems that present generalized functions of probability is not possible with the analytical Markov processes and, at the state of the art, the more effective solution is the simulation [9,19].

In general, all the previous techniques of resolution (analytical and simulative) have been implemented in several software applications for reliability analysis [5,6,8,19] but, despite that, the real effectiveness of these tools is still questionable because none of them can be used to design and solve “complex DFT” in a straightforward manner. A synthesis of the main features of some automated tools that we have tested can be found in Appendix A.

3.2.2 DYNAMIC GATES

In this section we give a brief description of the dynamic gates of DFTs (Figure 3.1).

3.2.2.1 FUNCTIONAL DEPENDENCY GATE

A functional dependency gate (FDEP) consists of a trigger event and a set of dependent events. The dependent basic events are functionally dependent on the trigger event. When the trigger event occurs, the dependent basic events are forced to occur. The separate occurrence of any of the dependent basic events has no effect on the trigger event. The output of the gate is a dummy, i.e., it is not taken into account in the calculation of the system failure probability. While the trigger can be any subsystem, i.e., a construct made of gates, originally the dependent events could be

only basic events. In [19] is proposed to extend the formalism allowing the triggering of any gate and not only BEs.

Name	Graphical Representation	Description (N input)
SPARE		It triggers only after the primary if all the N spares occur. Spares can be shared with other spare gate.
PAND		It behaves like an AND gate but it triggers only if the input events occur in the order from the left to the right.
SEQ		It forces the input events to occur from the left to the right order. It can model the gradual degradation of a system.
FDEP		This gate models the failure of the dependent input events if the primary occurs. The output is a dummy.

Fig. 3.1 DFT dynamic gates.

3.2.2.2 SPARE GATE

Systems with cold and warm spares cannot be modelled exactly using usual FTs because the system failure criteria cannot be expressed in terms of combinations of basic events, all using the same time frame.

The Spare gate has one primary input and one or more alternate inputs called *spares*. The primary input of a Spare gate is initially powered on and the alternate inputs are in standby mode. When the primary fails, it is replaced by the first available alternate input that switches from the standby mode to the active mode. In turn, when this alternate input fails, it is replaced by the next available alternate input, and so on.

In standby mode, the failure rate λ of BEs is reduced by a *dormancy factor* $\alpha \in [0,1]$. Thus, the failure rate in standby mode is $\alpha \cdot \lambda$, while in the active mode it switches back to λ . Depending on the value of α we can distinguish the three different situations:

- $\alpha = 0$, the spare is called *cold spare* and cannot fail before it becomes active;
- $\alpha = 1$, the spare is called *hot spare* and can fail in both the stand-by and active state with the same failure rate;
- $0 < \alpha < 1$, the spare is called *warm spare* and can fail before it becomes active, but with a “scaled” failure rate.

SPARE gates fail when the primary and all its spares have failed or are unavailable, i.e. used by other SPARE gates. In [19] is proposed to extend the formalism allowing spare events to be any subsystem.

3.2.2.3 PRIORITY AND GATE

The PAND gate models a failure sequence dependency. The priority-AND (PAND) gate is an AND gate with an additional condition: events must occur in a specific order [1,2,3]. The output of the gate is true if all the events have occurred in the left-to-right order in which they appear under the gate. If the inputs fail in a different order, the gate does not fail. Generally the gate accepts only two inputs and cascade of gates are used to represent the temporal condition on more inputs.

3.2.2.4 SEQUENCE ENFORCING GATE

The SEQ gate forces events to occur in a particular order. The input events are constrained to occur in the left-to-right order in which they appear under the gate. The sequence enforcing gate can be contrasted with the priority-AND gate in that the priority-AND gate *detects* whether events occur in a particular order (the events can occur in any order), whereas the sequence enforcing gate *allows* the events to occur only in a specified order. With the respect to the SPARE gate the SEQ gate can be seen as a SPARE gate with one primary and cold stand-by inputs.

3.2.3 SOLUTION METHODS

Fault trees with dynamic gates are typically solved by automatic conversion to equivalent Markov models [1,2,3]. In some cases, Monte Carlo simulation can be used to solve dynamic fault trees without converting to Markov models [9,19]. Once dynamic fault trees are converted to Markov models, the Markov models can be solved for state probabilities.

In order to reduce the number of states in the retrieved Markov Chain many algorithms based on modularization of the DFT have been proposed [8,20]. These methods aim to find independent sub-trees from a point of view of repeated events and dynamic gates. If an independent sub-tree contains a dynamic gate, then it will be solved using a Markov model; otherwise, it will be solved using BDD, and their solutions will be integrated to get the solution for the entire fault tree.

Other approaches use conditional probabilities to calculate the results of a dynamic module without generate the Markov Chain [4].

Another approach is to convert the DFT into a dynamic Bayesian network (DBN) [6,10,11]. With respect to CTMC, the use of a DBN allows one to take advantage of the factorization in the temporal probability model. As a matter of fact, the conditional independence assumptions implicit in a DBN enable a compact representation of the probabilistic model, allowing the system designer or analyst to avoid the complexity of specifying and using a global-state model (like a standard Markov Chain); this is particularly important when the dynamic module of the considered DFT is significantly large.

Other approaches to the resolution of DFT with the use of Probabilistic Boolean Algebra were proposed in [21]. A conversion of DFT into Generalized Stochastic Petri Nets was proposed in [12]. Finally, approaches based on Stochastic Process Algebra can be found in [7].

3.2.4 RELATED WORK

Other hybrid formalism that make use of both non state-space and state-space representation have been proposed. Among these we give a brief introduction of DRBD and BDMP.

3.2.4.1 DYNAMIC RELIABILITY BLOCK DIAGRAM

DRBDs [13] inherit the features of RBD in reliability modelling such as simplicity, versatility and expressive power and extend the formalism allowing taking into account the system dynamics. To this end in DRBD each component can be in three different states: active, i.e., working, failed, i.e., not operational, and

stand-by, i.e., reliable but not available. DRBD extend the capabilities of DFTs because of the use of state-space models at the high level model. However, the state-space model is limited to a specific structure (only three states are admitted), thus their application is limited by the formalism itself.

3.2.4.2 BOOLEAN DRIVEN MARKOV PROCESS

BDMP [14] extends FT with two new objects: triggers and Markov chains. Triggers are used to widespread failures of BEs or gates across the tree. To this end triggers carry on a Boolean value that forces BEs to behave differently according to the value of the trigger. In fact, BEs may have two modes. These two modes are represented by two different Markov chains. Only one Markov chain can be active at any time and the one that is selected to be active depends on the value of the trigger related to the BE. Modelling capabilities of BDMP extend the ones of DFT in that is possible to use Petri nets as the leafs of the tree. The main drawback however relies in the Boolean selector itself. In fact, only two processes are possible (depending on the Boolean values 1 or 0) while it could be necessary to use more than two processes to describe the behaviour of components, e.g., load sharing with multiple components (more than 2).

3.2.5 ADVANTAGES AND DISADVANTAGES OF DFT

Perhaps the main advantage of DFT is that of being a high level modelling formalism that, extending FT, results intuitive and easy to use. Although they do not cover all the possible scenarios, DFT have been referred ad a reference methodology by many researchers. Limitations of DFT have been addressed earlier in this chapter. Here we report the most meaningful:

- only exponential distributions can be used to define the time to failure of components;
- components are not repairable, thus limited measure of interest can be evaluated;
- complex redundancy and maintenance management, load sharing systems, etc. cannot be modelled.

During this doctoral course the main objectives have been to alleviate these problems. We have dealt with the issue of the kind of distribution that can be used to model components developing a software tool, MatCarloRe, to evaluate DFT via simulation. The tool will be presented in Chapter 4 while two published papers that describes the tool and compare it with other academic and commercial tool are presented in Appendix A. Moreover, in appendix A is reported another published paper where analytical and simulative resolution techniques are compared. The paper describes also the approaches used in modularization and introduces a classification based on weak and strong hierarchy.

To alleviate other problems, we have taken an approach similar to the idea of BDMP, but that is even more general. The idea again is to attach one or more state-space model to the leafs of the tree, but in our case we do not model dependencies via triggers but by an appropriate model of transitions of these state-space models. Adaptive Transition Systems (ATS) are even more general in that there can be several state-space models attached to any leaf. The behaviour of the system is entirely captured by these state-space models and thus the FT represents only a particular specification of a *reward* (a measure of interest) that we want to evaluate. For instance, temporal logics, e.g., Continuous Stochastic Logic (CSL) [22], can be used to specify the property of interest. ATS will be presented in Chapter 5. In Chapter 7 we present the application of ATS to evaluate repairable DFT. To our knowledge there is not at the moment any work where availability or reliability with repair of DFTs have been investigated exhaustively. An approach based on Stochastic Process Algebra can be found in [7].

3.3 STOCHASTIC EXTENSION OF PETRI NETS

Stochastic models have been extensively developed in the areas of performance and dependability evaluation. High level model specification formalisms have been developed in order to overcome the problems that arise when specifying the model at the level of the Markov chain, i.e., tedious and error prone. Stochastic extensions of Petri nets are a well know formalism that belong to this class.

Petri nets are abstract formal methods [23], for the description and analysis of flow of information and control in concurrent systems. Petri nets are graphically represented as collections of:

- **Places**, which are represented by circles. Places model state variables and can contain *tokens*.
- **Tokens**, are represented as black dots and represent the specific value of the state variables.
- **Transitions**, are drawn as rectangles. They model activities which cause a change in state.
- **Arcs**, are arrows between *places* and the *transitions* used to specify the interconnection between the two types of objects.

The graphical aspect of these models are attractive for practical modelling since they help in understanding how features of the real system are conveyed in the model.

Definition 3.1 (Petri Net). A Petri net is a 5-tuple $PN = (P, T, I, O, m(0))$ where:

- $P = (p_1, p_2, \dots, p_P)$ is the set of places;
- $T = (t_1, t_2, \dots, t_T)$ is the set of transitions;
- $I, O: (P \times T) \rightarrow N$ are the backward and forward incidence functions, i.e., define the arcs that connect places and transition and the assigned weights; and
- $m(0) = (m_1(0), m_2(0), \dots, m_P(0))$ is the initial marking, i.e., the number of tokens contained by places.

Transitions can be *enabled*, which means that they can *fire*. *Firing* means that the transition removes from its input places a number of tokens, defined by the *weight* of the input arc. It also adds to its output places the number of tokens defined by the *weight* of the output arc. A transition is enabled if all its input places are marked with as many tokens as specified by the backward incidence function.

The dynamic behaviour of the net is specified by its marking behaviour. A marking is an assignment of tokens to places. Classic PNs are independent of time and are characterised by the *nondeterministic* firing of transitions that are simultaneously enabled in a given *marking* (Figure 3.2). The marking of a Petri net determines the state of the Petri net.

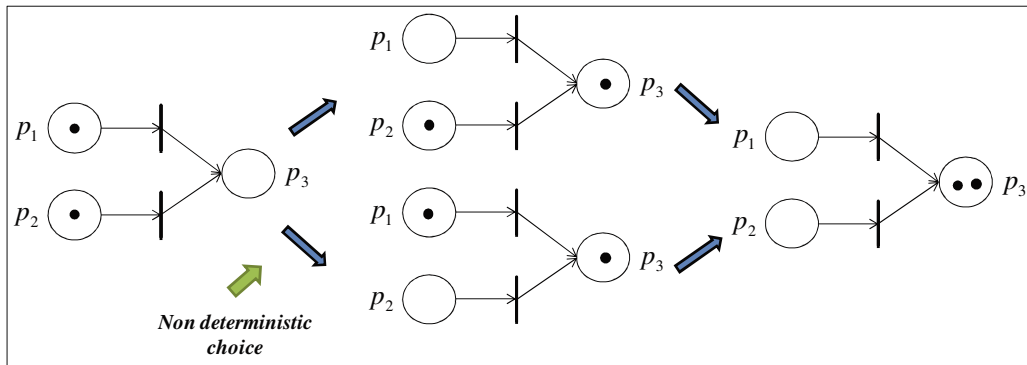


Fig. 3.2. An example of non-determinism in Petri nets.

The most common way to analyse the behaviour of a PN is by the construction of the *Reachability Graph* (RG). It allows to build a state-space model of a PN. The Reachability Graph is built by determining all the markings which result from the different transitions firing. The state-space can be finite or not depending on the fact that the net is bounded or not (Figure 3.3).

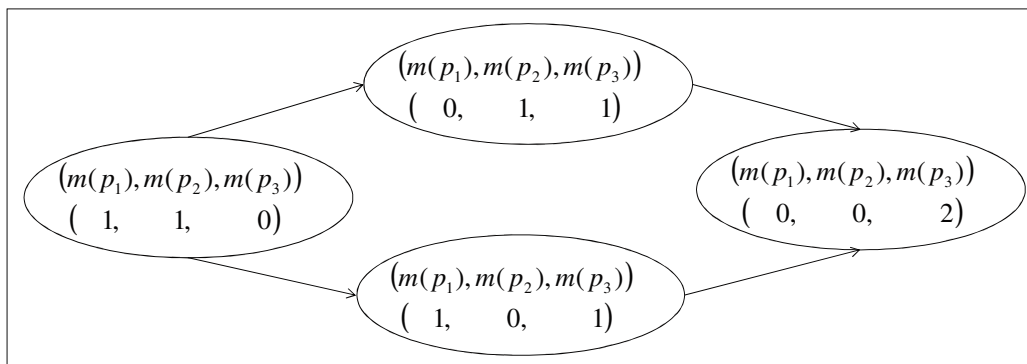


Fig. 3.3. The Reachability Graph of the PN in Figure 3.2.

3.3.1 STOCHASTIC PETRI NET CLASSIFICATION

In this section we present an overview of the some of the most know stochastic extensions of Petri nets.

3.3.1.1 STOCHASTIC PETRI NETS

Stochastic PN (SPN) are timed PN in which all the firing delays are exponentially distributed. The use of exponential distributions for the temporal specifications results in a PN that can be mapped on continuous-time Markov chains [15].

SPNs are formally defined by adding the set $\Lambda = (\lambda_1, \lambda_2, \dots, \lambda_T)$ to the definition of a Petri net. λ_i is the transition rate of the transition t_i . Since transition delays are exponential, λ_i is also the parameter of the exponential distribution governing the firing delay of t_i .

If two transitions are enabled at the same time, the transition that fires first will have the minimum delay. This is also known as the *race condition*. If t_1 and t_2 are both enabled, the probability that t_1 will fire first is given by $\frac{\lambda_1}{\lambda_1 + \lambda_2}$ (Figure 3.4).

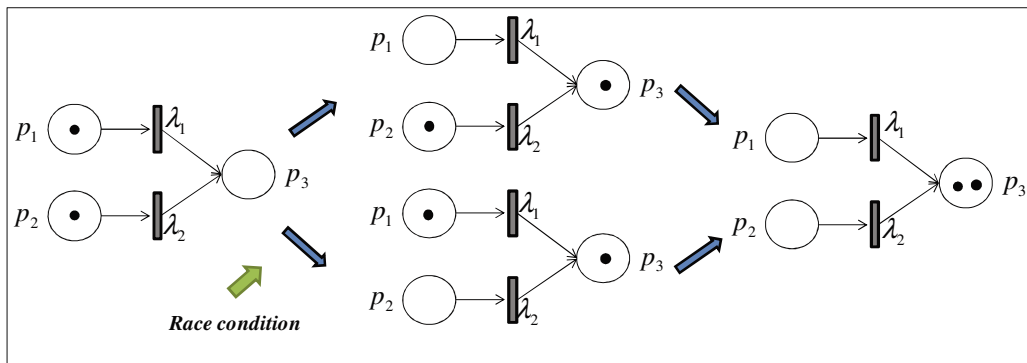


Fig. 3.4. An example of possible evolution of a SPN.

The Reachability Graph of a SPN is a CTMC if rates of transitions in the SPN are assigned to the transition between states in the reachability graph. The stochastic process underlying a SPN is a CTMC and can be solved with normal numerical methods.

3.3.1.2 GENERALIZED STOCHASTIC PETRI NETS

GSPNs are SPNs with the introduction of immediate delays [16]. In GSPN there are two types of transitions, *immediate* and *timed*. Immediate transitions fire in zero time once enabled, while timed transitions fire in a delay time defined by an exponential distribution.

To overcome the problem that arise in case of concurrent enabled immediate transitions a priority rate is assigned to each immediate transition. Immediate transitions have priority over timed transition.

The use of immediate transitions results in a more expressive PN but with little change to the analysis of the underlying CTMC. In fact, GSPN are analysed by considering two kinds of markings: *vanishing* and *tangible*. Vanishing markings are those in which the system spends zero time, i.e., markings that involve immediate transitions. On the other hand, markings that involve timed transitions are known as tangible.

The Reachability Graph of a GSPN is built considering first the *Extended Reachability Graph* (ERG) in which both vanishing and tangible marking are present. Successively, the Reachability Graph is obtained eliminating vanishing markings from the state-space model (Figure 3.5).

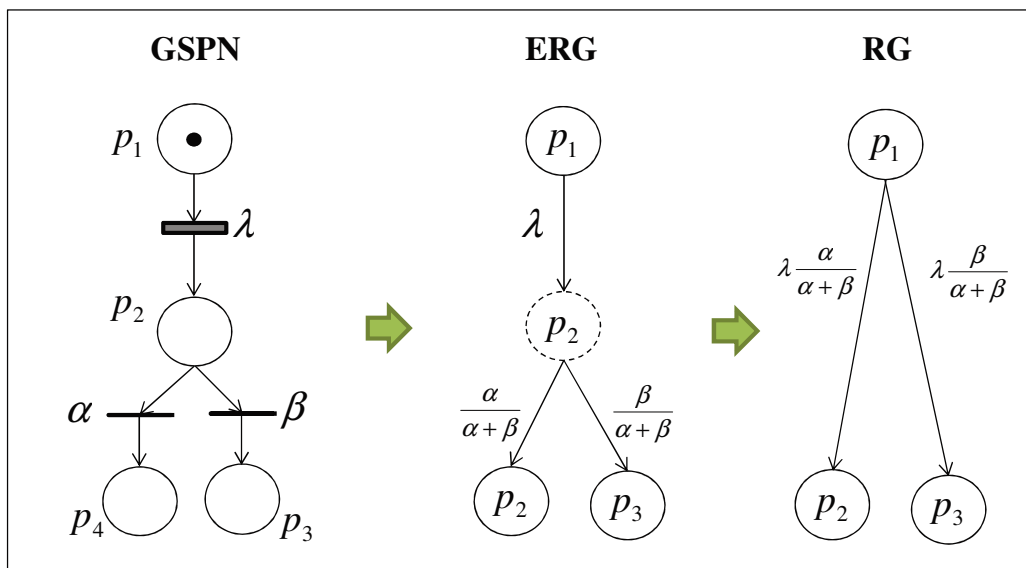


Fig. 3.5. An example of Reachability Graph construction of GSPN.

3.3.1.3 STOCHASTIC PETRI NETS WITH GENERAL FIRING TIME DISTRIBUTION

The need for non-exponentially distributed transition firing times in SPNs has been observed by several authors. To this end the following Petri net extensions have been proposed:

- Deterministic and stochastic Petri nets (DSPNs) [24-27], where the firing delay of timed transitions may be deterministic, i.e., fixed.
- Extended Stochastic Petri Nets (ESPN) [28], Markov regenerative SPNs (MR-SPN) [25,29], extended DSPN [30], where the firing delay of timed transitions may have arbitrary distribution.

The numerical solution for these models are possible only under the *enabling constrain*: only one GEN or DET transition can be enabled in any marking [31,32]. The underlying stochastic process may be semi-Markov or Markov regenerative process.

In the case the underlying process is a GSMP, PNs can be solved via simulation. PNs are naturally suited for Discrete Event Simulation, since they describe the behaviours of real systems in terms of events that correspond to transition firings [33-36].

3.3.2 STOCHASTIC ACTIVITY NETWORKS

Composition is a highly desirable feature when modelling complex systems since it enables human users to focus on manageable parts from which a whole system can be constructed.

In the basic PN formalism a model consists of a single net which covers the whole system to be studied. Therefore PN models of complex systems tend to become very large and confused and suffer from the state space explosion problem. Stochastic Activity Networks (SANs) are a modeling formalism which extends Petri Nets [17,36]. SANs constitute an approach to the structuring of PNs through the sharing of places between different subnets. In the presence of symmetric sub-models they tackle the state space explosion problem by directly generating a reduced reachability graph in which all mutually symmetric markings are collapsed into one.

The basic elements of SAN are places, activities, input gates and output gates.

Places in SAN have the same role and meaning of places of Petri Nets. They contain an arbitrary number of tokens.

Activities are equivalent to transitions in Petri Nets. They can take a certain amount of time to be completed (timed activities) or no time (instantaneous activities). Timed activities support several kind of distribution type that define the time to firing of the activity. Parameters of activities, i.e., the parameters of the supported distributions, may be marking dependent.

For each activity is defined an execution policy that allow to reactivate, i.e., resample the completion time, an activity depending on the marking of the net. This is done by the mean of two predicates: the activation predicate and the reactivation predicate. An activity is reactivated when both the predicates are true. In any marking, the activation predicate is true if it was true at the moment the activity was last enabled. Thus, the activation predicate keep trace of the past marking of the net. On the other hand the reactivation predicate is evaluated at every evolution step, i.e., every time an activity fires.

When an activity fires a case is chosen. Cases represent the possibility of taking a specified action. Each case is assigned a real number that can be marking dependent and its evaluated at the moment the activity fires. The case that will perform is chosen probabilistically on the basis of the assigned real number.

Activities may complete at the same time if instantaneous or deterministic, i.e., timed with fixed delay. In this case a non deterministic choice decides which activity will complete first.

Each activity may have one or more input arcs, coming from its input places (which precedes the activity) and one or more output arcs going to its output places (which follow the activity). In absence of input gate and output gate, the presence of at least one token in each input place makes it able to fire and after firing one token is placed in each output place.

Input gates and output gates, typical constructs of SAN, can modify such a rule, making the SAN formalism more rich to represent actual situations. Particularly, they consist in predicates and functions, written in C++ language, which contain the rules of firing of the activities and how to distribute the tokens after the activities have fired.

Output gates are connected to cases of activities. They can be used to change the marking of the net when the input activity fires and the input case is chosen.

Input gates have a twofold function: they serve to specify guard conditions on activities and to define marking changes, like output gates. To this, input gates are assigned an input predicate and an input function, respectively. The input predicate defines, on the basis of the marking of the net, if an activity may complete or not. It can if the predicate is true, otherwise it cannot.

As in Petri Nets, a marking depicts a state of the net, which is characterized by an assignment of tokens to all the places of the net. With respect to a given initial marking, the reachability set is defined as the set of all markings that are reachable through any possible firing sequences of activities, starting from the initial marking.

Other than the input and output *gates*, which allow to specifically control the net execution, SAN offers two more relevant high-level constructs for building hierarchical models: *REP* and *JOIN*. Such constructs allow to build composed models (*atomic models*) based on simpler sub-models, which can be developed independently and then replied and joined with others sub-models and then executed. Particularly, the construct *REP* allows to replicate sub-models and the construct *JOIN* allows to join sub-models.

The construct *REP* is useful for two reasons: first, from a modelling point of view it avoids the construction of the same model multiple times; and second from a resolution point of view the use of the *REP* construct allows to generate a reduced state-space model because aggregation of states are obtained from indications at the network level. For a complete explanation of how the state-space based model is constructed from a SAN the reader may refer to [36]. Here we just mention that for a state-space based model to be constructed the underlying stochastic process of a SAN model must be a Markov Chain, i.e., only instantaneous and exponential timed activities can be present in the model and the initial marking must be a stable marking (a tangible marking in PNs).

Finally, in SAN is possible to specify extended places. Extended places offer the possibility of using non integer marking and to make use of vectors to represent the

marking of places. By the mean of extended place, modelling capabilities of SAN can be extended to deal with continuous state spaces. Moreover, the use of vectors allows a more compact specification of models.

The SAN model specification and elaboration is supported by Möbius tool, developed by the University of Illinois. The tool allows to specify the graphical model, to define the performance measures through reward variables, to compute the measures by choosing a specific solver to generate the solution.

Measure of interest are specified at the network level by the definition of reward function. Reward functions can be of two kind: state reward and impulse reward. State reward are function of the marking of the net, while impulse reward are function of the completion of activities. Moreover, reward can be defined at a specific time, an interval of time or averaged over an interval of time. The performance measure is then evaluated as the expected value of the reward function.

Figure 3.6 shows several SAN models of a SPARE gate of a DFT with two inputs. Models differ in that increasing level of SAN objects are used. The reader may already appreciate how the same model can be constructed in different ways, often depending by the personal choice of the modeller.

Figure 3.6.a shows the SPARE DFT gate and the associated Markov chain. The active component of the SPARE is A, while the warm stand-by component is B. Figure 3.6.b shows a SPN model of the system. In this case two activities are used to represent the failure of B. The activity that represents the failure of B when in stand-by is enabled only if A is working, while the activity that represent the failure of B when active is enabled only when A has failed. These conditions are modelled by the two-way arrows associated with the places A_OK and A_failed, respectively.

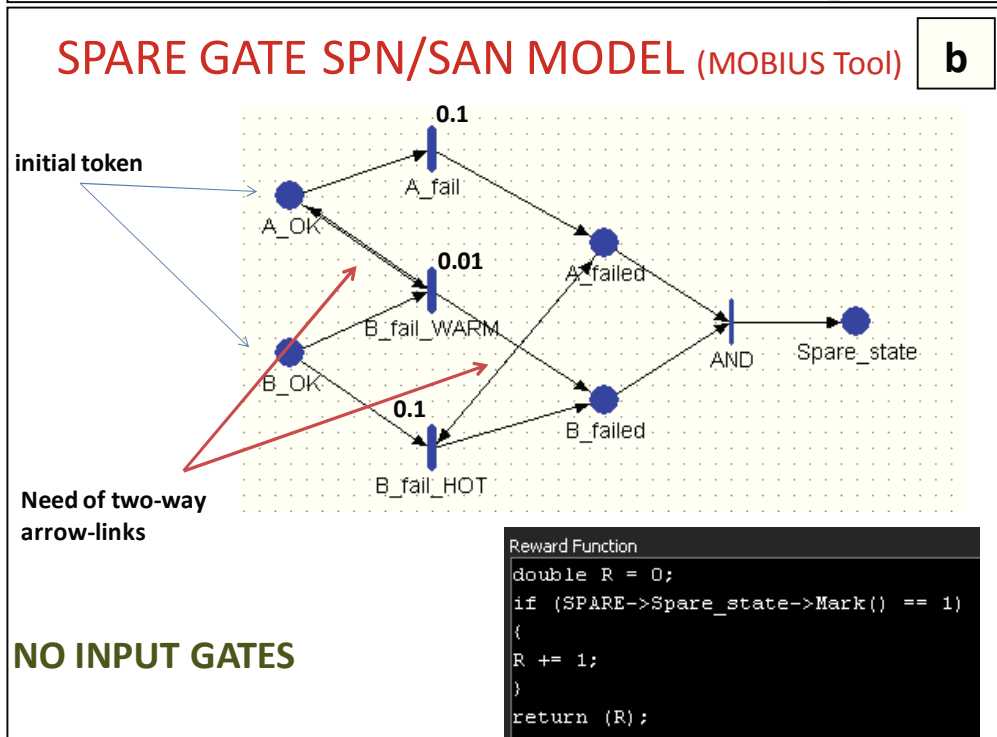
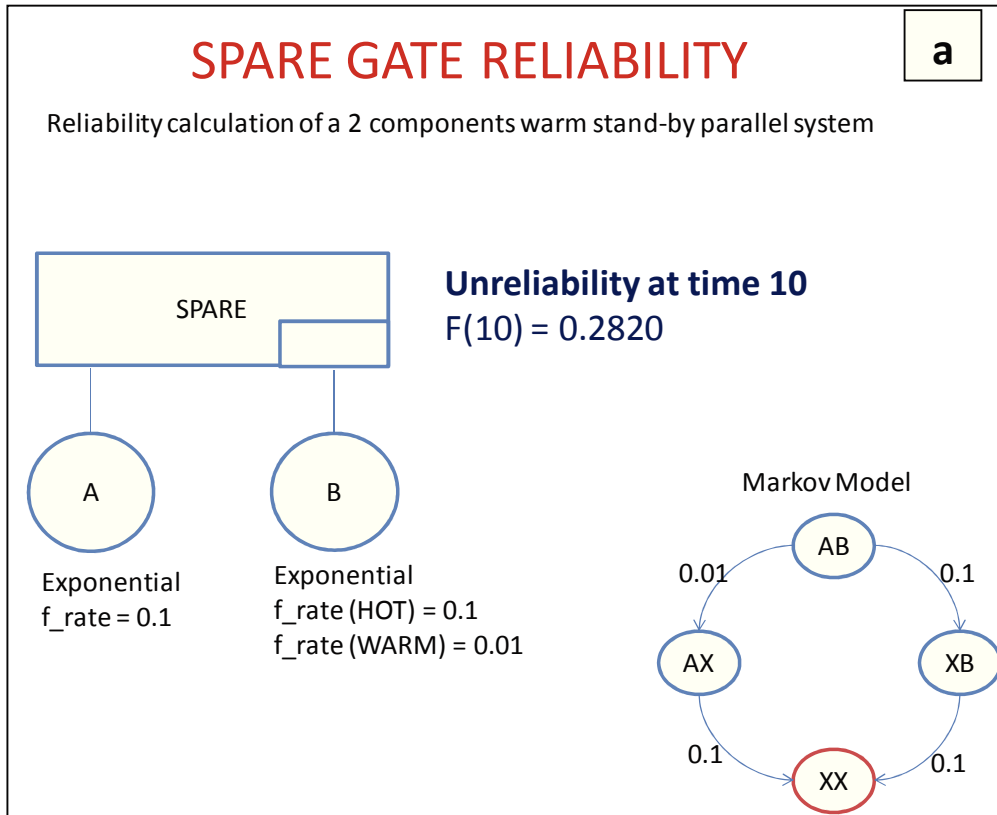
In Figure 3.6.c input gates have been introduced to replace the two-way links of Figure 3.6.b. The input predicate specified in the two input gates check the marking associated with A_OK and A_failed to enable the right activities relative to B. In Figure 3.6.d is shown that by the mean of the input functions is possible to avoid the use of output links from the two activities B_fail_WARM and B_fail_HOT. This could be done by the mean of output gates, too.

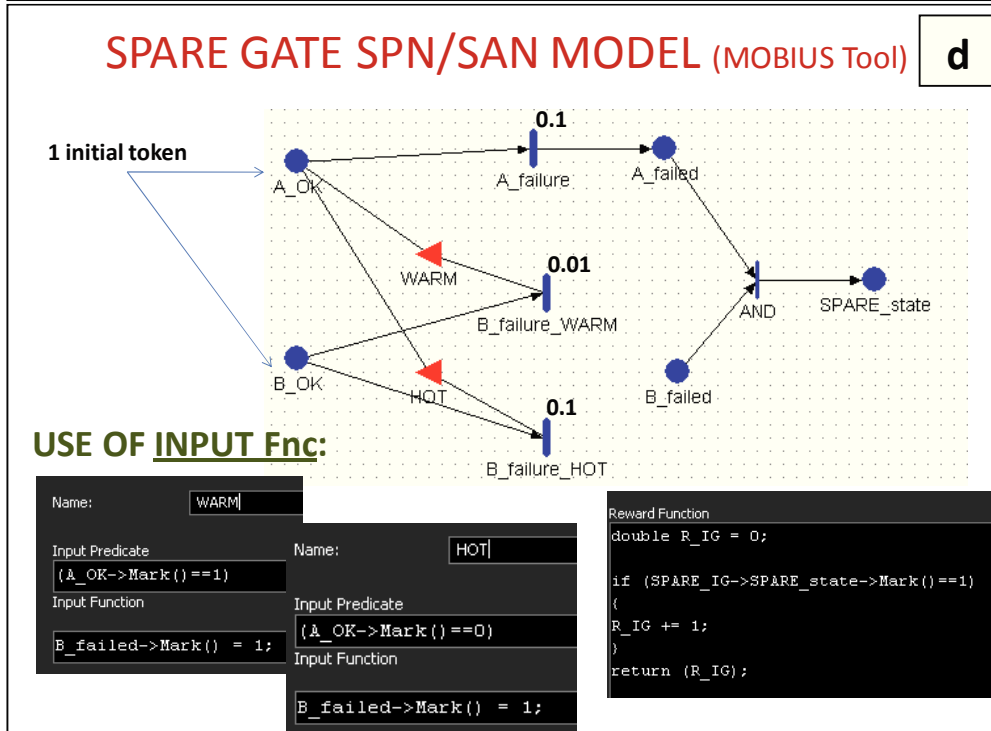
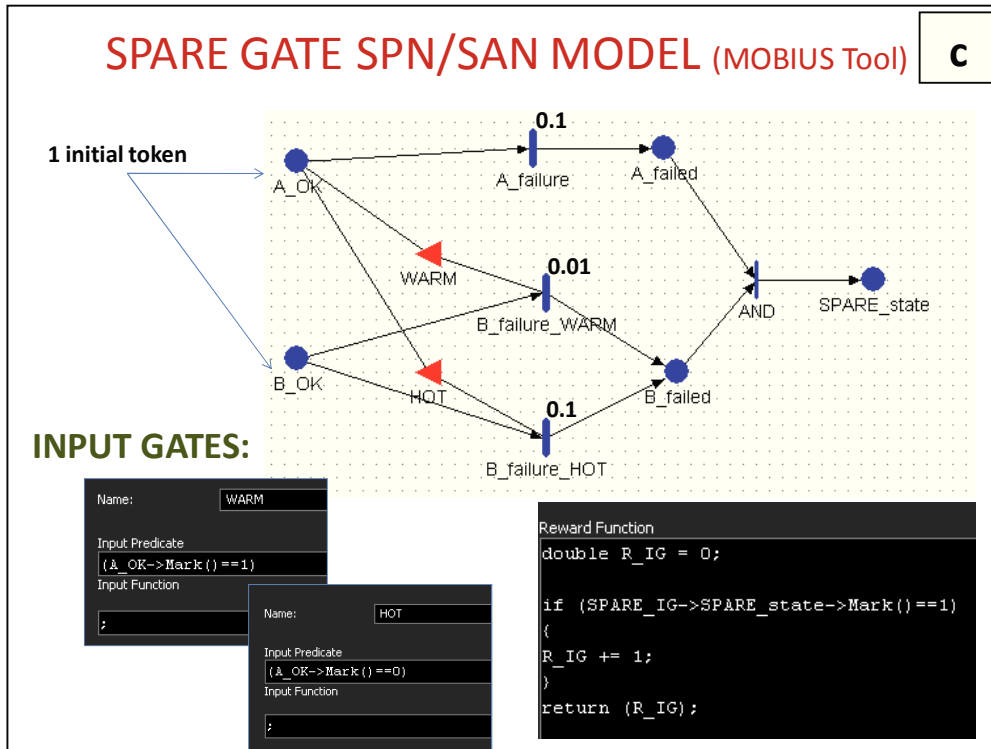
Figure 3.6.e shows the use of the reactivation predicate. The model results simplified, but we need to introduce a dummy place that is used to reactivate the activity B_fail. This is necessary when GEN distribution are used, while in the case of exponential distribution this additional construct can be avoided due to the memoryless property. In fact, we could define the reactivation predicate of B_fail only in terms of A_failed. In this case, if there would be more components than just A and B, B_fail would have been reactivated at any activity firing, but since the presence of the memoryless property the model would be correct.

Finally, all the above presented models are flat; a single net is used to model the SPARE system. In the case of large models this can lead to error and confusions. In Figure 3.6.f the use of the construct *JOIN* is shown. In this case three atomic models were created: A, B and AND and joined together, i.e., variables of atomic models are shared via the *JOIN* construct.

The examples above show some problem when modelling with SAN. In particular three issues are of practical importance:

- the flexibility offered by SAN allows to model a very general class of systems. However, as seen, also for very small systems the construction possibilities are so many that the resulting model is often a choice of the modeller. This could result in debugging difficulties and poor maintainability of the model over time;
- the use of the reactivation predicate could be problematic when large models are developed, due to the need of specifying apposite constructs that limit the modeller in abstracting from the complexity of the overall model.
- when *JOIN* constructs are used it is not possible to develop reward functions that take as inputs information from different sub-models. Thus, in this case, one must generate an atomic model that contains all the variables of interest used in the specification of the reward.





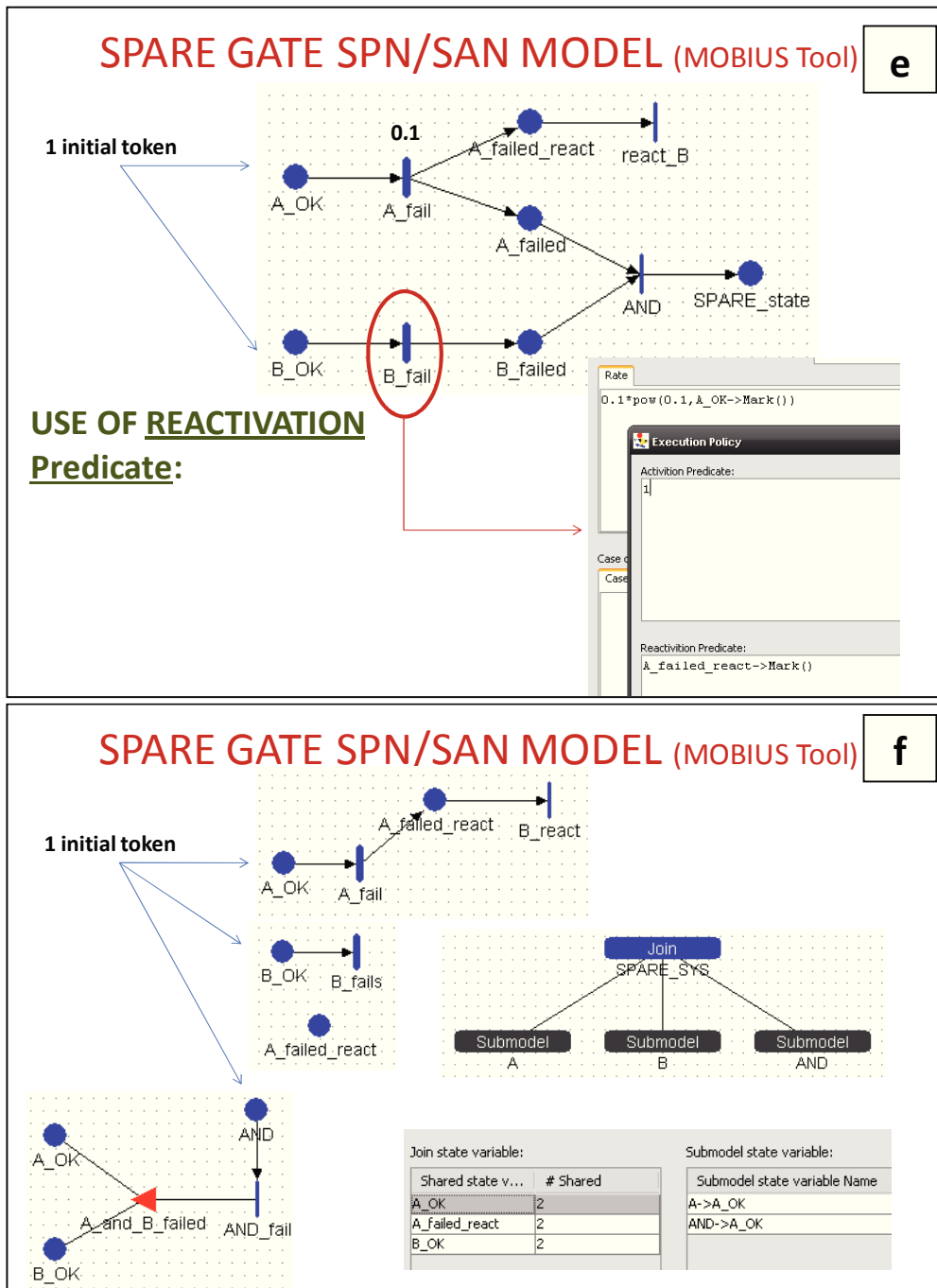


Fig. 3.6. Examples of SAN model of a SPARE gate with two inputs.

All these problems will be addressed in Chapter 6 where we propose an algorithm for the automatic generation of SAN models from Adaptive Transition Systems (ATS); introduced in Chapter 5. ATS gives the possibility of generating standardized SAN models that are compositional and where the issue of reactivating a transition is

explicitly managed. Moreover, the ATS-to-SAN conversion algorithm resolves the third problem by the use of extended places.

3.4 CONCLUSION

In this chapter we gave a brief description of formalism used in Reliability Engineering that we have considered as hybrid because of the high level language of description, typical of non state-space models, that they offer and because of the solution techniques that are common to state-space models.

In particular we have described DFT as an extension of FTs. We have seen that the main limitation of DFT lay in the formalism itself: DFT are not general enough. Spare components cannot have different failure rates depending on the subsystem where they are employed and load-sharing cannot be modelled. Since repairable components cannot be considered, measures like availability or reliability with repairs cannot be evaluated. If one try to extend DFT to consider repairable components one is subjected to the choice of a specific substitution logic. Maintenance management and non determinism are not consider as well. Finally, the failure logic of PAND gates is not able to consider complex ordering of events in the case of repairable components.

On the other hand, stochastic extension of Petri nets allow to model very general systems and are very well suited for discrete event simulation when general distributions are considered.

SAN extends SPN allowing one to compose atomic models of specific part of the modelled system. Thus, with SAN one of the main problem of SPN, i.e., SPN are flat, are overcome. The main problems of SAN, i.e., (i) the generality of the model that leads to a non standardized approach to the modelling activity; (ii) the difficulty of defining reactivation predicates; and (iii) the problem of defining reward functions when the *JOIN* construct is used, were illustrated in the previous section. As said all these issues, together with the generalization of DFT models to include repairable components, will be managed with the introduction of ATS.

BIBLIOGRAPHY

- [1] Ren, Y., & Dugan, J.B., 1998. Design of reliable systems using static and dynamic fault trees. *IEEE Transactions on Reliability*; 47:234-244.
- [2] Dugan, J.B, & Bavuso, S.J., 1990. *Fault trees and sequence dependencies*. Proc. Ann. Reliability and Maintainability Symposium; 232-235.
- [3] Dugan, J.B, & Bavuso, S.J., 1992. *Dynamic fault-tree models for fault tolerant computer systems*. *IEEE Transactions on reliability*; 41(3):363-377.
- [4] Amari, S., Dill, G., & Howald, E., 2003. *A new approach to solve dynamic fault trees*. Annual Reliability and maintainability symposium; 374–379.
- [5] Dugan, J.B., Venkataraman, B., & Gulati, R., 1997. *Diftree: a software package for the analysis of dynamic fault tree models*. In Proceedings Annual Reliability and Maintainability Symposium; 64-70.
- [6] Montani, S., Portinale, L., Bobbio, A., & Codetta-Raiteri, D., 2008. *RADYBAN: A tool for reliability analysis of dynamic fault trees through conversion into dynamic bayesian networks*. *Reliability Engineering and System Safety*; 93:922-932.
- [7] Boudali, H., Crouzen, P., & Stoelinga, M., 2007. *Dynamic fault tree analysis using input/output interactive Markov chains*. In Proceedings 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks DSN '07; 708-717.
- [8] Sullivan, K.J., Dugan, J.B., & Coppit, D., 1999. *The Galileo fault tree analysis tool*. In Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing; 232-235.
- [9] Durga Rao, K., Gopika, V., Sanyasi Rao, V.V.S., Kushwaha, H.S., Verma, A.K., & Srividya, A., 2009. *Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment*. *Reliability Engineering and System Safety*; 94:872-883.

- [10] Boudali, H., & Dugan, J.B., 2005. *A New Bayesian Network Approach to Solve Dynamic Fault Trees*. In Proceedings Annual Reliability and Maintainability Symposium; 451-456.
- [11] Boudali, H., & Dugan, J.B., 2006. *A Continuous-Time Bayesian Network Reliability Modeling and Analysis Framework*. IEEE Transactions on Reliability; 55:86-97.
- [12] Codetta-Raiteri, D., 2005. *The Conversion of Dynamic Fault Trees to Stochastic Petri Nets, as a case of Graph Transformation*. Electronic Notes in Electronic Computer Science; 127:45-60.
- [13] Distefano, S., & Puliafito, A., 2007. *Dynamic reliability block diagrams vs dynamic fault trees*. In Proceedings Annual Reliability and Maintainability Symposium RAMS '07; 71-76.
- [14] Bouissou, M. & Bon, J.L., 2003. *A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes*. Reliability Engineering and System Safety; 82:149-163.
- [15] Bause, F. & Kritzinger, P.S, 2002. *Stochastic Petri Nets*. An introduction to the theory. Vieweg.
- [16] Marsan, M.A., Balbo, G., Conte, G., Donatelli, S. & Franceschinis, G., 1995. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley.
- [17] Sanders, W. H., & Meyer, J. F., 2002. *Stochastic activity networks: Formal definitions and concepts*. Lectures on Formal Methods and Performance Analysis. Springer Verlag.
- [18] Relex. URL: <http://www.ptc.com/products/windchill/quality-solutions/>
- [19] Boudali, H., Nijmeijer, A.P. & Stoelinga, M.I.A., 2009. *DFTsim : a simulation tool for extended Dynamic Fault Trees*. Proceeding Spring Sim '09.
- [20] Gulati, R., & Dugan, J.B., 1997. *A modular approach for analyzing static and dynamic fault trees*. Proc. Ann. Reliability and Maintainability Symposium; 57-63.

- [21] Merle, G., Roussel, J., Lesage, J. & Bobbio, A., 2010. *Probabilistic Algebraic Analysis of Fault Trees With Priority Dynamic Gates and Repeated Events*. IEEE Transactions on Reliability; 250-62.
- [22] Baier, C. & Katoen J.P., 2008, *Principles of Model Checking*. The MIT Press.
- [23] Peterson, J.L., 1981. *Petri Net theory and the modelling of systems*. Englewood Cliffs, NJ: Prentice-Hall.
- [24] Marsan, M.A. & Chiola, G., 1987. *On Petri Nets with Deterministic and Exponentially Distributed Firing Times*. LNCS; 266:132-145. Springer Verlag.
- [25] Choi, H., Kulkarni, G. & Trivedi, K.S., 1993. *Transient analysis of deterministic and stochastic petri nets*. In Proc. 14-th Intern. Conference on Application and Theory of Petri Nets, Chicago, Illinois. Springer Verlag.
- [26] Ciardo, G. & Lindemann, C., 1993. *Analysis of deterministic and stochastic Petri nets*. In Proc. 5-th Intern. Workshop on Petri Nets and Performance Models; 160-169.
- [27] Lindemann, C., 1993. *An improved numerical algorithm for calculating steady-state solutions of deterministic and stochastic Petri net models*. Performance Evaluation; 18:75-95.
- [28] Dugan, J.B, Trivedi, K.S., Geist, R.M & Nicola, V.F, 1984. *Extended Stochastic Petri Nets: applications and analysis*. In Performance '84, E. Gelenbe, Ed., Paris, France, North-Holland.
- [29] Choi, H., Kulkarni, V.G. & Trivedi, K.S., 1993. *Markov regenerative stochastic Petri nets*. In Performance, North-Holland.
- [30] German, R. & Lindemann, C. 1993. *Analysis of Stochastic Petri Nets by the Method of Supplementary Variables*. In Performance '93, North-Holland.
- [31] Ciardo, G., Muppala, J. & Trivedi, K.S., 1991. *On the solution of GSPN reward models*. Performance Evaluation, 12(4):237-253.

- [32] Ciardo, G., German, R. & Lindemann, C., 1993. *A characterization of the stochastic process underlying a stochastic Petri net*. In Proc. 5-th Intern. Workshop on Petri Nets and Performance Models; 170-179.
- [33] Chiola, G. & Ferscha, A., 1993. *Distributed discrete event simulation of timed transition Petri Nets*. IEEE Parallel and Distributed Technology, 1 Special Issue on "Parallel and Distributed Systems: from theory to practice".
- [34] Chiola, G. & Ferscha, A., 1993. *Distributed simulation of timed Petri nets: exploiting the net structure to obtain efficiency*. In Proc. 14-th Intern. Conference on Application and Theory of Petri Nets, Chicago, Illinois. Springer Verlag.
- [35] Chiola, G. & Ferscha, A., 1993. *Exploiting Petri net model structure to improve distributed Simulation*. In 26-th Hawaii Intern. Computer Science Symposium, Honolulu, Hawaii.
- [36] Chiola, G. & Ferscha, A., 1995. *Performance comparable design of efficient synchronization protocols for distributed simulation*. In Proc. of MASCOT'95; 343-348.
- [37] Sanders, W.H. & Meyer J.F., 1999. *Reduced Base Model Construction Methods for Stochastic Activity Networks*, IEEE Journal on Selected Areas in Communication; 9(1):25-36.

CHAPTER 4

4.1 INTRODUCTION

In Chapter 3 we introduced DFTs [1-6,12,13] and addressed the main limitations. In this chapter we present a tool for the resolution of DFT via simulation with the objectives of overcoming the constraints typical of DFT. In particular with this tool we address the possibility of using general distribution function in the specification of the time to failure of the BEs of the tree.

MatCarloRe is a tool that was first developed in Simulink® in order to benefit from the high level interface that the tool provides. Successively, a Java graphical interface was developed and integrated with the Matlab® environment (a new tool that we call JDFTDes). The creation of the Java interface has two advantages with respect to the Simulink model: (i) allows a more friendly construction of the model, and (ii) allows to bypass the limitations imposed by the Simulink® formalism that does not allow to consider shared spare components.

The remainder of this chapter is organized as follows: in Section 2 we give an introduction of the Monte Carlo theory for Discrete Event Simulation. In Section 3 we present the Simulink® tool MatCarloRe. In Section 4 we report the application of the tool to a real case study. The JDFTDes will be shortly introduced in Section 5. Two accepted papers describing MatCarloRe and MatCarloRe-JDFTDes are reported

in Appendix A, where two accepted paper are attached. Finally in Section 6 are reported some conclusion.

4.2 MONTE CARLO SIMULATION OF DFT

Simulation programs, especially if well structured, are in general very comprehensible and known for the ease with which modifications and additions can be made [15-19,31]. Monte Carlo simulation is a valuable method which is widely used in solving real problems in many engineering fields. It has been used by [32, 33] for the study of system reliability, based on the well-developed neutron transport ideas. The simulation technique allows the estimation of reliability indices by simulating the actual process and random behaviour of the system through a computer model.

Monte Carlo simulation is implemented by running the model a large number of times, each representing an ensemble of random walks within the discrete state-space of system configurations, in order to generate a large number instances from which all the reliability indices required about the system are retrieved.

As the system is composed of many components (or elements) grouped together to perform a certain function, the system modelling begins with the identification of the components of which the system is composed. Let us denote by s_i the possible states in which the i -th component of the system may be. In the simplest case s_i may have two possible states: one is the "UP" or working ,the other is the "DOWN" or failed. The state of the system can be described by a vector S :

$$S = (s_1, s_2, \dots, s_i, \dots, s_n), \quad (4.1)$$

where n represents the number of components of the system. When a component changes its state, a new point in the state-space is reached. Some of these points are of failure for the whole system.

While in static-FT certain combinations of component failures correspond to system failure, in DFTs the same state can be either a failure state or a good one, depending on the previous state sequence. Hence, in DFT, given the notion of state in

eq. 4.1, it is not possible to define a system state a good one or a failed one without looking at the system evolution.

Vector S changes with respect to time, so each simulation consists of a collection of state vectors representing the movement of the random walk within the phase space. Such history can be written as:

$$H_k = (S_0, t_0; S_1, t_1; \dots; S_m, t_m), \quad (4.2)$$

where k represents the simulation-run index, t_i the time of the transition from state S_{i-1} to state S_i and m represents the index of the ending simulation time, i.e., the last state change within a simulation batch. The point (S_i, t_i) represents a state-space point, while the collection of all such points is called the state-space of the system. This space is continuous in time and discrete in the states. Transitions among states depend on the components transition from the UP state to the DOWN one, according to a stochastic law (the probability density function) that characterizes the component behavior.

The only information needed for the analysis are: a) the probability density function (pdf) of the time to failure of each component and their parameters values; b) the mission time of the system; and c) the system failure mode configuration.

The most common way to conduct the simulation of a FT is to consider the system as a whole (indirect Monte Carlo [36]); previous literature [34, 35] consider a system failure rate as the sum of all the component failure rate and calculate the transition time to the new state. After that, the component which performs the transition is chosen in a stochastic manner. When the component is chosen its failure rate is set equal to zero and the process is repeated till the occurrence of the TE or the end of the mission. The system operability (i.e. the occurrence of the TE) is evaluated each time a transition occurs. Generally it is convenient to make use of the FT methodology to check the system operability each time it changes its state.

The problem with this approach is that we must check the operability state of the system each time the system makes a transition; moreover it is needed to recalculate the system failure rate at each state change by imposing the failed component failure rate equal to zero.

When considering DFTs we need to take into account the system evolution history in order to assess the nature of the reached state. Moreover, in the case of components with different time distribution the traditional indirect Monte Carlo method, becomes impractical due to the lack of the memoryless property of general distributions.

A more straightforward method to account for time-dependent failure and repair behaviours of components is the direct Monte Carlo method [37]. It samples the time to failure of each BE instead of the overall system transition time. Following the idea of direct Monte Carlo, our simulation approach makes use simultaneously of the FT and the Monte Carlo methodologies. Instead of simulating the system walk in the phase space we consider BEs as basic entities. The *failure time of each BE is calculated at the beginning of each batch* and these information are passed to the gates which they are connected to. The gate state is determined and, if failed, its failure time is passed to the higher levels. In this way for each gate of the tree two information are available: the state of the gate before the mission time and, if failed, the time when this condition became true.

Information about the time of events are important when considering dynamic gates. Moreover many simulation data can be stored in a very straightforward way: for each gate we can obtain the failure number of occurrences, the mean time to failure and information about which connected subsystem forced mostly the failure of the gate.

The approach followed is very suitable for dynamic gates since the failing order is tracked. Therefore, in this way it is not necessary to store the previous system states and check the order of occurrence to assess whether a state is of failure or good.

The complexity of the algorithm is carried by each single gate, whose logic is programmed in order to infer its own state. For the same reason, with this approach there is no need to update the overall system failure rate at each transition.

Generated all the transition times of each component, the failure time for each gate can be calculated based on the logic of the gate itself with respect to its inputs.

At the highest level information about the system state are available and used to estimate the system reliability.

Due to the nature of the approach, it would be straightforward to program or modify the logic of the gates in a modular way, without compromising the environment already set. In this way, the end user can just make full use of the gates created by the programmers and exploit with some small knowledge of Simulink® the power of the library.

4.3 SIMULINK® LIBRARY

The simulation tool implemented makes use of a high level modeling interface that allows the user to assemble the DFT by picking basic events and gates from a library and dropping these elements on a Simulink® sheet. BEs and gates are then linked together to create the system configuration.

The tool consists of a Simulink® library called MatCarloRe (Figure 4.1), formed of blocks representing the various elements of DFT, such as, the dynamic gates PAND, SPARE, SEQ and FDEP as well as the static gates AND, OR and Voting and the BEs.

Each block representing a gate can receive n input and distribute m output by simply using the *mux* and *demux* blocks available in the main Simulink® library. Inputs and outputs are of two kind: we define y as the binary vector which indicates whether the input and output have occurred (value 1) or not (value 0), and t as the vector containing the failure times. Only for blocks representing the SPARE and the FDEP gates is not provided the input vector y .

Once the model is built and the input parameters are defined it is possible to run the simulation defining the total number of batches. At each batch the model returns a binary value that indicates whether the system has reached the state of failure or not. In particular, the model returns the value 1 if the fault is reached, 0 vice versa.

To avoid large amounts of data storage the Simulink® block *memory* is used to set the progressive sum of results of each iteration. In this way only the total value of

batches that revealed a fault is considered and stored in the Matlab® workspace by the block TE. The estimated unreliability of the system can be finally obtained dividing the value stored in TE by the total number of batches.

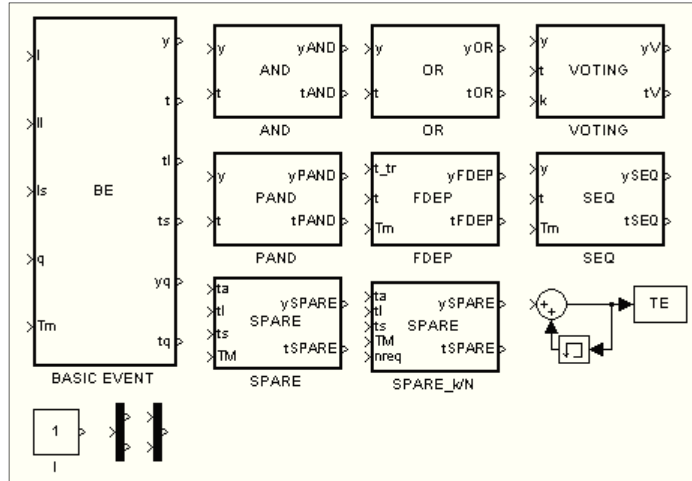


Fig. 4.1. MatCarloRe library elements.

In the next section it is shown how to build a simulation model for a DFT introducing in more details the BE block and the dynamic gates. In order to proof the validity of the tool the results performed with the MatCarloRe are compared with analytical results.

4.3.1 THE BE BLOCK

The BE block is designed to generate the times of failure of basic events. At the time the tool was built it could generate only the times of failure for components with exponential distribution as well as fixed probability. The version developed by the support of the JAVA interface integrated with the Matlab® environment has been integrated to support also Weibull distributions. However, other type of distribution can be easily integrated.

For components subjected to random failures the unreliability at time t can be expressed by the following relation:

$$F(t) = 1 - e^{-\lambda t}, \quad (4.3)$$

where λ is the failure rate of the component. The sampled time to failure is then calculated by the inverse relationship:

$$t^* = -\frac{\log(1-F^*)}{\lambda}, \quad (4.4)$$

where F^* is a random number generated in $[0, 1]$. If t^* is smaller than the mission time t_m , the component is considered as failed.

For events supplied with a fixed probability q the following procedure is defined: a random number q^* generated uniformly in $[0, 1]$ is extracted and compare with the value q . This comparison returns a failure time according to the following relation:

$$t^* = \begin{cases} +Inf & \text{if } q^* \geq q \\ t_q^* & \text{if } q^* < q \end{cases} \quad (4.5)$$

where t_q^* is a random number generated uniformly in $[0, t_m]$.

4.3.2 THE PAND BLOCK

The PAND block models the logic that underlies the PAND gate of a DFT [10,14,23]. The block logic is illustrated in the flowchart in Figure 4.2. First, according to the vector y , it is verified if all the input events occurred. If this condition is not satisfied the gate does not trigger. Otherwise the following conditions are checked: $t_i < t_j$ for $\forall i < j$ with $i, j = 1, 2, \dots, n$, where n is the number of inputs of the gate. If such control over failure times is satisfied the gate triggers with a time to failure equal to the maximum failure time of its inputs.

To test the validity of the block to perform the requested calculation we use a small example. A PAND gate with two BEs is considered. The failure rates of BEs are $\lambda = 0.1$ for the first component from the left hand side of the gate and $\lambda = 0.01$ for the second one. The mission time is $t_m = 10$.

In Figure 4.3 is shown the model built for the simulation with the MatCarloRe tool. It is possible to see the BE block which takes as inputs the failure rates of the two components and the mission time of the system. The output of the block is the time to failure of the two components (i.e. vector t) and the binary vector y which

indicates whether the time to failure of the components is smaller than the mission time. These vectors are given as input to the PAND block. The output scalar y_{PAND} of the PAND block is passed to a construct that perform the progressive sum of results over the number of batches and at the end of all the iterations the result is stored by the block TE into the Matlab® Workspace.

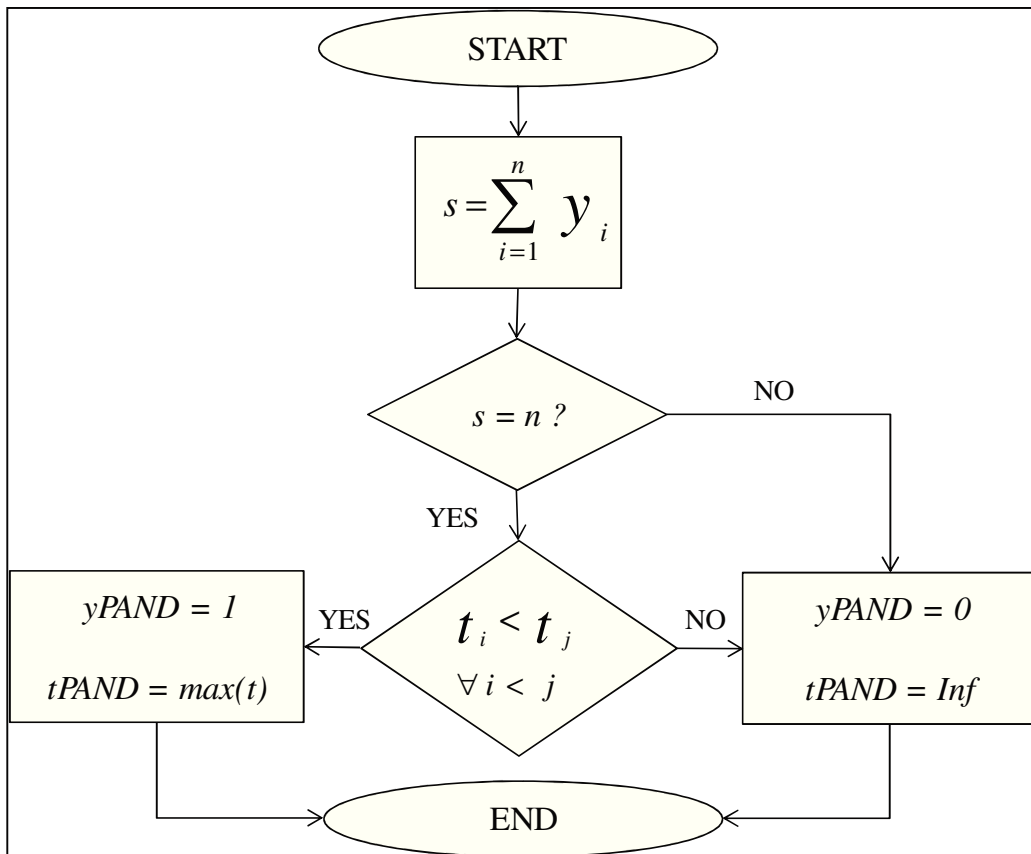


Fig. 4.2. Flow Chart of the PAND Block.

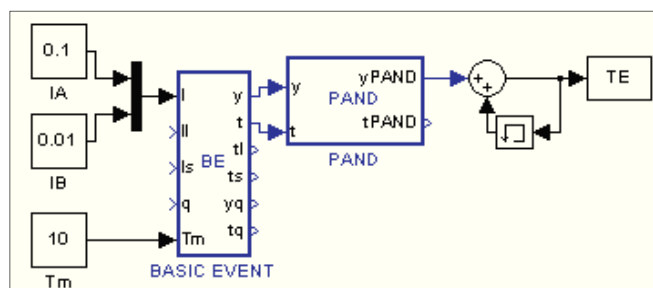


Fig. 4.3. Simulation model of a PAND gate with two basic event with failure rate $\lambda = 0.1, 0.01$ and mission time $t_m = 10$.

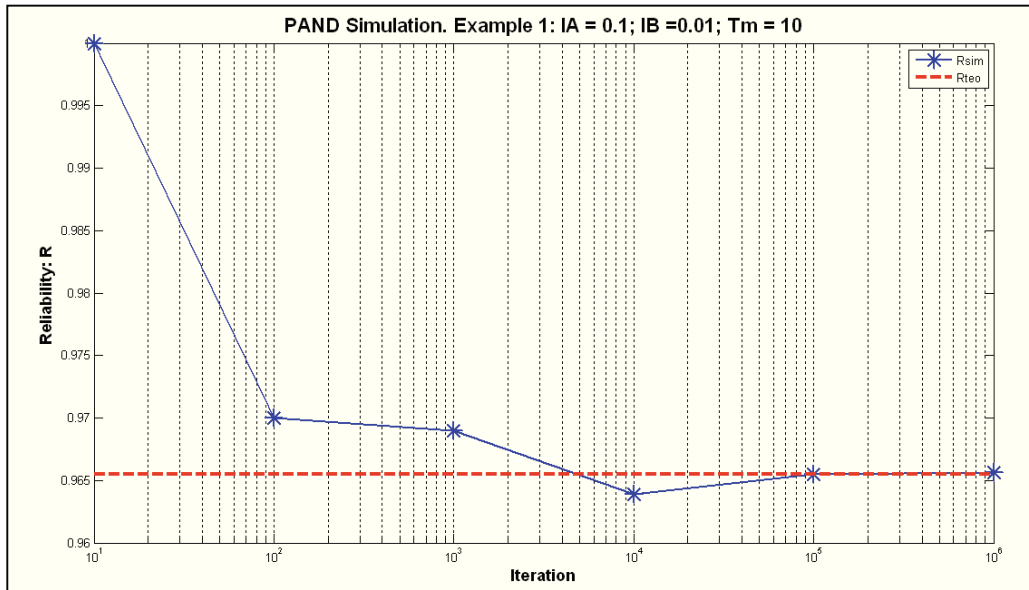


Fig. 4.4. Simulated Vs analytical reliability of a PAND gate with two basic event with failure rate $\lambda = 0.1, 0.01$ and mission time $t_m = 10$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulation results.

The simulated reliability versus the analytical one calculated by the mean of the associated CTMC are shown in Figure 4.4. The total number of batches are set to 10^i with $i = 1, 2, \dots, 6$. It is evident that for a large number of iterations (of five magnitude order) the error is very low and acceptable.

4.3.3 THE SPARE BLOCK

The SPARE block models the logic that underlies the SPARE gate of a DFT [10,14]. The block logic is illustrated in the flowchart in Figure 4.5.

Before to describe the structure of the algorithm of the SPARE block let consider the operations performed by the BE block. In presence of spare components the BE block sample two time to failure for the considered elements; one relative to the stand-by state and the other relative to the active state. The sampled time to failure relative to the active state is considered active, while the sampled time to failure relative to the stand-by state is considered passive and will be used by the SPARE block only if the component is requested by the gate.

Our implementation of the SPARE gate extends the classical one of DFT where only one primary component can be considered. In our implementation the SPARE block allows to consider more than one primary component for the gate.

The steps performed by the algorithm are:

- first a permutation of the vector t , representing the sampled time to failure of primary components, is made, sorting the vector in the ascending order;
- then, it is examined, among the primary components whether there are components that have failed before the end of the mission time; if no failure is verified the gate will not trigger;
- on the other hand, the algorithm checks if there are spare components able to replace the primary component that failed first.

Let us consider the logic of replacement of a generic active component which fails. Its replacement can take place only if:

- the spare part is still available (namely, it has not been used to replace another failed component) and;
- the time to failure of the spare (during its stand-by condition) is greater than the time to failure of the primary component that must be replaced.

If these two conditions are not satisfied by any spare component the gate triggers with a time of occurrence equal to the last failed primary component. Otherwise the block updates the time to failure of the primary component by adding the sampled time to failure (when active) of the spare component chosen for the replacement. The substituting spare is finally declared as busy. The search for active components applicants for replacement is repeated till there are primary components with failure time smaller than the mission time.

It is worth to highlight that the order in which the spares are chosen to replace the failed component follows the graphical order of positioning defined by the gate (e.g. in case of two spares that can replace an active failed component, it is chosen to replace the component with the spare that graphically is placed to the left).

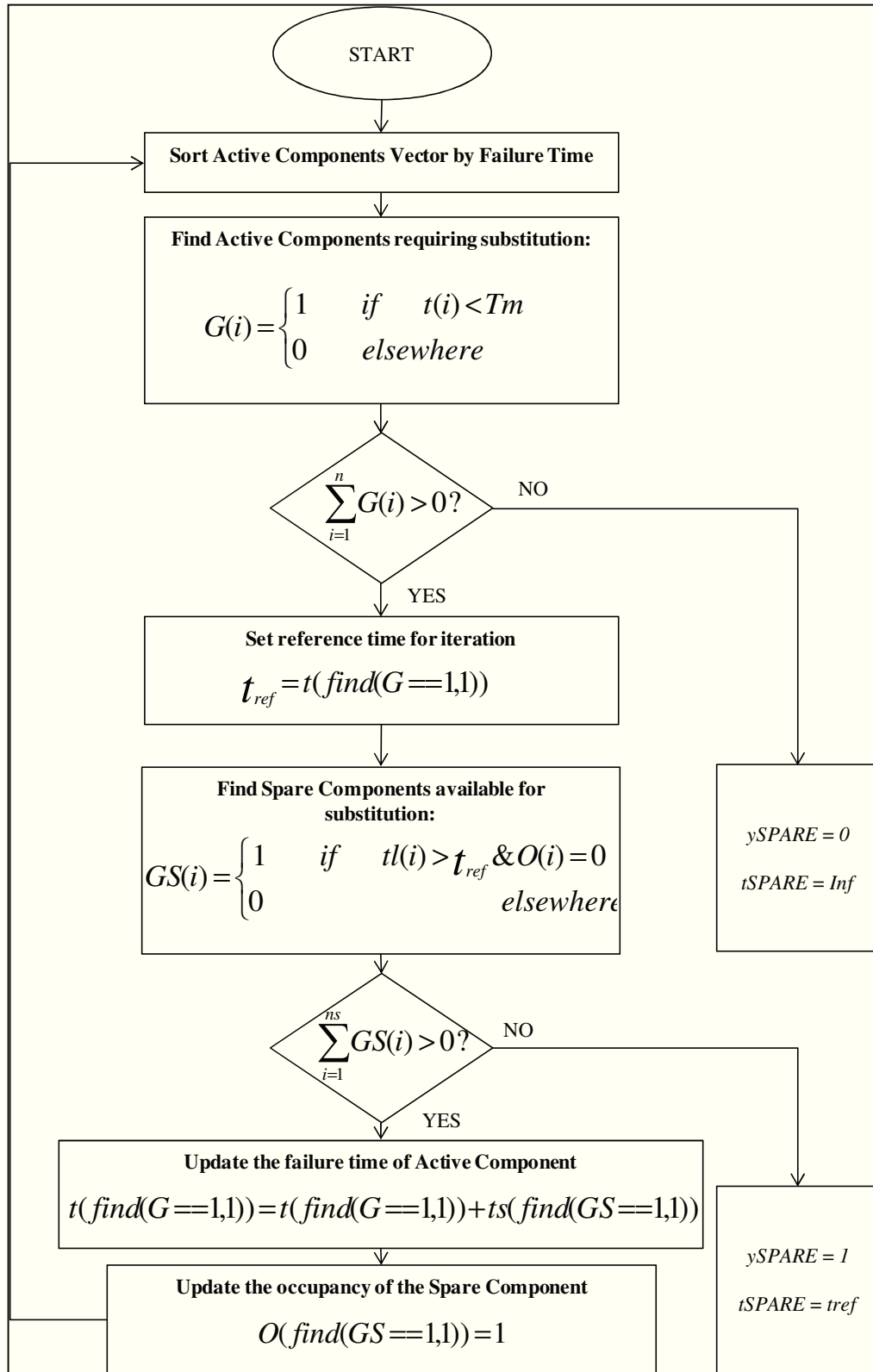


Fig. 4.5. Flow Chart of the SPARE Block.

The SPARE block offers many advantages in terms of modelling power. Many reliability tools (e.g., Relex®, Galileo® [7-9,11,38]) present some difficulties about the construction of DFT with SPARE gates: if a spare part is shared among more components, the DFT will have as many SPARE gates as the number of active components which share the spare. In this way, the first input of the generic SPARE gate is the active component, while the second input is the shared spare part, common to all the set of SPARE gates drawn. The final logic implemented by the set of SPARE gates is realized linking them together by an OR gate in the upper level (Figure 4.6).

Another situation is redundancy in the active components (i.e. not all the active components are requested for the system to work). In this case an AND gate is placed to the upper level (Figure 4.7). If the number of active components is even greater than the presented examples in the previous figures the modelling activity is even more complex involving the use of AND and OR gates in the tree structure. Therefore what the SPARE gate of the MatCarloRe library is able to do is to bypass all these architectural tricks, by the simple use of a single block called SPARE_k/N, where k is the number of components requested to work and N is the number of initial primary components.

The differences in the flow chart between the SPARE block and SPARE_k/N block are located in the first rhombus of the chart in Figure 4.5 where it is needed to consider the $N-k$ failures allowed for the system to work.

To test the validity of the two blocks SPARE and SPARE_k/N the results of two simple example are shown. Let consider a system composed of two primary and two spare components with failure rate $\lambda = 0.1$. The latency factor for the spare components is $a = 0.1$ and the mission time is $t_m = 10$. Figures 4.8 and 4.10 show the models built with the MatCarloRe tool using the SPARE block and SPARE_k/N block, respectively. The simulation models are equal in both cases except that the number of components needed for the system to work in the second example are defined by $nreq$.

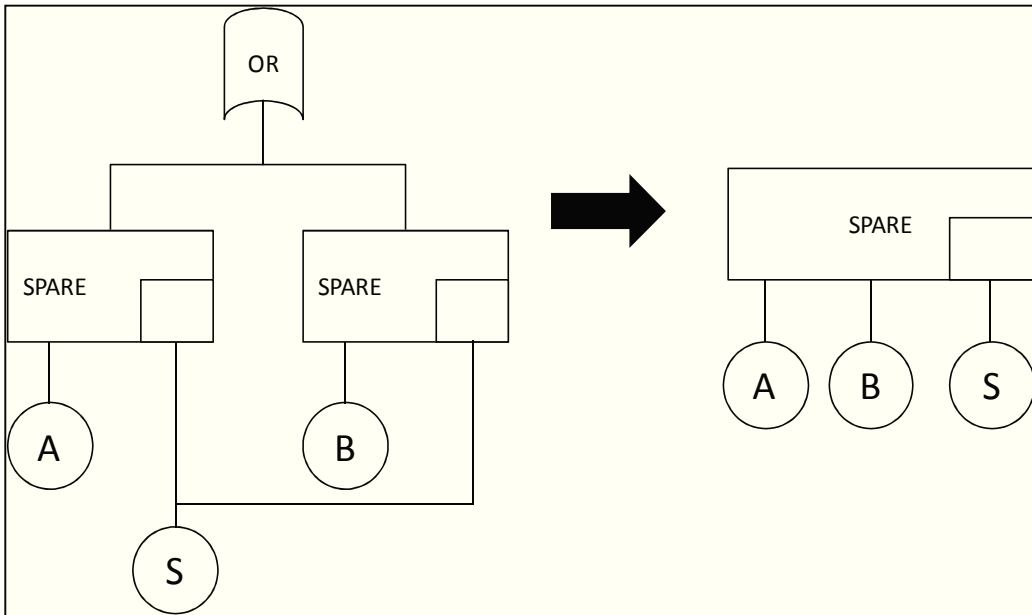


Fig. 4.6. Illustration of SPARE gates modelling vantages introduced by the tool; case of common shared spare. Left: Relex® model; right: MatCarloRe model.

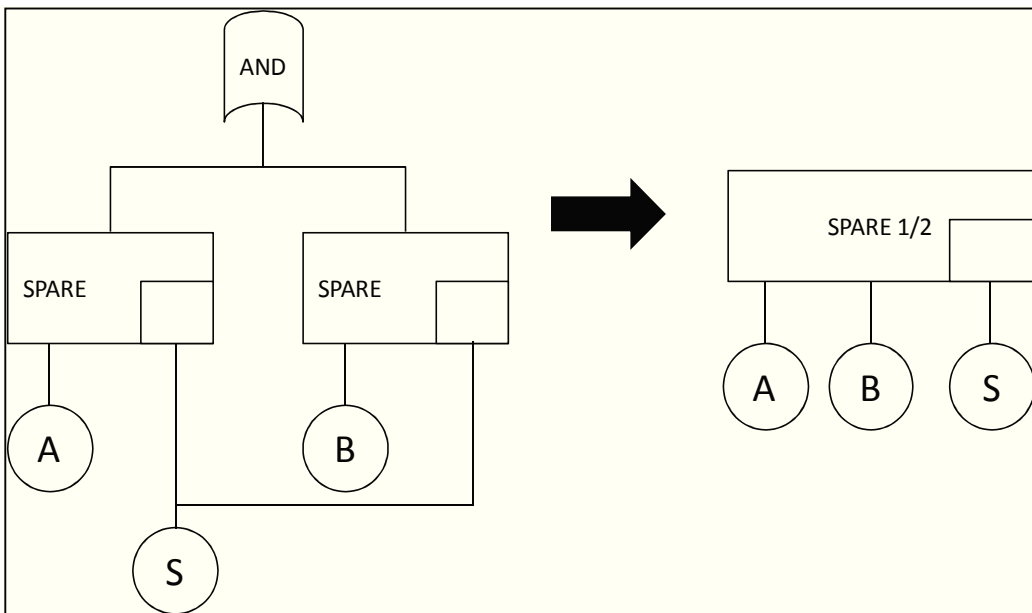


Fig. 4.7. Illustration of SPARE gates modelling vantages introduced by the tool; case of redundancy in active components. Left: Relex model; right: MatCarloRe model.

The BE block takes as inputs the failure rates of the two active components, the failure rate of the spare components when in stand-by state and when active and the mission time of the system. It returns as output the time to failure of the two active components stored in the vector t . The time to failure of spare parts when in the stand-by state and when active are given respectively in the vectors tl and ts . Vectors t , tl and ts and the mission time t_m are the input of the SPARE block. The output y_{SPARE} , taken over repeated iterations, is then used to compute the reliability of the system.

In Figure 4.9 is shown the simulated reliability versus the analytical calculated by the mean of the associated CTMC with respect to the number of batches. Again, for a number of iterations of the order of 10^5 the error of prediction is very low and acceptable.

The results of the simulation of the k/N model are shown in Figure 4.11. Again for iteration of order 10^5 the simulated reliability is very close to the CTMC results.

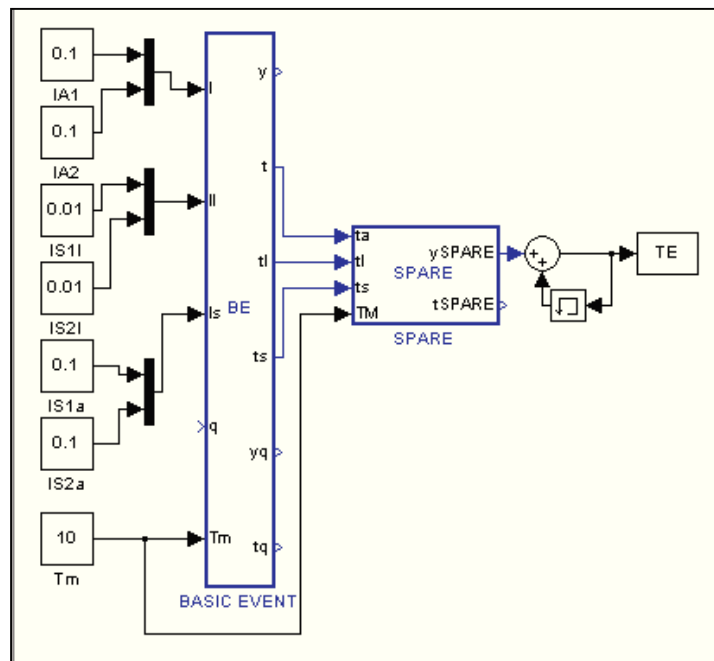


Fig. 4.8. Simulation model of a SPARE gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$.

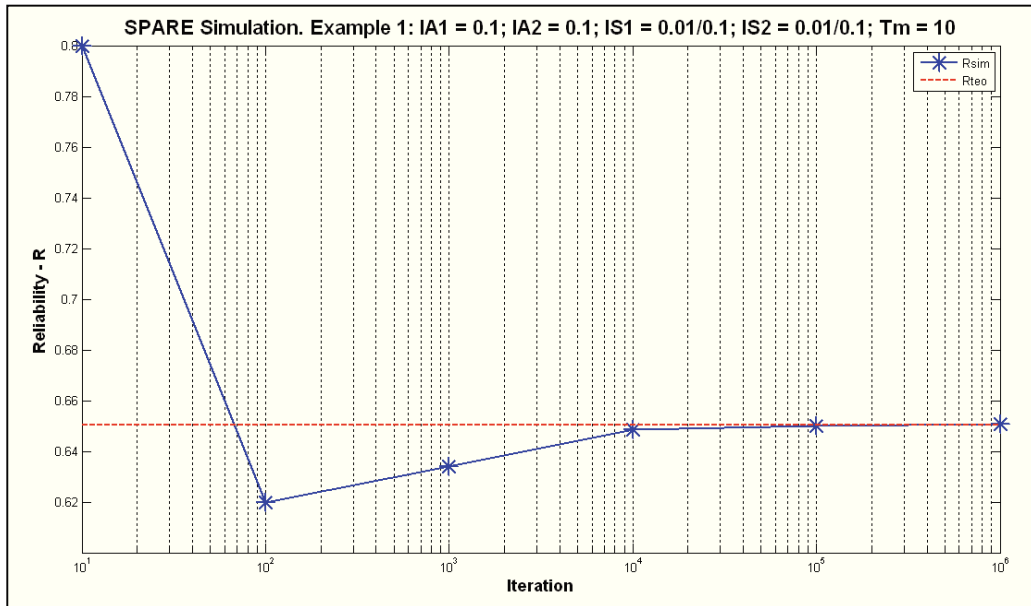


Fig. 4.9. Simulated Vs analytical reliability of a SPARE gate: two active components ($\lambda = 0.1$); two spare components ($\lambda = 0.1$); latency factor $a = 0.1$; mission time $t_m = 10$.. Dotted line: analytical result; marked line: simulated results.

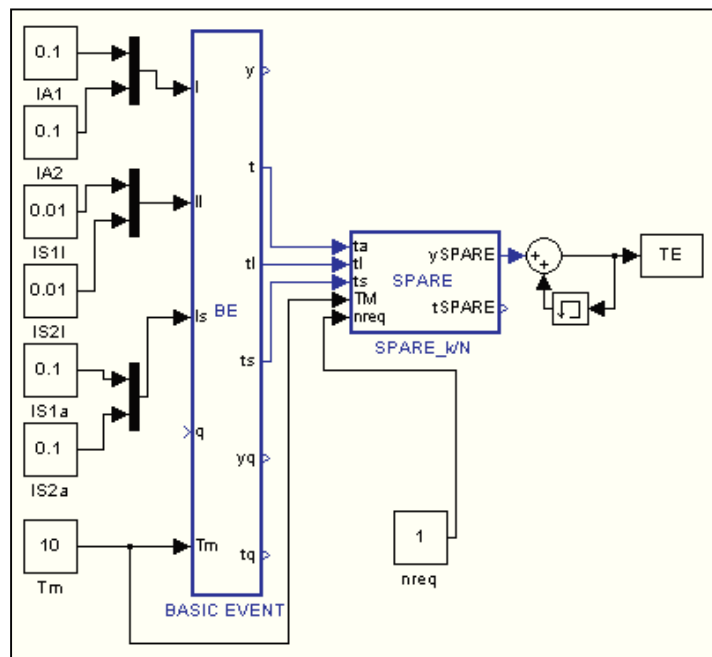


Fig. 4.10. Simulation model of a SPARE_{k/N} gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$; $nreq = 1$.

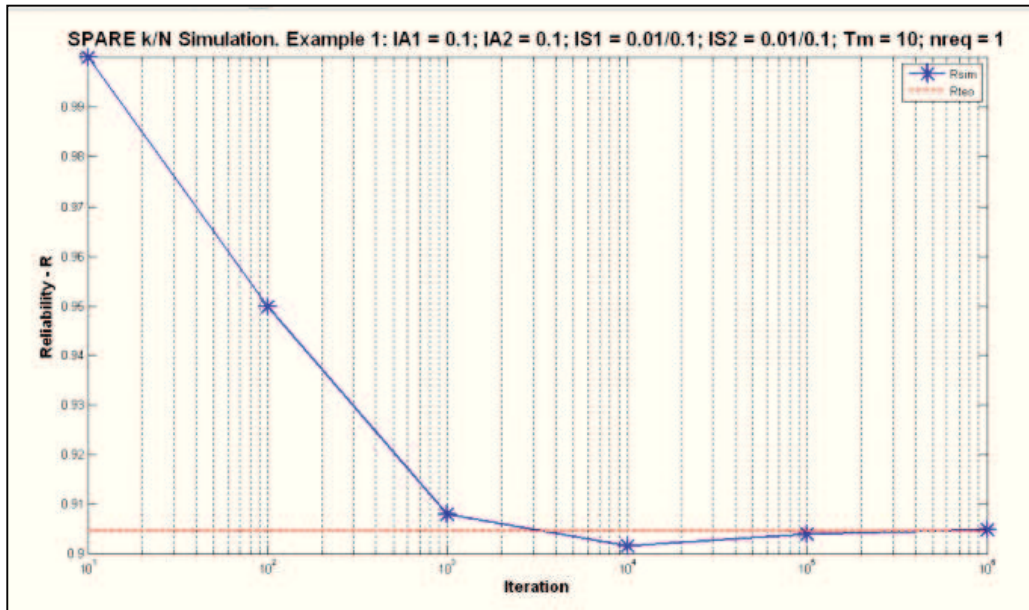


Fig. 4.11. Simulated Vs analytical reliability of a SPARE k/N gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$; $nreq = 1$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

4.3.4 THE SEQ BLOCK

The feature of the SEQ gate is to force the components - inputs of the gate - to move towards the state of failure in a fixed order [10,14,23]. This order is usually expressed graphically by the position of the gate inputs, from left to right. It is generally used to represent different levels of degradation of a component. Therefore, a condition for the gate to trigger is the occurrence of all its inputs. The algorithm used for this task is simple: the SEQ block firstly calculates the sum S of the time to failure of all its inputs. If S is smaller than the mission time the gate triggers with a time to failure equal to S .

4.3.5 THE FDEP BLOCK

The FDEP block models the FDEP gate of a DFT [10,14,23]. The feature of this gate is to force the input components to reach the failure state if the trigger event has occurred before they fail by themselves. The block checks if the failure time of each input component is smaller than the trigger failure time. If the condition is true the component will fail with its own failure time. Vice versa the component will occur with failure time equal to the trigger failure time.

In the construction of a model with a FDEP, each component subjected to the action of the trigger is firstly connected to the FDEP gate and then the output of the latter is passed to the gate interested by the given component. We do not show the flow chart due to the simplicity of the task performed by the block. Likewise we do not show any example for the FDEP block due its similarity with an OR gate between the trigger event and any of the basic event of the gate.

4.4 A COMPARATIVE EXAMPLE

In this section we present a case of study of a real system, in order to demonstrate the effectiveness of the simulation tool to calculate the reliability of such systems. The case of study represents the DFT model of a plant section for the alkylation and treatment of light olefin. Following the top-down procedure of the FT analysis, the tree was designed. The static-tree structure is shown in Figure 4.12 and Table 4.1 reports the component failure rates.

Beside the FT model of Figure 4.12, a DFT was designed in order to consider a more realistic safety behavior that the plant exposes. The dynamic re-arrangement considered concerns the modeling of the gates IE1, IE8 and of the TE. In fact, in the static modeling they are represented with the traditional AND gates. That results in an approximate evaluation of the logic for the real system, since time dependencies cannot be considered with the static-FT. In the DFT, the gate IE8 was substituted with a SPARE gate as in a classic cold stand-by redundant configuration. The second

re-arrangement is done by substituting IE1 with a PAND, in order to consider the priority condition that IE3 has on IE4. The same process is applied at the TE gate.

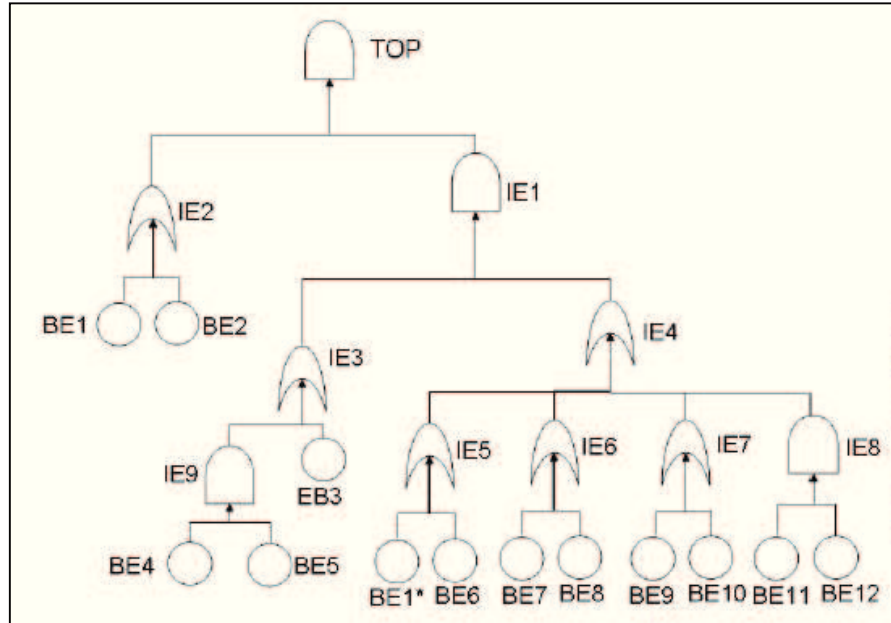


Fig. 4.12. FT of the section plant considered.

Table 4.1. Input data for basic events of the FT of Fig. 4.10.

ID	λ	q
BE1	-	$1.0 \cdot 10^{-5}$
BE2	$9.1 \cdot 10^{-4}$	-
BE3	-	$1.0 \cdot 10^{-7}$
BE4	$1.7 \cdot 10^{-4}$	-
BE5	$7.5 \cdot 10^{-4}$	-
BE6	$9.1 \cdot 10^{-4}$	-
BE7	$4.5 \cdot 10^{-3}$	-
BE8	$8.6 \cdot 10^{-3}$	-
BE9	$4.5 \cdot 10^{-4}$	-
BE10	$7.9 \cdot 10^{-3}$	-
BE11	$1.5 \cdot 10^{-4}$	-
BE12	$9.5 \cdot 10^{-4}$	-

Three cases were studied: (i) simulation of the static-FT, (ii) simulation of the DFT without fixed probabilities by substituting q with the relative failure rate calculated through (3) (i.e., assuming the value of F equal to q and calculating the failure rate through inverse relationship); (iii) the simulation of the DFT with the original parameters of Table 1.

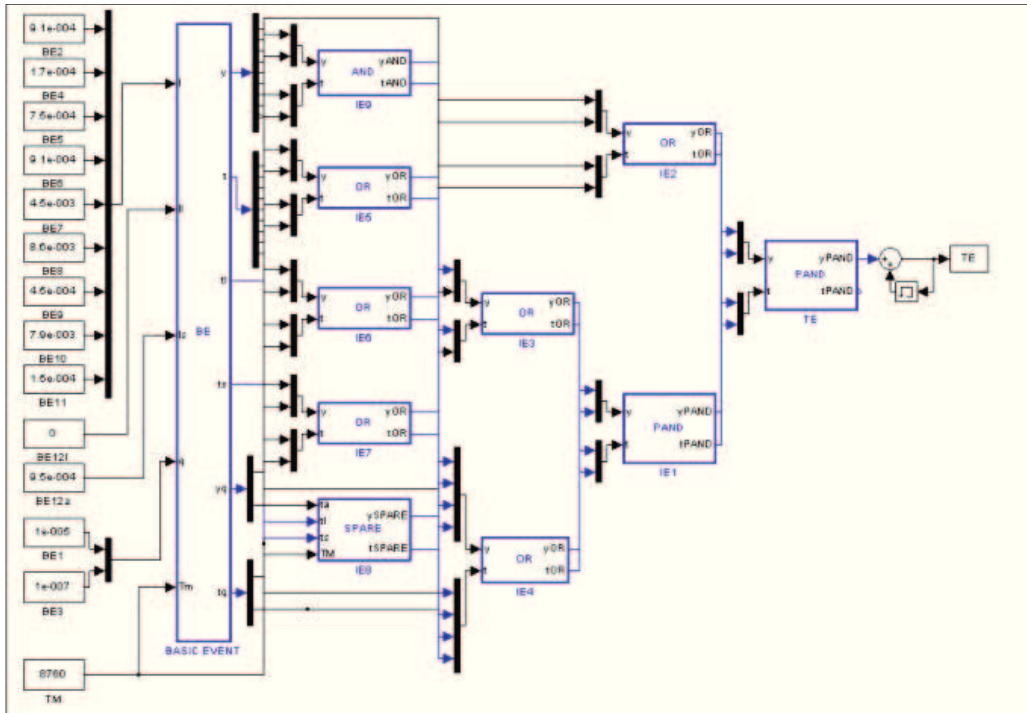


Fig. 4.13. MatCarloRe model of the DFT of Fig. 4.10. Case (iii).

The analytical resolution of these three cases expose different levels of complexity. In fact, the case (i) is the simplest because no time dependencies arise and traditional combinatorial techniques can be used. Case (ii) introduces two kind of dynamic gates. One of them is placed at the TE. It makes impossible the use of techniques to relax the complexity of the model (e.g. modularization [20-30]) without incurring in approximated calculation. The case (iii) can be classified as the most complex since it cannot be solved with the use of the traditional CTMC paradigm due to the presence of fixed probabilities. For this last case no analytical result are presented. The model of the case (iii) implemented in the MatCarloRe tool is shown in Figure 4.13.

We conducted a simulation for each case choosing a maximum number of batches of 10^8 . For cases (i) and (ii) analytical results were computed by the implementation of the model in Relex®. The unreliability of the simulation model converges to the analytical result with 10^3 iterations in the case (i) with a very small relative error. In the case (ii) more iterations are needed to obtain valid results because of the more complex nature of the system involving temporal dependencies. About 10^8 iterations to achieve a small estimating error. In the case (iii) we claim that the number of iterations needed to achieve a small error in case (ii) could be used as well. This is supported by the fact that the unreliability seems to stabilize around 10^8 iterations. Results are reported in Table 4.2.

Table 4.2. Unreliability and relative error for the model MatCarloRe of the FT in Fig. 12. Case(i): SFT with fixed probability for BE1 and BE3; Case(ii): DFT with failure rate for BE1 and BE3; Case(iii): DFT with fixed probability for BE1 and BE3.

Iter	Case (i)	Err.rel%	Case (ii)	Err.rel%	Case (iii)
10¹	0,7000	9,45%	0	100,00%	0
10²	0,8000	3,48%	0	100,00%	0
10³	0,7730	0,01%	0	100,00%	0
10⁴	0,7717	0,18%	1,00E-04	88,39%	0
10⁵	0,7734	0,04%	6,00E-05	13,03%	3,00E-05
10⁶	0,7733	0,02%	6,20E-05	16,80%	4,40E-05
10⁷	0,7732	0,02%	5,58E-05	5,12%	5,49E-05
10⁸	0,7731	0,00%	5,36E-05	0,98%	5,45E-05
Fteo	0,7731		5,31E-05		

4.5 JDFTDES: A JAVA-MATLAB[®] INTEGRATED TOOL

MatCarloRe has one main disadvantage: it is not possible to model complex structures involving repeated spares. In fact, while the SPARE block can handle

multiple primary components that share the same spares, problems arise when more SPARE gates that share the same spare component (of order greater than 1) have other components as inputs that are not the same. Figure 4.14 shows an example of a SPARE system that cannot be modelled by MatCarloRe. In this case, in fact, we cannot use any of the aggregation logic represented in Figure 4.6 or 4.7 because the element B2 can substitute only B but cannot substitute A. Since, MatCarloRe does not admit that spare components are input of more SPARE block, the system in Figure 4.14 cannot be solved by the tool.

To overcome this limitation a Matlab® implementation of the tool was developed. To this end, functions implemented in the blocks of the Simulink® library were converted into Matlab® functions, forming a Matlab® library.

In this way by calling those functions following the hierarchy of the tree is possible to obtain a simulative model that is identical to the one that can be modelled by MatCarloRe.

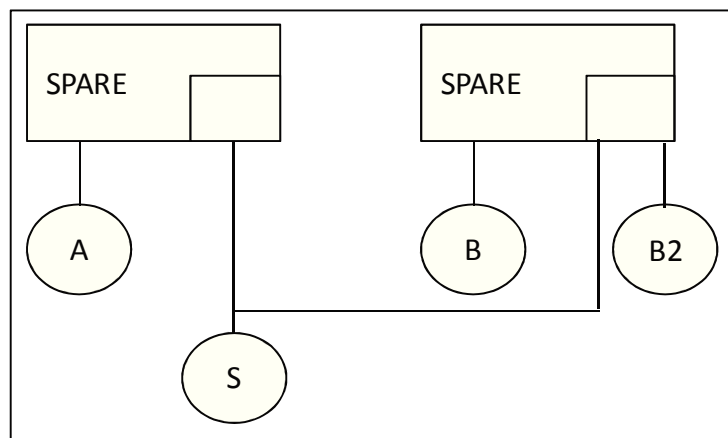


Fig. 4.14. Example of SPARE system that cannot be modelled by MatCarloRe.

To solve the mentioned problem we defined a new function called ALL_SPARE that evaluate iteratively all the SPARE functions that are implemented in the model. By the mean of the ALL_SPARE gate, SPARE gates can share information about the occupancy of a spare component in a specific SPARE gate, thus allowing SPARE gates to communicate to each other.

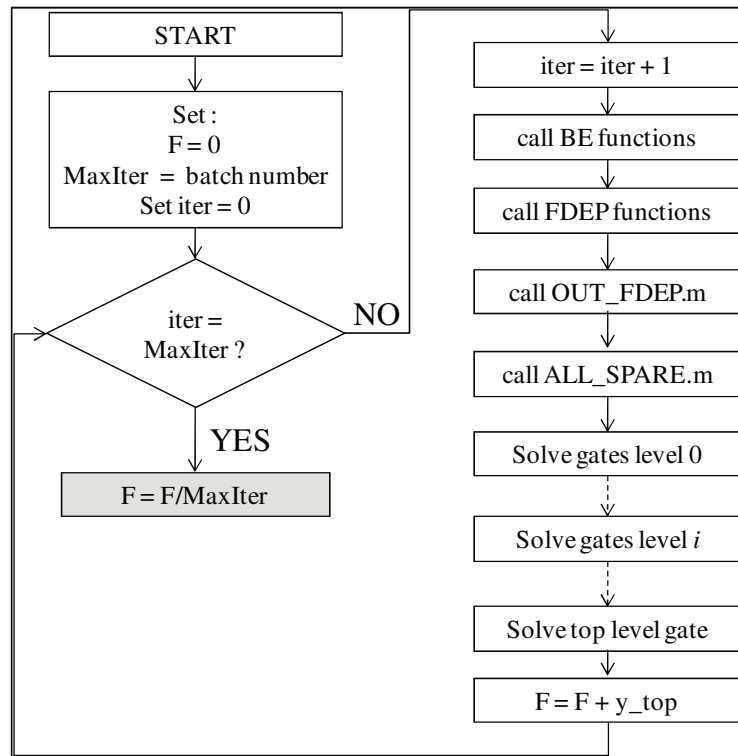


Fig. 4.15. Algorithm of the resolution of a DFT with the Matlab® library.

The algorithm for the resolution of a DFT by the mean of the Matlab® library is shown in Figure 4.15. The figure shows that after the computation of the sampled time to failure of BEs, FDEP gates are evaluated. In this way the corrected time to failure of elements inputs of FDEP gates are assigned. Successively, the ALL_SPARE function is called, which in turn calls recursively a number of SPARE functions equal to the number of SPARE gates in the model. When this is done the remaining gates are evaluated by the mean of the supporting functions in a bottom-up procedure until the top gate. This procedure is nested into a for loop that counts for the number of batches.

While solving modelling and evaluation concerns the Matlab® library results very difficult to organize when it comes to model a system. To this end a JAVA graphical interface that support the construction of the DFT was developed. By the mean of the interface the model of the system is generated in terms of a Matlab® functions where all the necessary functions of the Matlab® library are automatically organized.

4.5.1 JAVA INTERFACE

The creation of a DFT model with the MatCarloRe syntax can be error prone and tedious. In Figure 4.15 it is shown the flow chart that describes how to prepare the Matlab® script. At first, the initialization of the main variables (F, number of iterations) is needed. Then, the simulation runs inside a loop that contains the commands that refer to the DFT model; the functions of the library have to be invoked in the order shown in Figure 4.15, according with the structure of the DFT. The «BE functions» need to be invoked for first, in order to sample the correspondent times of failure of each BE; the «FDEP functions» and «OUT_FDEP functions» are called if the DFT contains FDEP gates, in order to refresh the time of failure of the BEs which are connected with such kind of gate. Hence, if there are spare gates which share spare components, they have to be assembled invoking the «ALL_SPARE function» function. Once this first part of the code is written, it is possible to call all the other functions (which represent the gates) following a bottom-up approach since the information of the lower level of the fault tree are requested to the upper gates. In the end, the unreliability is computed as the ratio between the number of TE occurrence over the number of iterations.

In order to simplify the effort of typing the code of a model with the MatCarloRE syntax, a graphic user interface (GUI), the jDFTDes, was developed. The jDFTDes is a code processor that translates a graphic model of DFT in a program for the MatCarloRE engine. The jDFTDes stands for “Java® DFT Designer”, a java package that can be invoked directly under the Matlab® shell. The choice of using Java was natural, since Matlab® runs under a Java Virtual Machine (JVM) and this permits to use the Java interpreter and run programs written in Java. In our application, we created a Java Archive (JAR), a Java file that includes all the classes of the jDFTDes. The jDFTDes can be invoked through the “javaaddpath(°)” command, specifying the path where the library is located and creating a dummy variable that contains an instance of the java main frame of the jDFTDes. jDFTDes is greatly simple and the construction of a DFT model is straight, easy and fast. This was accomplished implementing a “drag and drop” interface that permits a quick

interactions with the element of the DFT. In fact, by clicking the right button it is possible to change the property of a component (rename, modify the type of component and change the order of the dependency) while by clicking the left button it is possible to add an input (if the component clicked is a gate) or specify the type of CDF if the component is a BE.

The text field T_m must contain the value of the time mission and the text field “ITERATION” the number of batches requested for the simulation. Once the DFT is assembled and the input are correctly set, by clicking “COMPUTE” the jDFTDes will process the graphical model generating the code for the MatCarloRE and finally dumping it in the Matlab® shell.

4.6 CONCLUSION

In this chapter we summarized an integrating technique of Monte Carlo simulation and FT methodology for reliability assessment of complex systems in presence of time dependencies. Our simulating environment can go beyond the limitations of analytical methodologies with the additional advantage of a high level modeling interface based on the FT method.

With the current implementation of the tool DFT with general distributions can be solved via simulation. Moreover the tool has shown its effectiveness with respect to commercial and academic tool for the resolution of DFT. In Appendix A an accepted paper is reported where comparison with known tool is performed on the basis of the kind of the supported distributions as well as in terms of simulation time. Currently, MatCarloRe and its most recent version (Java-Matlab) are still under development. Two are the research lines that have been considering: the first is to exploit the use of variance reduction techniques to speed up the simulation [15-19]. First results have shown that these techniques can be applied effectively when non repairable components are considered. Second, we want to achieve the modeling capabilities introduced in Extended-DFT [39] where the following limitations are overcome:

- FDEP gates cannot accept input that are not BEs;
- SPARE gates cannot accept inputs that are not BEs.

However, the tool carries on the limitations of DFTs in terms of modelling power:

- the kind of behaviour that can be modelled depends on the gates that are used in the formalism (as an example is not possible to model shared load, dependencies that are not related to fault logics etc.)
- it is not possible to evaluate the availability of the system since the recovery logic was not formalized;
- it is not possible to evaluate measure like reliability with repairs or more complex behaviours that depend on the occurrence of the failure of subparts of the tree.

To overcome this limitations we recognize the need of a methodology that offers more degree of freedom to modeller in describing dependencies without losing the benefit of a high level language of description. In order to address these issues we recognize the necessity to separate the behaviour of the system from the fault logic into two separate and interconnected model. The behavioural model is a description of the behaviour of the system while the fault model represents the configurations that bring to the system failure. The two model in additions are interconnected in the sense that the behavioural model must pass information about the state of the components to the fault model and the information retrieved by the evaluation of the fault model can be passed back to the behavioural model in order to allow more complex behaviours. The fault model, free from the behavioural part, is static and can be represented by a FT of a RBD for instance.

However, this implementation needs the formalization of a new modelling semantic. We will show that with the introduction of Adaptive Transition Systems (ATS) these limitations can be overcome giving to the modeller a flexible formalism to model systems with such a behaviour.

BIBLIOGRAPHY

- [1] Roberts, N. H. , Vesely, W. E., Haasl, D.F., & Goldberg, F.F. (1981). Fault tree handbook, NUREG-0492. Washington: US NRC.

- [2] Ren, Y., & Dugan, J. B. (1998). Design of reliable systems using static and dynamic fault trees. *IEEE Transactions on Reliability*, 47, 234-244.
- [3] Siu, N. (1994). Risk assessment for dynamic systems: an overview. *Reliability Engineering and System Safety*, 43, 43-73.
- [4] Cepin, M., & Mavko, B. (2002). A dynamic fault tree. *Reliability Engineering and System Safety*, 75, 83-91.
- [5] Amari, S., Dill, G., & Howald, E. (2003). A new approach to solve dynamic fault trees. *Annual Reliability and maintainability symposium*, 374–379.
- [6] Distefano, S., & Puliafito, A. (2007). Dynamic reliability block diagrams vs dynamic fault trees. In *Proceedings Annual Reliability and Maintainability Symposium RAMS '07*, Jan. 22-25, 71-76.
- [7] Dugan, J. B., Venkataraman, B., & Gulati, R. (1997). Diftree: a software package for the analysis of dynamic fault tree models. In *Proceedings Annual Reliability and Maintainability Symposium*, Jan. 13-16, 64-70.
- [8] Dugan, J. B., Trivedi, K. S., Smotherman, M. K., & Geist, R. M. (1986). The hybrid automated reliability predictor. *Journal of Guidance, Control, and Dynamics*, 9, 319-331.
- [9] Sullivan, K. J., Dugan, J. B., & Coppit, D. (1999). The galileo fault tree analysis tool. In *Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, June 15-18, 232-235.
- [10] Boudali, H., Crouzen, P., & Stoelinga, M. (2007). Dynamic fault tree analysis using input/output interactive markov chains. In *Proceedings 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks DSN '07*, June 25-28, 708-717.
- [11] Montani, S., Portinale, L., Bobbio, A., & Codetta-Raiteri, D. (2008). RADYBAN: A tool for reliability analysis of dynamic fault trees through conversion into dynamic bayesian networks, *Reliability Engineering and System Safety*, 93, 922–932.

- [12] Bobbio, A., Portinale, L., Minichino, M., & Ciancamerla, E. (2001). Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. *Reliability Engineering and System Safety*, 71, 249–260.
- [13] Volovoi, V. (2004). Modeling of system reliability petri nets with aging tokens. *Reliability Engineering and System Safety*, 84, 149–161.
- [14] Durga Rao, K., Gopika, V., Sanyasi Rao, V. V. S., Kushwaha, H. S., Verma, A. K., & Srividya, A. (2009). Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliability Engineering and System Safety*, 94, 872–883.
- [15] Marsaguerra, M., Zio, E., Devooght, J., & Labeau, P. E. (1998). A concept paper on dynamic reliability via Monte Carlo simulation. *Mathematics and Computers in simulation*, 47, 371–382.
- [16] Marquez, A. C., Heguedas, A. S., & Iung, B. (2005). Monte Carlo-based assessment of system availability. A case study for cogeneration plants. *Reliability Engineering and System Safety*, 88, 273–289.
- [17] Zio, E., Marella, M., & Podollini, L. (2007). A Monte Carlo simulation approach to the availability assessment of multi-state systems with operational dependencies. *Reliability Engineering and System Safety*, 92, 871–882.
- [18] Zio, E., Podofillini, L., & Levitin, G. (2004). Estimation of the importance measures of multi-state elements by Monte Carlo simulation. *Reliability Engineering and System Safety*, 86, 191–204.
- [19] Marseguerra, M., & Zio, E. (2004). Monte Carlo estimation of the differential importance measure: application to the protection system of a nuclear reactor. *Reliability Engineering and System Safety*, 86, 11–24.
- [20] Chatterjee, P. (1975). Modularization of fault trees: A method to reduce the cost of analysis. *SIAM Reliability and Fault Tree Analysis*, 101–137.
- [21] Rosenthal, A. (1980). Decomposition methods for fault tree analysis. *IEEE Transactions of Reliability*, R-29, 136–138.

- [22] Khoda, T., Henley, E. J., & Inoue, K. (1989). Finding modules in fault trees. *IEEE Transactions on Reliability*, 38, 165-176.
- [23] Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1992). Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Transactions on reliability*, 41, 363-377.
- [24] Dugan, J. B., Sullivan, K. J., & Coppit, D. (2000). Developing a low cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transaction on reliability*, 49, 49–59.
- [25] Meshkat, L., Dugan, J. B., & Andrews, J. D. (2002). Dependability Analysis of Systems With On-Demand and Active Failure Modes, Using Dynamic Fault Trees. *IEEE Transactions on reliability*, 51, 240-251.
- [26] Anand, A., & Somani, A. K. (1998). Hierarchical analysis of fault trees with dependencies, using decomposition. *Proceedings Annual on Reliability and Maintainability Symposium*, 69–75.
- [27] Huang, C. Y., & Chang Y. R. (2007). An improved decomposition scheme for assessing the reliability of embedded systems by using dynamic fault trees. *Reliability Engineering System Safety*, 92, 1403–1412.
- [28] Lanus, M., Yin, L., & Trivedi, K. S. (2003). Hierarchical Composition and Aggregation of State-Based Availability and Performability Models. *IEEE Transactions on reliability*, 52, 44-52.
- [29] Feinberg, B. N., & Chiu, S. S. (1987). A Method to Calculate Steady-State Distributions of Large Markov Chains by Aggregating States. *Operations Research*, 35, 282-290.
- [30] Malhotra, M., & Trivedi, K. S. (1993). A methodology for formal specification of hierarchy in model solution. In *Proceedings Fifth International Workshop Petri Nets and Performance Models*, (PNPM-1993), 258–267.
- [31] Windebank, E. (1983). A Monte Carlo Simulation Method Versus a General Analytical Method for Determining Reliability Measures of Repairable Systems. *Reliability Engineering*, 5, 73-81.

- [32] Goldfeld, A., & Dubi, A. (1987). Monte Carlo Methods in Reliability Engineering. *Quality and Reliability Engineering International*, 3, 83-91.
- [33] Dubi, A. (1989). Monte Carlo Methods in Reliability. Operation Research and System Engineering Commission of the European Communities Joint Research Centre, Ispra Italy.
- [34] Lewis, E. E., & Bohm, F. (1984). Monte Carlo Simulation of Markov Unreliability Models. *Nuclear Engineering and Design*, 77, 49-62.
- [35] Dubi, A. (1986). Monte Carlo Calculations for Nuclear Reactors. *CRC Handbook of Nuclear Reactors Calculations*, Vol. II, CRC PRESS.
- [36] Wu, Y. F., & Lewins, J. D. (1992). Monte Carlo Studies of Engineering System Reliability. *Annual nuclear engineering*, 19, 825-859.
- [37] Zio, E. (1995). Biasing the transition probabilities in direct Monte Carlo. *Reliability Engineering and System Safety*, 47, 59-63.
- [38] Relex. URL: <http://www.ptc.com/products/windchill/quality-solutions/>
- [39] DFTsim. URL: <http://wwwhome.cs.utwente.nl/~marielle/papers/BNS09.pdf>

CHAPTER 5

5.1 INTRODUCTION

It is often possible to represent the behaviour of a system by specifying a discrete number of states it can occupy and by describing how the system moves from one state to another as time progresses. We take the view that the evolution of a system can be the result of *interaction* among different parts of the system.

This kind of approach, based on communicating (or interacting) transition systems, has been used in Stochastic Process Algebras (SPA), e.g., Performance Evaluation Process Algebra (PEPA), Interactive Markov Chains (IMC), etc. [1,2]. Here, the communication mechanism between models of the system interacting parts is based on synchronization of transitions that share the same name (like in the CSP approach [3]), i.e., state changes occur simultaneously across the models of interacting parts.

In PEPA synchronization between timed transitions (exponential) is defined taking the minimum between the rates of synchronized transitions, while in IMC the problem is solved distinguishing timed transitions from synchronizing transitions (instantaneous). These approaches are limited to Markov models. IMC supports also phase-type distributions.

Among SPA extensions to non Markovian models is Interactive generalized Semi Markov Processes (IGSMP) [4]. However the solution for these methods is very expensive, thus one usually restores to simulation, e.g., the SPADE language [5].

A different approach has been used in the PRISM language (a stochastic and probabilistic model checker [7]), where transition systems are replaced by modules, states by variables and transitions by functions of these variables [6,8]. There can be more than one variable for each module, and each module can update the values of its internal variables. PRISM allows synchronization of transition, too, but in this case the rate of the transition is chosen as the product of individual rates of the synchronized transitions.

PRISM introduces also guards on transitions, i.e., Boolean expressions that define the conditions for which a transition (a command in the PRISM language) may complete. Guards are functions of the variables defined in the model; also of variables belonging to other modules. PRISM lacks of a graphical interface, but is very powerful due to the possibility of specifying variables and synchronization. Again PRISM can be used only when the underlying process is Markov.

In the context of model checking, when general distributions need to be accounted into the model, this has brought to statistical model checking, i.e., solved via simulation [10]. One relevant work is that of [9] where a statistical model checking is implemented based on the definition of a DESP. DESP resemble the definition of generalized semi-Markov process (GSMP) given in [11].

Another well known class of formalisms that have been widely used in dependability modelling are stochastic extensions of Petri nets (SPN) [12]. They allow one to analyze systems with various dependencies between the modelled elements and to use a range of distributions to model their stochastic behaviour. We have seen in Chapter 3 that Stochastic Activity Networks (SANs), an extension of SPN, allow compositional modelling, overcoming to one of the main limits of SPN in case of large and complex systems [13]. These formalisms represent the system in terms of a network composed of: places, containing tokens; and transitions, whose function is to change the marking of the net, i.e., the number of token in a place.

In SAN the modeller can define predicates of arbitrary complexity using directly a high level language (e.g., C++ in the Mobius tool [14,26]). Predicates are as guards in the PRISM, inhibiting transitions if the predicate is false. In most cases the predicates implement a simple Boolean function on set of state variables maintained in the model. Moreover, SANs allow to define parameter of transition dependent on the marking of the net. This feature is not included, for instance, in the previous mentioned formalism (PEPA, IMC, PRISM).

Inspired by the indicated formalisms, we introduce a modelling approach for dependability analysis of complex interdependent systems where temporal dependencies have a central role. In the mentioned approaches we argue that temporal dependencies based on the ordering of occurrence of transitions can be modelled with increased complexity of the model itself. The resulting model is very large and one cannot completely abstract from the complexity of the interaction between the modelled elements.

We model systems in terms of interacting transition systems (TS) with stochastic features [15]. The model consists of a set of TSs, each representing the state space of an element modelled in the systems. A TS can model a component as well as a specific dimension of the state space of the element. The main value of the approach is the *separation of concerns*: the modeller and the problem domain experts may benefit from being able to focus on the specific aspect of the state-space of a modelled element and abstract out the complexity of its entire state-space of the system.

The class of interactive transition systems that we define are *Adaptive Transition System* (ATS). ATS is an high level modelling formalism that provides a concise and compositional way to describe the behaviour of interdependent reactive systems with general time distributions. Non determinism can be included in the model as well. Different kind of synchronization procedures can be defined, too. *In ATS different parts of the system are modelled by different transition systems* that are said adaptive because they adapt to each other according to their relative evolution as time progresses.

Adaption is a consequence of the kind of dependencies existing between parts of systems. In ATS we define a formal language to represent two kinds of dependencies that may exist in a system:

- *state dependencies*, where the behaviour of interdependent systems is subjected to the different states they may be in; and
- *temporal dependencies*, where the behaviour of interdependent systems is subjected to ordering of occurrence of state-transitions.

From a modelling point of view, these two kinds of dependencies require different information about the state of the system parts. In fact, state dependencies can be modelled considering only indicator functions of the states occupied by the modelled elements. On the other hand, modelling temporal dependencies requires to know whether a defined sequence of actions has taken place.

Another way to classify dependencies is in terms of impacts, i.e., the effects that a state or temporal dependency may have on the system behaviour. We model impacts by defining a *model of transition* that allows to capture the following behavioural aspects that may change as the system evolves in time:

- the existence or not of a transition between pair of states;
- the change in the mathematical law that define the *time to complete* of a transitions;
- the change in the values of the parameters of these laws; and
- the possibility to restart a transition.

It is by the model of transitions that ATSS adapt their behaviour with respect to the evolution of other parts of the system as time progresses. In order to allow these possible behaviours we assign to transitions a set of attributes similar to those given in SAN for activities, i.e., *type*, *activation*, *reactivation* and *parameter*. In the ATS language *attributes of transitions* are variables that are defined by opportune functions, that we call *transition functions*.

Transition functions define the communication mechanism between ATSs. Outputs of transition functions are *transition variables*, i.e., *activation, reactivation and parameter variables*, and inputs are *system variables*.

System variables are a reading of the system state at a given time. They are outputs of *system functions*, a set of functions defined over the evolution of the system. In this dissertation we give the definition of three kind of system functions that define three kind of system variables. However, the class of system functions and variables could be easily extended. Here we present system variables that give the following information about the state of a system:

- the state occupied by each ATS;
- the last completed transition among all ATS models; and
- the time of completion of each transition when it last occurred.

These system variables are said: *state indicator, transition indicator and transition timing variables*, respectively. *State indicator variables* and *transition indicator variables* can be used to define the class of state dependencies described above, while *transition timing variables* can be used to define temporal dependencies. They can impact on any of the attributes of transitions described above, although *transition indicator variables* are generally used only as inputs of *reactivation functions*.

The specific structure of transition functions is defined by the modeller. It is by the specification of such structures that dependencies among parts of the systems are modelled. Transition functions take the last values of system variables, i.e., the value of these variables after the most recent state-change, to update the value of transition variables, thus allowing ATS to adapt to changes in the system (Figure 5.1).

Due to the fact that only the last value of system variables, i.e., the value at the most recent state-change, the class of temporal dependencies that can modelled through the definition of transition functions is restricted to the most recent ordering of events. However, even with this limitation a wide class of systems can be represented. In this setting, each transition can be seen as a corridor that moves at point in time. At each completion of a given transition, the related corridor moves

forward to the new position. In this way the modeller can benefit from the knowledge of the relative position of corridors when state changes occur.

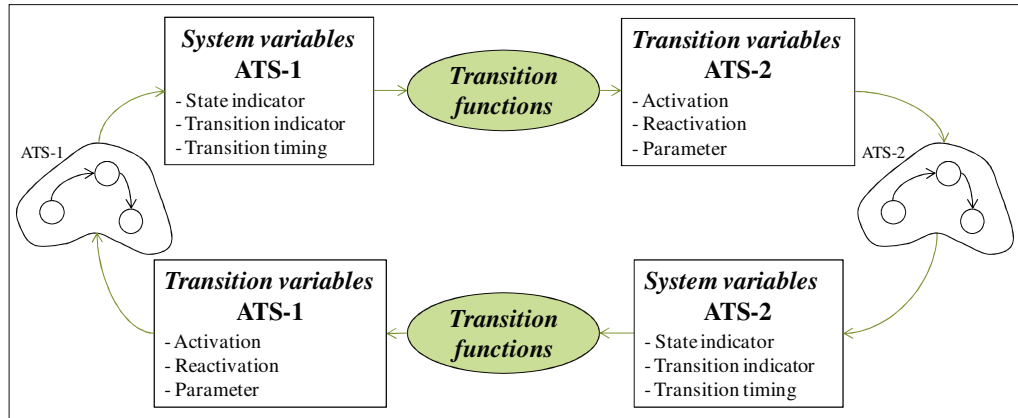


Fig. 5.1. Communication mechanism between ATS models.

Properties of the system are studied by the definition of *reward variables*. Reward variables are outputs of *reward functions* and inputs of reward functions are system variables.

In our framework we extend the notion of reward variables. In fact we allow reward variables to be inputs of transition functions. Thus, not all the reward functions that are defined in the model are used as a mean to evaluate performance indexes of the system. While in the latter case one wish to estimate the expected value of these variables; in the case they are used as inputs of transition functions they are seen as variables that highlight composite information about the system (Figure 5.2).

In fact, since reward functions take system variables as inputs they can be seen as a composite reading of the state of the system. They express an higher level of abstraction.

Reward variables and systems variables are input of transition functions. The main consequence of this choice is to move the complexity of modelling dependencies to the model of transitions. In fact, defined the ATS of the interacting elements in terms of state and transitions the definition of transition and reward functions can be done subsequently. In this way one benefits of abstracting from the model of the other system parts when building the different ATSs.

Moreover, we show that, with the aid of a Fault Tree [16] like formalism, where gates are generalized to include not only Boolean operators, transition and reward functions can be graphically represented, improving the readability of the model.

We argue that also the task of debugging and maintain a model are facilitated by the graphical representation of transition functions, and thus of dependencies between the modelled elements. This, for instance, in comparison to Petri Nets is a major advantage since in these models there is often a lack of standardized approach to modelling (the model depends from the modeller perspective).

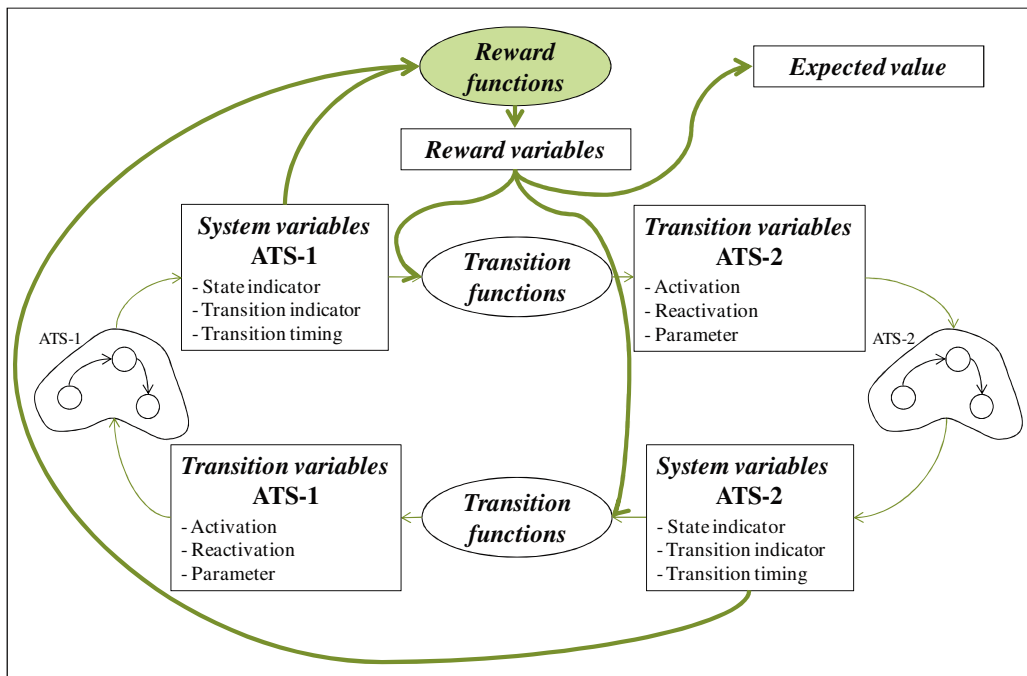


Fig. 5.2. Relations between variables.

To build a state-space model of ATS that can be solved analytically we propose a definition of state based on *system variables* with constrain on the class of possible structure of transition and reward functions.

However, transition functions and systems variables can be of any kind. In this case by giving ATS with an *execution logic similar to the one defined for GSMP* we show how a simulation model can be implemented directly from the execution rules of ATS.

Compared to IMC, we do not make use of the powerful formalism of SPA to address problems related to the size of the state-space. Our intent is to define a formalism that offers a structured approach to modelling complex systems that can be easily implemented in a simulation language and under some constrain and variables-transformation can be solved analytically via the generation of the underlying Markov chain.

The state-space of an ATS can be built by an algorithmic procedure that resemble the construction of a reachability graph in SPNs. Due to the timing relation the retrieved state-space can be very large even in case of small systems. We give some indication to achieve a reduction by an appropriate definition of the state where only those variables influent for the evolution and for the estimate of a performance measure are considered.

In comparison to Petri net, however, probabilistic choice has not yet been defined in ATS. However the extension seems straightforward. Synchronization could be also be defined for future development of the formalism. Although we do not give a formal definition of synchronization between ATS, we show how it can be implemented with the aid of a case study based on a real system. Finally, we argue that a SPA formalization of ATS can be achieved, with the results of improving the techniques used for the analytical solution of the system.

The remainder of this chapter is structured as follows: Section 2 gives a formal definition of ATS with a detailed explanation of the variables defined above and their possible relations.

In Section 3 we present Ordered-ATS (OATS) a class of ATS with constrains on the structure of transition and reward functions. Within Section 3 we introduce a Fault Tree like formalism to the graphical representation of these functions.

In section 4 we present a contrived example, the heat-power system, a system composed of four components with various kind of functional dependencies. Although the example does not exploit the full range of modelling capabilities of ATS, it shows how build the model of transition in a tutorial fashion.

Section 5 concerns with the evolution of ATS. We define the process of completion of a transition considering non deterministic choices and race conditions between transition.

In Section 6 we introduce Markov-OATS (MOATS) a class of OATS for which a Markov model can be retrieved. We give the general rules for the generation of the state-space model defining an appropriate notion of state.

Section 7 gives the general rules for solving the model via simulation. We show a case study based on the POWER-TELCO network nearby Rome Fiumicino (IRIIS Project [17,18]). In this case study ATS assume a very general form. In fact, one kind of reward functions represents the power associated with a node of the network and transition functions take as inputs these values. The power associated with the nodes is evaluated by solving the balance equations of the network, i.e., Kirchhoff circuit laws, given the absence of some node due to a failure. As a result some new node can become disconnected from the network due to overloading or missing of current. This process taking place into the Power network has effect on the batteries that supply energy to the Telco nodes.

Finally, in Section 8 are reported some related works, conclusion and possible lines of further study.

5.2 ADAPTIVE TRANSITION SYSTEM

Definition 5.1. An ATS is a 5-tuple $ATS = (S, f_T, f_S, f_R, q(0))$, where:

- $S = \{S_1, \dots, S_N\}$ is a finite set of N transition systems. The i -th element of S is specified in terms of a finite set of NQ_i states $Q_i \in Q$ ($i = 1, 2, \dots, N$) and a finite set of NT_i transitions $T_i \in T$ ($i = 1, 2, \dots, N$) between pairs of states. Q and T are the set of states and transitions of the overall transition system S , respectively. Elements of Q_i are denote as $Q_{i,j}$ ($j = 1, 2, \dots, NQ_i$) and elements of T_i as $T_{i,j \rightarrow j+1}$ ($j = 1, 2, \dots, NT_i - 1$) or as $T_{i,k}$ ($k = 1, 2, \dots, NT_i$).

- $f_S: S \rightarrow V_S$, is a set of system functions that define the state of the system at a given time. We represent the system state in terms of three kinds of system functions. $f_S = \{s, t, \pi\}$, where:
 - $s: Q \rightarrow [0,1]$ are said *state indicator functions* (associated with each state $Q_{i,j}$);
 - $t: T \rightarrow [0,1]$ are said *transition indicator functions* (associated with each transition T_{ik}); and
 - $\pi: T \rightarrow \mathcal{R}$ are said *transition timing functions* (associated with each transition T_{ik}).
- $f_R: V_S \rightarrow \mathcal{R}$, is a set of reward functions defined over the set of system variables $v_S \in V_S$;
- $f_T: V_S \times V_R \rightarrow V_T$ is a set of transition functions defined over the set of system and reward variables, $v_S \in V_S$ and $v_R \in V_R$, respectively. Transition functions are of three kinds. $f_T = \{a, r, \theta\}$, where:
 - $a: V_S \times V_R \rightarrow [0,1]$ are said *activation functions*;
 - $r: V_S \times V_R \rightarrow [0,1]$ are said *reactivation functions*; and
 - $\theta: V_S \times V_R \rightarrow \mathcal{R}$ are said *parameter functions*.
- $q(0) \in Q$, is the initial state. $q_i(t) \in Q_i$ is the *state variable* of S_i at time t . As the system moves at point in times, we are interested in $q_i(n)$, the values of these variables as a state-transition occurs. With $q_i(n) = Q_{i,j}$ we say that during the interval $I_n = [\tau_n, \tau_{n+1}[$, S_i is in state $Q_{i,j}$.

5.2.1 SYSTEM VARIABLES

As said in Definition 5.1 system functions $f_S = \{s, t, \pi\}$ are of three kinds:

- $s: Q \rightarrow [0,1]$ are said *state indicator functions* and are defined for each state $Q_x \in Q$. Given a state $Q_{i,j} \in Q_i$, we define the state indicator function of $Q_{i,j}$ in the time interval $I_n = [\tau_n, \tau_{n+1}[$ as:

$$s_{ij}(n) = \begin{cases} 1, & \text{if } q_i(n) = Q_{i,j} \\ 0, & \text{otherwise} \end{cases}. \quad (5.1)$$

- $t: T \rightarrow [0,1]$ are said *transition indicator functions* and defined for each transition $T_x \in T$. Given a transition $T_{i,j \rightarrow j+1} \in T_i$, we have that:

$$t_{i,j \rightarrow j+1}(n) = \begin{cases} 1, & \text{if } q_i(n) = Q_{i,j+1} \wedge q_i(n-1) = Q_{i,j} \\ 0, & \text{otherwise} \end{cases}, \quad (5.2)$$

for $n > 0$ and $t_{i,j \rightarrow j+1}(0) = 0$. Thus, transition indicator function take on value 1 only if the transition is the one that most recently completed. The output of this function, can be analyzed also from the point of view of the states that the transition connects.

In fact, if $t_{i,j \rightarrow j+1}(n) = 1$ we retrieve the following information: $Q_{i,j}$ is the state that was left at the n -th time step (or the state that most recently was left) and that $Q_{i,j+1}$ is the state that was entered at the n -th time step (or the state that most recently was entered). On the other hand, if $t_{i,j \rightarrow j+1}(n) = 0$ we can only state that $Q_{i,j}$ and $Q_{i,j+1}$ were not involved in any state-transition at the n -th time step.

Transitions indicator functions can be seen as messages that are sent when a transition complete. In ATS, the presence of instantaneous and deterministic transitions can cause that more transitions complete at the same time (concurrent transitions). Although occurring at the same time, their completions occur at different time steps. Two cases are possible: (i) transitions are consecutive in the same ATS; or (ii) transitions belong to different ATS or are not consecutive. In the latter case, the modeller should be aware of the non-deterministic choice that is made on the ordering of completion of concurrent transitions. It is important to remind this fact in the context of transition indicator variables because they are impulse variables, i.e., they take on value 1 as long as a transition was the last that completed.

- $\pi: T \rightarrow \mathcal{R}$ are said *transition timing functions* and as transition indicator functions are assigned to each transition $T_x \in T$. Given a transition $T_{i,j \rightarrow j+1} \in T_i$, we have that:

$$\boldsymbol{\pi}_{i,j \rightarrow j+1}(n) = \begin{cases} \tau_n, & \text{if } \mathbf{s}_{i,j \rightarrow j+1}(n) = 1 \\ \boldsymbol{\pi}_{i,j \rightarrow j+1}(n-1), & \text{otherwise} \end{cases}, \quad (5.3)$$

for $n > 0$ and $\boldsymbol{\pi}_{i,j \rightarrow j+1}(0) = 0$. Transition timing functions returns the value of the most recent time point at which a given transition completed.

From eq. (5.3) we see that the value of transition timing variables is kept the same until the transition completes again. Moreover, the value of these variables is set to 0 at the initial time. Transitions are seen as events that move along the real axis as corridors in a race. With this setting these variables express a relative ordering between transition, independently of the states they connect.

Again, from the perspective of entered and left states, $\boldsymbol{\pi}_{i,j \rightarrow j+1}$ returns the most recent time point value at which $Q_{i,j}$ was left and $Q_{i,j+1}$ was entered. Moreover, since the value of $\boldsymbol{\pi}_{i,j \rightarrow j+1}$ is kept the same until a new completion of $T_{i,j \rightarrow j+1}$, the information about the time $Q_{i,j+1}$ was last entered is present in the model even if the system has moved out from $Q_{i,j+1}$.

This is true in the case of a state with only one ingoing and outgoing transition, but it is not in general, i.e., a state with more ingoing or outgoing transitions. In the general case to retrieve information about the time a state was last entered, one should evaluate the maximum value of the transition timing functions of those transitions directed to the state. In these cases, were system variables must be related together in order to retrieve measure at higher levels, reward functions must be considered.

5.2.2 REWARD FUNCTIONS

Reward functions are defined over the set of possible values of system functions. Reward functions are commonly used to study some property of interest of the system. From a general perspective reward functions can be seen as relations that combine information of different parts of a system in order to generate a specific output. Reward variables, the outputs of reward functions, can be used as a specification of a measure of interest or simply considered as inputs of transition

functions. In the latter case they are used to define the behaviour of the system with higher levels of detail.

If a reward is used as a mean to evaluate a performance measure, three kind of reward categories can be considered: instant of time, interval of time and averaged interval of time. Another categorization that is often given about rewards is that they can be rate rewards or impulsive rewards. Rate rewards are related to the occupancy of a particular state, while impulse rewards are related to the completion of transitions.

ATS allow to model both rate rewards and impulse rewards. Inputs of rate reward functions are state indicator functions and transition timing functions and inputs of impulse reward functions are transition indicator functions. In addition these two kind of rewards can be implemented together in a single reward function. More specification about rewards classification can be found in [19].

We allow reward variables to be inputs of transition functions. This follows considerations like: to evaluate a measure like reliability with repair we allow repairs until a system failure state is reached. Since the reward is a specification of a system failure state at an higher level we can “stop” all transitions of the model when a specific value of the reward is reached.

5.2.3 MODEL OF TRANSITIONS AND TRANSITION FUNCTIONS

In ATS *a transition represents a set of events* whose occurrence makes the system move from a state to another. Each event of a transition is referred as a *mode of a transition*. For each mode is specified a *type* and a set of *attributes*. Possible types are: stochastic (*stoc*), deterministic (*det*) and instantaneous (*inst*). The difference between these classes lays in the different underlying process that defines the *time to occur of an event*. Stochastic means that the time to occur is defined via a probability density function, deterministic that an event takes a fixed amount of time to occur with probability one, and instantaneous that the event occurs in zero time having precedence over the other types.

Attributes related to a mode are of three kinds: activation (*Act*), reactivation (*React*) and parameters (*Par*). The activation defines if an event may occur, the reactivation if an event must be activated again (i.e., sampled again in simulation terminology) and the parameter defines the parameters of the function specified by the type of the mode. Stochastic and deterministic types support all the three attributes *Act*, *React* and *Par*, while instantaneous types support only the activation attribute.

Modes of transitions are useful in situations where the type of function that determines the time to complete of transitions can change over time. In this case, having defined different modes of a transition, the activation associated to each mode can be used to define the way a transition may complete given a system configuration. For instance, think about a component with a constant failure rate that can fail instantaneously if some condition occurs. In this case one can model the failure of the component by a transition with two modes that stand for the natural failure mode of the component (the exponential one) and the one dependent on external factors (the instantaneous one). Moreover, competition between different modes are also allowed, i.e., more modes can be active simultaneously.

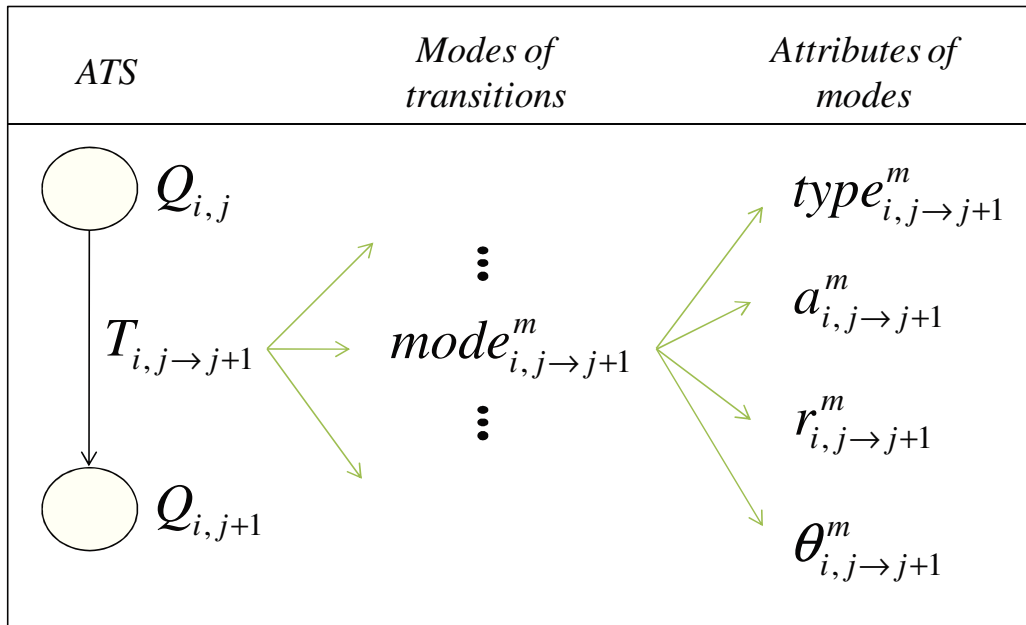


Fig. 5.3. Modes and attributes of transitions.

Let $T_{i,j \rightarrow j+1}$ be a transition and $mode_{i,j \rightarrow j+1}$ the set of possible modes. The m -th mode is a 4-tuple $mode_{i,j \rightarrow j+1}^m = (type_{i,j \rightarrow j+1}^m, \mathbf{a}_{i,j \rightarrow j+1}^m, \mathbf{r}_{i,j \rightarrow j+1}^m, \boldsymbol{\theta}_{i,j \rightarrow j+1}^m)$ where (Figure 5.3):

- $type_{i,j \rightarrow j+1}^m \in \{stoc, det, inst\}$ is a label that specifies the mathematical relation that defines the time to occur of the mode. “*stoc*” represents “stochastic” and can be further specified in terms of a specific probability density function, e.g., “*exp*” stands for exponential; “*det*” stands for deterministic, i.e., takes a fixed amount of time to occur, and “*inst*” for instantaneous, i.e., occurs as soon as it is enabled;
- $\mathbf{a}_{i,j \rightarrow j+1}^m$, is the activation function;
- $\mathbf{r}_{i,j \rightarrow j+1}^m$, is the reactivation function; and
- $\boldsymbol{\theta}_{i,j \rightarrow j+1}^m$, is the parameter function.

These three functions are related to the attributes of the mode. The activation function to the activation of a mode, the reactivation function to the reactivation of a mode and the parameter function to the mode parameters. To this end outputs of activation and reactivation functions are binary, while the outputs of parameter functions take on values in R .

Transition variables change over time. Since the system moves at point in time, transition variables are updated at every state change. In particular:

- The output of the *activation function* $\mathbf{a}_{i,j \rightarrow j+1}^m(n)$ is a binary variable that defines whether $mode_{i,j \rightarrow j+1}^m$ may complete, given the system configuration at the n -th time step. It can if the variable takes on value 1 and will not if the variable takes on value 0.
- Transition parameters can change during the system evolution too, i.e., they may be state dependent. The *parameter function* $\boldsymbol{\theta}_{i,j \rightarrow j+1}^m(n)$ defines the value of the parameter of $mode_{i,j \rightarrow j+1}^m$ at the n -th time step.
- In some cases it is necessary to *reactivate a mode of a transition*. For instance, any change of the transition parameter value will trigger

“reactivation”, a notion defined for SAN activities, which will trigger a generation of a new value for the duration of the particular transition mode using the current values of the mode parameter(s). The *reactivation function* $\mathbf{r}_{i,j \rightarrow j+1}^m(n)$ states if $mode_{i,j \rightarrow j+1}^m$ must be reactivated at the n -th time step. The reactivation is triggered when this variable takes on value 1, it is not when it takes on value 0.

The m -th mode of $T_{i,j \rightarrow j+1}$ is said *enabled* at the n -th time step if the system is in $Q_{i,j}$ and if the activation function $\mathbf{a}_{i,j \rightarrow j+1}^m(n)$ takes on value 1. We have:

$$\mathbf{en}_{i,j \rightarrow j+1}^m(n) = \begin{cases} 1, & \text{if } \mathbf{q}_i(n) = Q_{i,j} \wedge \mathbf{a}_{i,j \rightarrow j+1}^m(n) = 1 \\ 0, & \text{otherwise} \end{cases}. \quad (5.4)$$

The *time to occur*, $\mathbf{tto}_{i,j \rightarrow j+1}^m(n)$, of the m -th mode of $T_{i,j \rightarrow j+1}$ is given by:

$$\mathbf{tto}_{i,j \rightarrow j+1}^m(n) = \begin{cases} +\infty, & \text{if } \mathbf{en}_{i,j \rightarrow j+1}^m(n) = 0 \\ \text{type}_{i,j \rightarrow j+1}^m(\boldsymbol{\theta}_{i,j \rightarrow j+1}^m(n)) + \tau_n, & \text{if } \mathbf{en}_{i,j \rightarrow j+1}^m(n) = 1 \wedge \\ & \wedge (\mathbf{r}_{i,j \rightarrow j+1}^m(n) = 1 \vee \mathbf{tto}_{i,j \rightarrow j+1}^m(n-1) = +\infty) \\ \mathbf{tto}_{i,j \rightarrow j+1}^m(n-1), & \text{if } \mathbf{en}_{i,j \rightarrow j+1}^m(n) = 1 \wedge \\ & \wedge \mathbf{r}_{i,j \rightarrow j+1}^m(n) = 0 \wedge \mathbf{tto}_{i,j \rightarrow j+1}^m(n-1) < +\infty \end{cases} \quad (5.5)$$

Thus, the time to occur of a mode is set to: (i) infinite if the mode is not enabled; (ii) equal to the previous value if enabled at some previous time step and not reactivated at the current time step; and (iii) it is given by the defined type of the mode and its parameters if enabled at the current time or reactivated.

A transition is enabled if at least one of its modes is enabled. The time to complete of a transition can be derived from the time to occur of its events. Let $\mathbf{ttc}_{i,j \rightarrow j+1}(n)$ be the time to complete of $T_{i,j \rightarrow j+1}$. It is given by:

$$\mathbf{ttc}_{i,j \rightarrow j+1}(n) = \min_m (\mathbf{tto}_{i,j \rightarrow j+1}^m(n)). \quad (5.6)$$

Inputs of transition functions are system variables. We have that:

$$v_T(n) = f_T(v_S, v_R, n) \quad (5.7)$$

The specific form of f_T and its inputs are defined by the modeller. It is via the set of f_T that the modeler defines the dependencies between the elements of the system and the communication mechanisms between ATSS.

5.3 ORDERED ADAPTIVE TRANSITION SYSTEMS

Generally speaking transition and reward functions can be of any kind. In practice we see that most situation can be modelled considering only a subclass of possible relations.

We define a Ordered-ATS (OATS) an ATS where *transition timing variables* are related only by relational operators ($<, \leq, =, \geq, >$). In practice transition timing variables can be related by min, max operators and use the outputs of the latter as inputs of the relational operators ($<, \leq, =, \geq, >$). The use of relational operators allows to define a notion of state when a *base model* in the form of a Markov chain must be derived.

In order to make easier the construction and the readability of transition and reward functions we propose a graphical formalism similar to one used in fault trees. We model transition and reward variables as the *top node* of an acyclic graph whose *initial nodes* are system variables (and also reward variables in the model of transition functions) and intermediate nodes are *gates* that implement a specific function of their inputs.

Top nodes are evaluated bottom-up from the initial nodes through the various levels of the hierarchy defined by the tree. A top node can be connected to a single gate (the top gate) or to a single initial node and can take on the values of the outputs associated with these nodes.

Intermediate nodes can be any function of their inputs. Moreover, weights can be associated to gates and the links that connect nodes of the tree. Therefore, parameter of a function implemented by a gate are the weight of the gate and those of the input links, while input variables take on the values of the input nodes.

The structure of the graph is similar to a FT and since we use it to model transition and reward functions we call it *functional tree (f-T)*.

Definition 5.2 Formally, a *f-T* is a 5-tuple (I, G, T, C, W) , where:

- I , is the set of initial nodes;
- G , is the set of intermediate nodes (or gates);
- T , is the top node;
- C , is the set of connections between I , G , and T ; and
- W is the set of weights associated to G and C .

Gates implement a function of their inputs. For a given $G_i \in G$, we have that $g_i = g_i(w_i, W_i, I_i)$ where $w_i \in W$ is the weight associated to the gate and $W_i \in W$ are the weights associated with the inputs $I_i \in I \times G$.

In OATS we the following specification for g_i are essential for transition timing variables to be used:

- LESS, ($<$), is a functions with general inputs (in R) whose output can take on vale 1 if the first input is less that all the others and 0 otherwise.
- MORE, ($>$), is a function with general inputs (in R) whose output can take on vale 1 if the first input is greater that all the others and 0 otherwise.
- EQ, ($=$), is a function with general inputs (in R) whose output can take on vale 1 if all its inputs are equal and 0 otherwise.

Moreover we can have the combinations \leq and \geq . Other functions are common mathematical, algebraic and Boolean operators. In some case, a function can have a more complex structure. We have found that in modelling parameter functions the following gates can be used:

- \prod , or scaling gate. The gate is assigned a weight θ_{SL} . Also input links are assigned weights α_j , namely the scaling factors. The name derives from the fact that α_j scales the value of θ_{SL} if the j -the input node takes on value 1 (when inputs are binary).

$$y_{\Pi} = \theta_{SL} \cdot \prod_j \alpha_j x_j. \quad (5.8)$$

- Σ , or sum-progressive gate. Input links are assigned weights α_j . The name derive from the fact that α_j sums if the j -the input node takes on value 1 (when inputs are binary).

$$y_{PR} = \sum_j \alpha_j \cdot x_j. \quad (5.9)$$

Generally the structure of a f -T of an OATS implements an “extended” Boolean function of its inputs. In facts, all inputs are binary except for transition timing functions that however *are* related by (and, in OATS, must be) relational operators ($<, \leq, =, \geq, >$) seen above. In this case all the considered variables in the f -T become binary.

Weights, however, can restore the presence of non binary variables, but, do not change the notion of the underlying information considered by the set of system variables. This, as we will show, allows to define a notion of state of the system in a *base model*, i.e., state-space model, only in terms of system variables.

5.4 THE HEAT-POWER SYSTEM

In this section we present a contrived example to show how to build an ATS model of a system. Let consider a system made up of the following elements: three heat pumps P0, P1 and P2 and a generator G. P1 is in share-loading with P2 and both are in cold stand-by with respect to P0. G provides power to the pumps and a failure causes the instantaneous failure of all the pumps. Let assume the components are not repairable and can fail with a constant failure rate. The system failure is: “no heat is provided to the service”.

Let S_0, S_1, S_2, S_3 denote the four ATS models of P0, P1, P2 and G, respectively. Each of them is made up of two states representing the working and the failed condition. Let $Q_{i,0}$ and $Q_{i,1}$ denote the working and failed state ($i = 0,1,2,3$). $T_{i,0 \rightarrow 1}$ is the transition that connects $Q_{i,0}$ to $Q_{i,1}$.

Transitions $T_{i,0 \rightarrow 1}$ (0,1,2) represent two kind of events: the failure of P0, P1 and P2, respectively, due to internal faults or the failure due to the failure of the generator G. Thus, we assign two modes to these transition in order to represent both the possible events that can cause the state change. Let $mode_{i,0 \rightarrow 1}^m$ ($m = 1,2$) be these modes. Each mode is represented by a 4-tuple $(type_{i,0 \rightarrow 1}^m, a_{i,0 \rightarrow 1}^m, r_{i,0 \rightarrow 1}^m, \theta_{i,0 \rightarrow 1}^m)$. $type_{i,0 \rightarrow 1}^m$ is “exponential” for $m = 1$ and “instantaneous” for $m = 2$. The following activation functions can be considered for P0:

$$a_{0,0 \rightarrow 1}^1 = s_{3,0} \text{ and } a_{0,0 \rightarrow 1}^2 = s_{3,1},$$

meaning that that the exponential mode is active ($a_{0,0 \rightarrow 1}^1$) if the generator is in the working state ($s_{3,0}$) and that the instantaneous mode is active ($a_{0,0 \rightarrow 1}^2$) only if the generator is the failed state ($s_{3,1}$). Reactivation and parameter are fixed for both modes. We assign:

$$r_{0,0 \rightarrow 1}^1 = r_{0,0 \rightarrow 1}^2 = 0, \theta_{0,0 \rightarrow 1}^1 = \lambda_p \text{ and } \theta_{0,0 \rightarrow 1}^2 = 0.$$

Figure 5.4 shows the ATS model of P0.

State transition diagram	Transition modes	Type	Activation	Reactivation	Parameter
	$mode_{0,0 \rightarrow 1}^1$	exp		$r_{0,0 \rightarrow 1} = 0$	$\theta_{0,0 \rightarrow 1} = \lambda_p$
	$mode_{0,0 \rightarrow 1}^2$	inst		$r_{0,0 \rightarrow 1} = 0$	$\theta_{0,0 \rightarrow 1} = 0$

Fig. 5.4. ATS model of P0.

P1 and P2 are cold stand-by of P0. Thus, they can fail due to internal faults (“exponential” mode) only if P0 has failed ($s_{0,1}$). In this case activation functions are defined as:

$$a_{i,0 \rightarrow 1}^1 = \text{AND}(s_{3,0}, s_{0,1}) \text{ and } a_{i,0 \rightarrow 1}^2 = s_{3,1} \quad (i = 1,2).$$

meaning that that: (i) the exponential mode is active ($\mathbf{a}_{i,0 \rightarrow 1}^1$) if the generator is in the working state ($\mathbf{s}_{3,0}$) and the pump P0 is in the failed state ($\mathbf{s}_{0,1}$); and that (ii) the instantaneous mode is active ($\mathbf{a}_{i,0 \rightarrow 1}^2$) only if the generator is the failed state ($\mathbf{s}_{3,1}$).

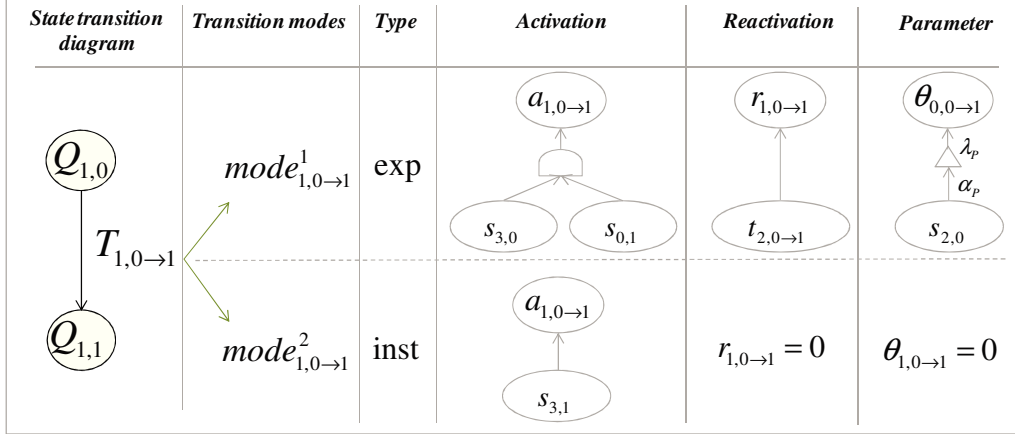


Fig. 5.5.ATS model of P1.

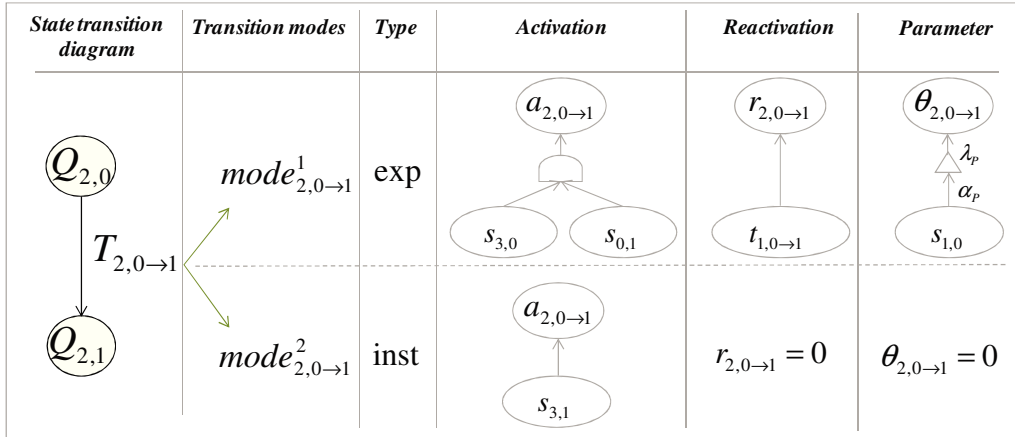


Fig. 5.6.ATS model of P2.

Since, P1 and P2 are in load sharing, parameter of the “exponential” mode are dependent on their reciprocal state. In particular let say that their parameter λ_p is scaled by $\alpha_p < 1$ if they both work. Thus, we can define the parameter functions as:

$$\theta_{i,0 \rightarrow 1}^1 = \text{SL}(\lambda_p, \alpha_p, \mathbf{s}_{j,0}) \text{ and } \theta_{i,0 \rightarrow 1}^2 = 0 \quad (i, j = 1, 2; i \neq j).$$

To be implemented, a change in the parameter needs reactivation. The following reactivation functions can be defined:

$$\mathbf{r}_{i,0 \rightarrow 1}^1 = t_{j,0 \rightarrow 1} \text{ and } \mathbf{r}_{i,0 \rightarrow 1}^2 = 0 \quad (i, j = 1, 2; i \neq j),$$

where we have taken advantage of the impulse nature of transition indicator functions. In particular the reactivation function of the exponential mode of the “failure transition” of P1 ($r_{1,0 \rightarrow 1}^1$) takes on value 1 if P2 was the component that most recently failed ($t_{2,0 \rightarrow 1}$). Figure 5.5 and 5.6 show the ATS model of P1 and P2.

Finally, the model of the generator G (Figure 5.7) is defined by the following transition functions:

$$a_{3,0 \rightarrow 1}^1 = 1, r_{3,0 \rightarrow 1}^1 = 0 \text{ and } \theta_{3,0 \rightarrow 1}^1 = \lambda_G,$$

where the transition $T_{3,0 \rightarrow 1}$ is assigned only one mode of occurrence (exponential).

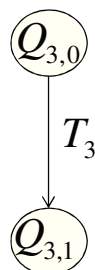
<i>State transition diagram</i>	<i>Transition modes</i>	<i>Type</i>	<i>Activation</i>	<i>Reactivation</i>	<i>Parameter</i>
	\rightarrow $mode_{3,0 \rightarrow 1}^1$	exp	$a_{3,0 \rightarrow 1} = 1$	$r_{3,0 \rightarrow 1} = 0$	$\theta_{3,0 \rightarrow 1} = \lambda_G$

Fig. 5.7.ATS model of G.

The reward function can be expressed in terms of the failed state of the three pumps (according to the top event: “no heat provided to the service”) as:

$$v_R = \text{AND}(s_{j,1}) \Big|_{j=0,1,2}.$$

The choice of using only system functions associated with the pumps follows to the fact that the generator is involved indirectly in the demand satisfaction. Thus, in ATS one can analyze the system at different level of details, considering first the elements directly related to the system requirements.

5.5 EVOLUTION AND EXECUTION OF ATS

The kind of systems we model are discrete event dynamic systems [9,10,11], thus, they move from a state to another in points in time $\tau_0 \leq \tau_1 \leq \dots \leq \tau_n \leq \dots$ defined

over R^+ where τ_0 is the initial instant and τ_n for $n > 0$ is the instant of occurrence of the n -th event.

The next event that will occur (or the next transition that will complete) is chosen by a race condition between concurrent enabled transitions. Transition can have the same time to complete in the case they are instantaneous or fixed. On the other hand, since the point probability of a continuous random variable is zero, stochastic transitions can never complete at the same time.

When more transitions have the same time to complete a non-deterministic choice is made. In fact, in this case it is not defined which transition will trigger first, i.e., the process of completion of a transition. Moreover, it is not said that in the resulting state all the previous concurrent transitions will be still enabled. Concurrent transitions can exist into the same ATS and among ATSS. In practice, non-determinism is solved probabilistically assigning the same probability to each simultaneous transitions.

Basically, every time a transition completes, the new value of system, reward and transition functions with the consequent update of the time to complete of transitions (Figure 5.8) are evaluated.

Given the time to complete $\mathbf{ttc}_x(n)$ ($x = 1, 2, \dots, NT$) of $T_x \in T$ (with T_x the x -th transition of the set of all the transitions of the model T of cardinality NT), the choice of the transition that will complete is determined by selecting non-deterministically among all the transition with minimal time to complete.

Let $T'(n) = \{T_x' \in T \mid \forall T_x \in T, \mathbf{ttc}_x'(n) \leq \mathbf{ttc}_x(n)\}$ denote the set of transitions with minimal time to complete. We non-deterministically select T_x^* from the set $T'(n)$. Obviously, given (5.8), it follows that that $\tau_{n+1} = \mathbf{ttc}_x'(n)$.

In practice, the choice is effectuated randomly over the set of the transitions with minimum time to complete. If we denote with $\mathbf{X}(n)$ the random variable that can take on $\#T'(n)$ possible values equally probable, with total probability mass 1, we say that $\mathbf{X}(n) = X_{i,j \rightarrow j+1}$ if $T_{i,j \rightarrow j+1}$ is randomly selected from the set $T'(n)$. Thus, we define the *choice* function as:

$$c_{i,j \rightarrow j+1}(n) = \begin{cases} 1, & \text{if } X(n) = X_{i,j \rightarrow j+1}, \forall i, j: T_{i,j \rightarrow j+1} \in T. \\ 0, & \text{otherwise} \end{cases} \quad (5.10)$$

Given $c_{i,j \rightarrow j+1}(n) = 1$ the state variable of the i -th ATS at time step τ_{n+1} , $\mathbf{q}_i(n+1)$, is updated following:

$$q_i(n+1) = \begin{cases} Q_{i,j+1}, & \text{if } c_{i,j \rightarrow j+1}(n) = 1, \forall i, j: T_{i,j \rightarrow j+1} \in T. \\ q_i(n), & \text{otherwise} \end{cases} \quad (5.11)$$

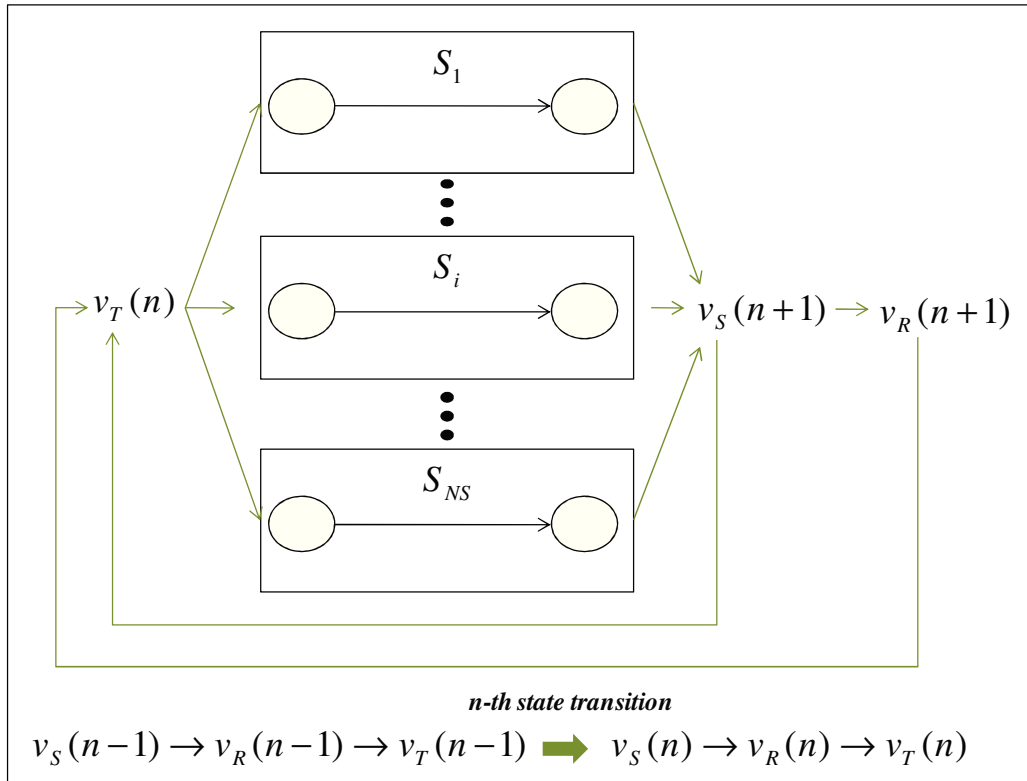


Fig. 5.8. Evolution of ATS.

The stochastic process underlying an ATS is defined by $Z_n = (\mathbf{q}(n), \tau_n)$, where $\mathbf{q}(n) \in Q$ is the random variable that define the state at the n -th step and $\tau_n \in \mathcal{R}$ defined the time at which the last state-transition occurred. Since all the variables defined above (system, reward and transition) can be defined in terms of Z_n , all variables are random variables. Moreover, since reward and transition variables are a function of system variables, only system variables can be considered in the definition of the state for the construction of a base state-space model of ATS. In the next section we show how to generate a Markov base model of OATS. OATS allows

to generate a finite state space because timing variables can be converted in ranking variables, i.e., rank of the order of completion of transitions.

5.6 MARKOV ORDERED ADAPTIVE TRANSITION SYSTEMS

In this section we show how is possible to generate a Markov model given an OATS. An OATS is said a Markov-OATS (MOATS) if modes of transitions are: (i) instantaneous or exponential; (ii) if there is not infinite cycles of instantaneous transitions; and (iii) if the first state is stable, i.e., there are not instantaneous transitions enabled at τ_0 .

We use a construction procedure that resemble the common algorithm used in Petri nets to generate the reachability graph of the net where *states of the base model* are defined as the marking of the net and *transitions of the base model* are the enabled transitions in each marking. In our approach a state of the base model will be derived in terms of the system variables of the ATS model while transitions of the base model will be derived from the enabled transitions given an ATS configuration, i.e., a given value of system variables.

Before to give the notion of state of the base model of an OATS, let discuss the construction procedure. We start considering the first state and checking the enabled transitions, their kind and parameters. A transition is enabled if there is at least an enabled mode. If there are more enabled modes, we have the following situation:

- if there is at least one instantaneous mode enabled, the transition is considered instantaneous; and
- if there are only exponential modes enabled, the transition is considered exponential with rate equal to the sum of the parameters of the enabled modes.

At the level of transitions enabled in a state, if there are instantaneous transitions, exponential transitions are discarded, because of the priority of instantaneous

transitions, and a weight, equal to the inverse of the total number of enabled instantaneous transitions, is assigned to each instantaneous transition.

Enabled transitions in a state define the set of the reachable states. These states can be states that have been already encountered during the construction procedure (states are compared on the basis of the values of their internal (system) variables). If a state is an “old” state a connection in the base model is defined between the old state and the one from where the considered transition departs. If it is a “new” state, it is included in the set of states “to be explored”, i.e., verify the enabled transitions and define the reachable states.

Obviously, for each connection between states is defined the type, exponential or instantaneous, and a weight for instantaneous transitions or a rate for exponential.

When this procedure is completed, in order to generate the equations associated with a Markov chain, states with outgoing instantaneous transitions (unstable states) must be eliminated from the model. We do this, by multiplication of the weights of instantaneous transitions to the rates of exponential transitions ending in the state where instantaneous transitions depart from.

The definition of the state must allow the evaluation of the enabling conditions of transition, the value of the parameters of transitions and the values of reward variables. In the following we give our definition of state in two cases: models with and without transition timing variables.

5.6.1 MOATS WITHOUT TRANSITION TIMING VARIABLES

In the case of MOATS without transition timing functions the state of the system is defined considering only indicator functions. In particular we define a state as the vector whose entries are the indicator functions of the system. It is $M = (\mathbf{s}_{i,j}, \mathbf{t}_{i,k})$ $\forall i, j, k: Q_{i,j} \in Q, T_{i,k} \in T$.

However, in the case *transition indicator variables are input only of reactivation functions*, the definition of a state is reduced to $M = (\mathbf{s}_{i,j})$. This fact is related to the memory-less property of exponential distributions.

We show the construction procedure of the Markov chain of the Heat-Power system presented in Section 5.4. For the sake of simplicity we have chosen to represent a state by ordered labels of the names of components (for instance with “P0” we represent the working state of P0 and with “-” the failed one). Moreover we do not consider transition indicator functions for the reasons stated above.

In Figure 5.9.a M_0 represents the initial state where all components are in the working state. Then we consider the enabled transitions in M_0 . Let $en(M_n)$ be the set of enabled transitions:

$$en(M_n) = \{T_{i,j \rightarrow j+1} \in T: \mathbf{s}_{i,j}(n) \cdot \sum_m \mathbf{a}_{i,j \rightarrow j+1}^m(n) > 1\}, \quad (5.12)$$

where $\mathbf{s}_{i,j}$, the state indicator variables, are the entries of M_n and the only inputs of $\mathbf{a}_{i,j \rightarrow j+1}^m$.

For the system in figure $en(M_0) = \{T_{0,0 \rightarrow 1}, T_{3,0 \rightarrow 1}\}$, i.e., only P0 and G can fail. The nature of transitions (*inst*, *exp*) is determined considering the type of the enabled modes. In this case the enabled modes of both transitions are exponential, thus transitions in the base model are exponential.

In general, given $en(M_n)$ we must check if the reachable states are new or have been already discovered. Two states are identical if $M_i = M_j$. Given M_0 two new states M_1 and M_2 are found. The process is then repeated until all states are discovered.

If there are enabled instantaneous transitions, enabled exponential transitions are discarded. In the Heat-Power system example there are not simultaneously enabled instantaneous and exponential transitions.

If there are more enabled instantaneous transitions, for each of them is assigned a weight. For instance, in M_2 there are three enabled instantaneous transitions, i.e., P0, P1 and P2 can fail instantaneously. In this case transitions are assigned a weight equal to the inverse of the sum of the concurrent enabled instantaneous transitions (in this case 1/3).

This procedure leads to the construction of an Extended reachability graph where stable and unstable states are present. The procedure can be summarized as follows:

- define the initial state M_0 ;
- evaluate the value of the reward variables in M_0 ;
- check the enabled transitions in M_0 ;
- check the kind of the enabled transition in M_0 and assign a parameter (a rate if exponential and a weight if instantaneous);
- evaluate the reachable states given the enabled transition in M_0 and add those states to the set “to be explored”.

The procedure continues exploring each state in the set “to be explored” as it was done for M_0 . In this case, however, not all the reachable states belong to the set to be discovered, i.e., some of them are “old” states.

Once that the above procedure has been completed (Figure 5.9.a) we must eliminate unstable states from the model. We start considering those unstable states that are reached by an exponential transition. These states are then eliminated from the model and transitions replaced by transitions connecting the preceding and successive states with rate equal to the product of the weight of the instantaneous transition and the rate of the exponential one (Figure 5.9.b).

From Figure 5.9.a we see that unstable states are $M_2, M_5, M_6, M_7, M_9, M_{10}, M_{11}$. M_2 is the first unstable state that must be eliminated since it is the only one where the incoming transitions are only exponential. This is done connecting directly M_0 to M_5 , M_6 and M_7 . The rates of these transitions are $\frac{\lambda_G}{3}$ where λ_G is the failure rate of the generator G . In some cases an unstable state may have more incoming exponential transitions, e.g., M_5 has two incoming exponential transitions departing from M_0 and M_1 . In these case one operates in the same way considering a transition per time and eliminating the unstable state once that all transitions have been merged.

Finally, if there are more transitions connecting two states, they are aggregated considering the sum of their rates. In Figure 5.9.c is shown the resulting state-space,

where we have highlighted system failure state as circles in bold red. Failure states, are retrieved evaluating the reward function in a each reachable state. Lumping techniques can be further applied in order to reduce the state-space.

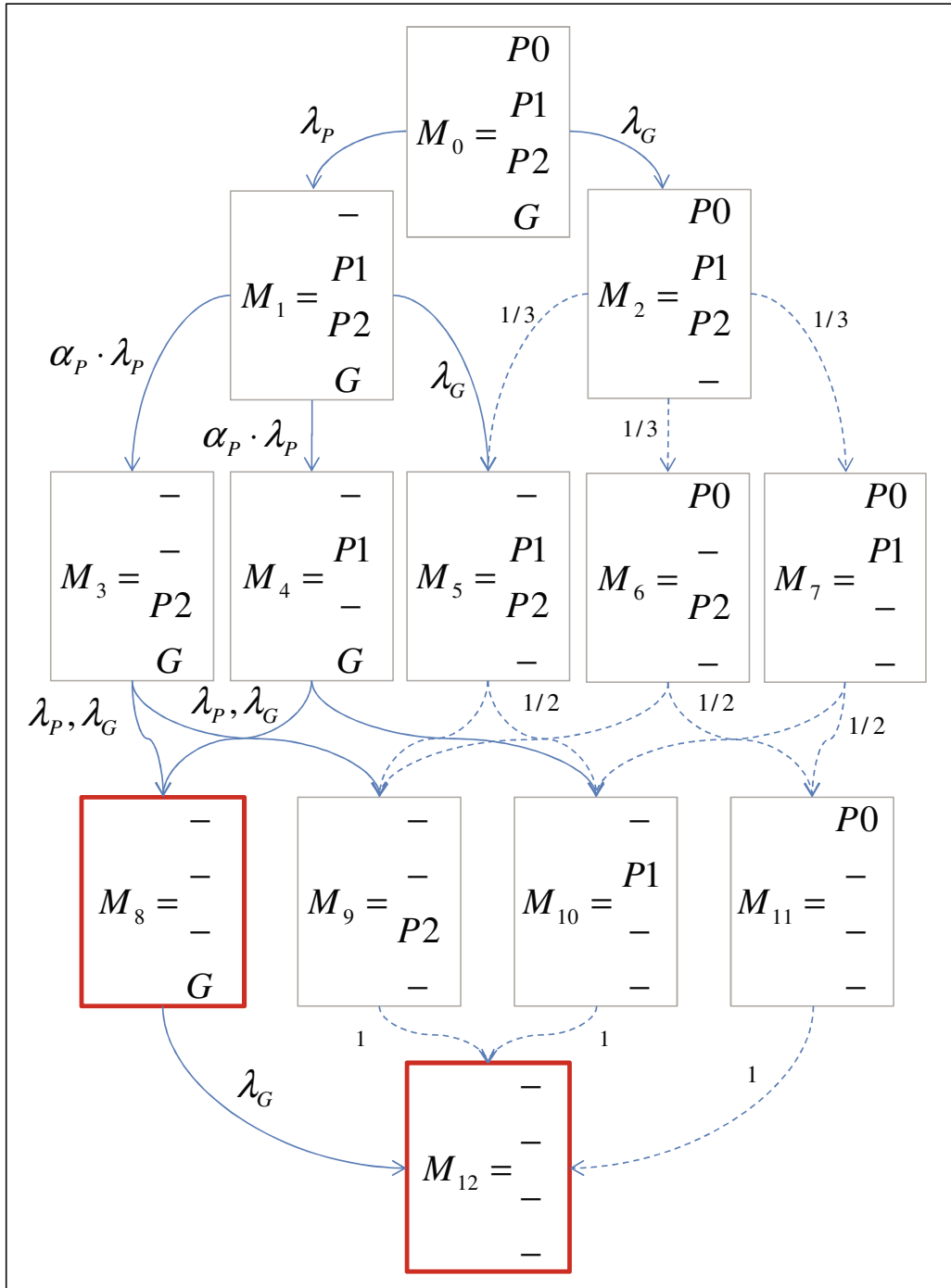


Fig. 5.9.a. Extended Reachability Graph of the Heat-Power system of Section 5.4.

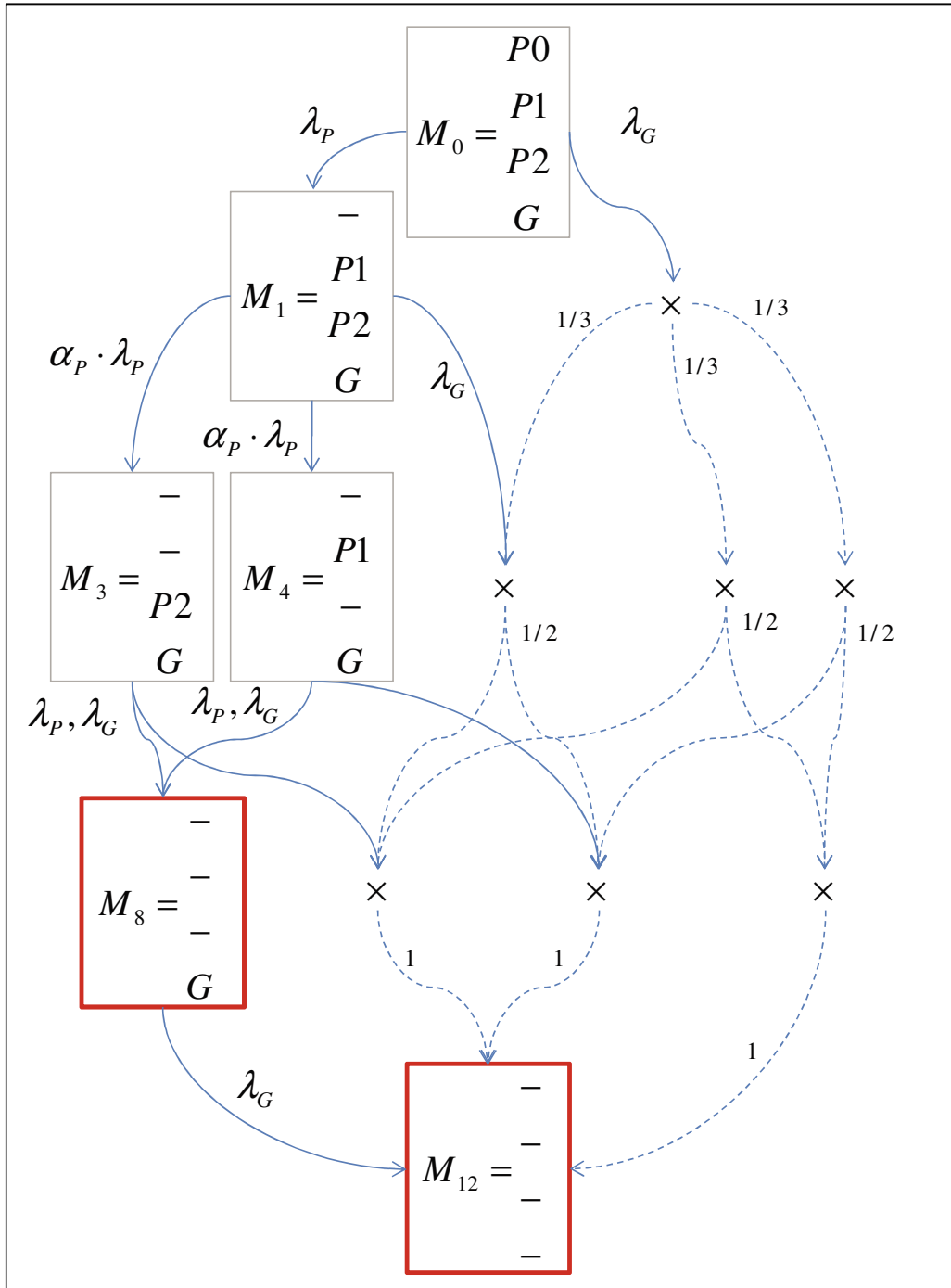


Fig. 5.9.b. Reduction process of the Extended Reachability Graph of Figure 5.9.a.

Reward functions that contribute to the evaluation of a performance property are evaluated and assigned to each state of the generated Markov chain. In this case the model is a Markov Reward model [20].

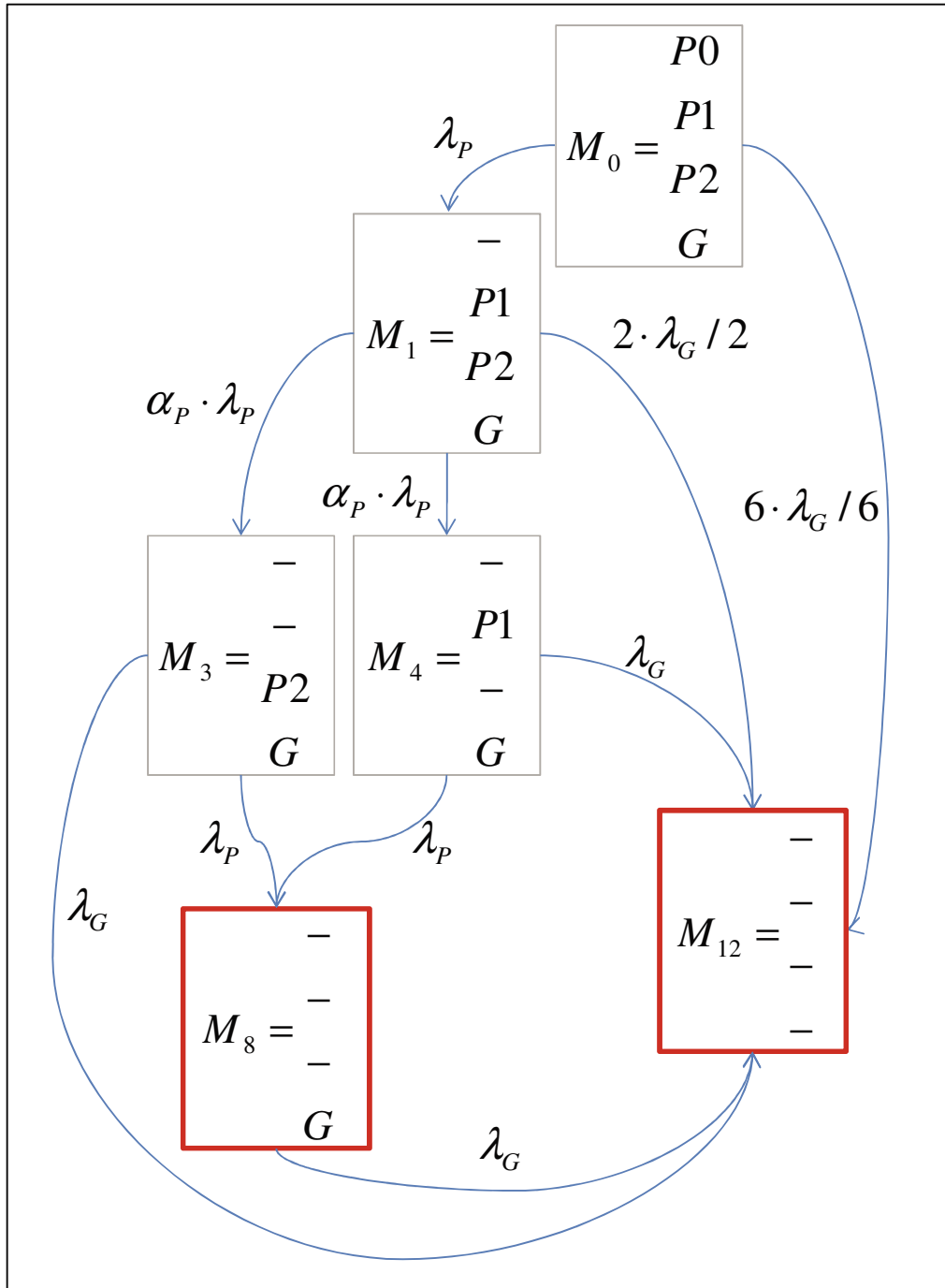


Fig. 5.9.c. Markov Chain of the ATS model presented in Section 5.4.

In the case of the Heat-Power system as well as for all the case where a measure of interest is the reliability or availability of the system, reward variables are binary. In fact, for these kind of measures we are interested in the probability of being in a

particular set of states of the system. Thus, a value of one will be assigned to those generated states of interest and 0 otherwise. In this case, solved the differential equations associated with the Markov chain, we can sum all the probabilities associated with those states where the reward takes on value 1.

5.6.2 MOATS WITH TRANSITION TIMING FUNCTIONS

In the case of MOATS models with timing functions the definition of the state of the base model must include the ordering of completion of transitions. To this end, we introduce a ranking function that associate a number (a rank) between 1 and the total number of transitions, NT , to each transition.

Since transitions timing variables are initialized to 0 at the initial time, we set the initial rank of all transitions equal to 1, $\mathbf{rank}_{i,k}(0) = 1 \ \forall T_{i,k} \in T$.

Let $T_{i,k}' \in T$ be the transition that completes at τ_n and let $R(n) = \{T_{i,k}^* \in T \mid \forall T_{i,k} \in T \exists T_{x,y}: \mathbf{rank}_{i,k}^*(n) = \mathbf{rank}_{x,y}(n)\}$ be the set of transitions whose ranking is not unique. For instance, at τ_0 all transitions belong to the set $R(0)$. We set the new rank of $T_{i,k}'$ to 1, $\mathbf{rank}_{i,k}'(n+1) = 1$. Remaining ranks at τ_{n+1} can be evaluated considering the kind (“*inst*”, “*exp*”) of $T_{i,k}'$. In particular:

if $T_{i,k}'$ is instantaneous we have the following cases ($\forall T_{x,y} \in T: T_{x,y} \neq T_{i,k}'$):

- if $R(n) = \emptyset$ or $R(n) \neq \emptyset$ and $T_{i,k}' \notin R(n)$

$$\mathbf{rank}_{x,y}(n+1) = \begin{cases} \mathbf{rank}_{x,y}(n), & \text{if } \mathbf{rank}_{x,y}(n) < \mathbf{rank}_{i,k}'(n); \\ \mathbf{rank}_{x,y}(n) - 1, & \text{if } \mathbf{rank}_{x,y}(n) > \mathbf{rank}_{i,k}'(n); \end{cases} \quad (5.13)$$

- if $R(n) \neq \emptyset$ and $T_{i,k}' \in R(n)$

$$\mathbf{rank}_{x,y}(n+1) = \mathbf{rank}_{x,y}(n); \quad (5.14)$$

Figure 5.10 shows an example of possible evolution of the system between two states given a completing instantaneous transition and the conditions seen above. $R(n) = \emptyset$ stands for “*no repeated ranking*”. In figure, transition whose ranking is 4 is selected to complete and its ranking is set to 1 in the new state. All ranking values less than 4

are kept fixed, while ranking values higher than 4 are scaled by 1. If $R(n) \neq \emptyset$ and $T_{i,k}' \in R(n)$ the transition that completes belong to the set of “repeated ranking”. In this case we keep fixed all the ranking values, unless the one of the completing transition that is set to 1. Finally, if $R(n) \neq \emptyset$ and $T_{i,k}' \notin R(n)$ the new marking is defined as for the case $R(n) = \emptyset$.

$T_{i,k}': inst$	$R(n) = \emptyset$	$T_x' \in R(n)$	$T_{i,k}' \notin R(n) \neq \emptyset$
τ_n	3-2- 4 -1-5	3-3- 2 -2-1	4-2- 3 -2-1
	↓	↓	↓
τ_{n+1}	3-2-1-1-4	3-3-1-2-1	3-2-1-2-1

Fig. 5.10. Example of ranking values evolution as new states are discovered (*inst*).

If $T_{i,k}'$ is exponential we have the following cases ($\forall T_{x,y} \in T: T_{x,y} \neq T_{i,k}'$):

- if $R(n) = \emptyset$ or $R(n) \neq \emptyset$ and $T_{i,k}' \notin R(n)$

$$\mathbf{rank}_{x,y}(n+1) = \begin{cases} \mathbf{rank}_{x,y}(n) + 1, & \text{if } \mathbf{rank}_{x,y}(n) < \mathbf{rank}_{i,k}'(n). \\ \mathbf{rank}_{x,y}(n), & \text{if } \mathbf{rank}_{x,y}(n) > \mathbf{rank}_{i,k}'(n). \end{cases} \quad (5.15)$$

- if $R(n) \neq \emptyset$ and $T_{i,k}' \in R(n)$

$$\mathbf{rank}_{x,y}(n+1) = \mathbf{rank}_{x,y}(n) + 1; \quad (5.16)$$

Figure 5.11 shows an example of possible evolution of the system between two states given an exponential transition and the conditions seen above. If $R(n) = \emptyset$ the transition whose ranking is 4 is selected to complete and its ranking is set to 1 in the new state. All ranking values higher than 4 are kept fixed, while ranking values less than 4 are added 1. If $R(n) \neq \emptyset$ and $T_{i,k}' \in R(n)$ we add 1 to all ranking values, unless the one of the completing transition that is set to 1. Finally, if $R(n) \neq \emptyset$ and $T_{i,k}' \notin R(n)$ the new ranking is defined as for the case $R(n) = \emptyset$.

Given the ranking of transitions we define the state of the system by the following vector $M = (\mathbf{s}_{i,j}, \mathbf{t}_{i,k}, \mathbf{rank}_{i,k}) \forall i, j, k: Q_{i,j} \in Q, T_{i,k} \in T$, or in the case there are not

impulse reward and transition indicator variables are input only or reactivation functions as $M = (s_{i,j}, \mathbf{rank}_{i,k})$.

$T_{i,k}': \text{exp}$	$R(n) = \emptyset$	$T_x' \in R(n)$	$T_{i,k}' \notin R(n) \neq \emptyset$
τ_n	3-2- 4 -1-5	3-3- 2 -2-1	4-2- 3 -2-1
	↓	↓	↓
τ_{n+1}	4-3-1-2-5	4-4-1-3-2	4-3-1-3-2

Fig. 5.11. Example of ranking values evolution as new states are discovered (*exp*).

The procedure to build the state-space is identical to the case without transition timing functions. Transition functions and reward functions can be evaluated from the information contained in the state vector. In OATS transition timing functions can be inputs only of relational operators. In the case of ranking functions we use the inverse of these operators. Thus, min (max) operator becomes a max (min) operators and the relational operator “>” (“<”) becomes “<” (“>”).

Although this state notation allows to specify the underlying stochastic process in terms of a CTMC, the main drawback is that the resulting state-space is very large even for very small systems. In fact, with this definition of state, different succession of events define different states even if state indicator functions are the same. The advantage, however, is that the class of systems that can be modelled by a Markov chain is larger. In fact, differently from ATS models without transition timing functions, where the state space is simply given by the orthogonal product of the state space of ATSs (and where some state can be isolated), in an OATS the definition of a state of the base model include the specific succession of events that leads to that state.

To explain this fact let us consider an ATS model made up of two ATSs each with two states connected by two transitions. Let A and \bar{A} be the states of the first ATS and B and \bar{B} those of the second. Moreover let a and \bar{a} (b and \bar{b}) be the transitions from \bar{A} to A (\bar{B} to B) and A to \bar{A} (B to \bar{B}), respectively. Furthermore, let the activation functions of a and b be $act = \text{NAND}((\pi_{\bar{a}} > \pi_{\bar{b}}), (\pi_{\bar{b}} > \pi_a), \text{AND}(\bar{A}, \bar{B}))$.

act is retrieved considering the condition for the deactivation, that is: the system is in state (\bar{A}, \bar{B}) given that \bar{B} was entered before \bar{A} (i.e., component B has already failed when A fails) and given that A was entered before \bar{B} (e.g., component A was not repaired after the failure of B).

Figure 5.12 shows a reduced Markov chain of the system where we have considered only the ranking of those transitions whose timing functions define *act*. This procedure permits, indeed, to reduce the dimension of the state space. For the sake of clarity, in figure we show the relations defined in *act* in the definition of the state.

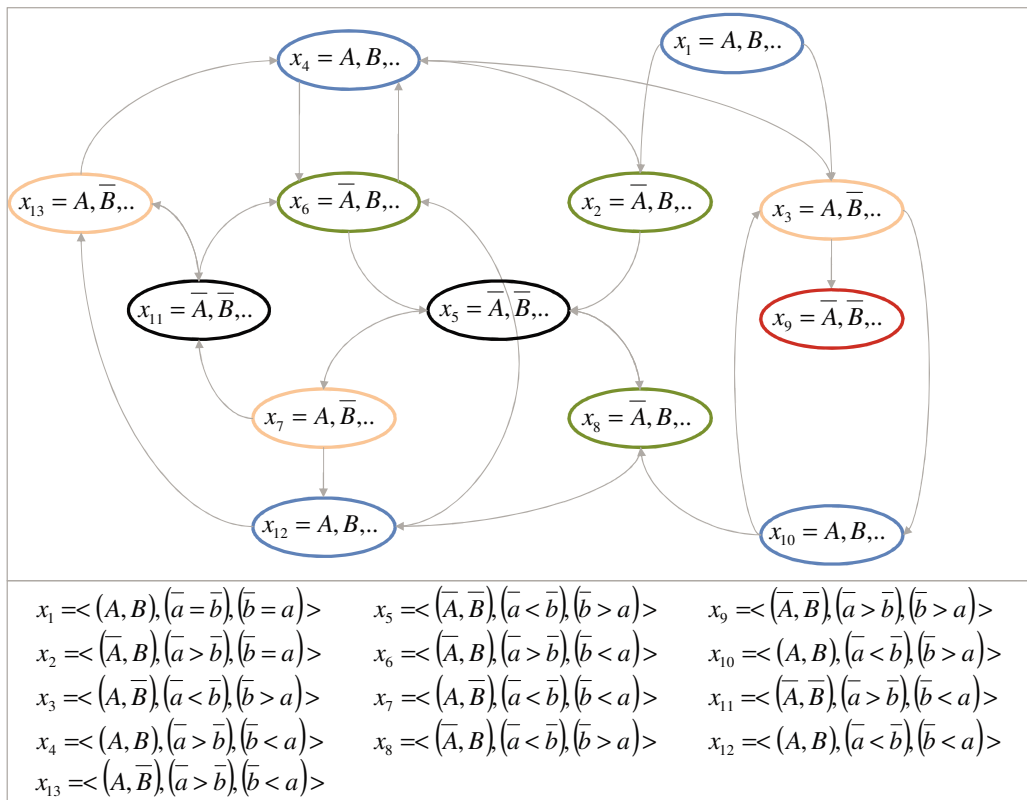


Fig. 5.12. Example of construction of a Markov chain of a MOATS with transition timing variables.

Figure 5.13 shows the lumped Markov chain. It is shown the state transformation and the resulting system of differential equations. The model reminds a PAND gate with two repairable components as inputs where it is not possible to exit the failed state (x_9). Moreover the failed state can be reached only if both components pass

from the nominal state to the failed state in the succession $AB \rightarrow A\bar{B} \rightarrow \bar{A}\bar{B}$, while the succession $\bar{A}\bar{B} \rightarrow A\bar{B} \rightarrow \bar{A}\bar{B}$ will lead to the safe state.

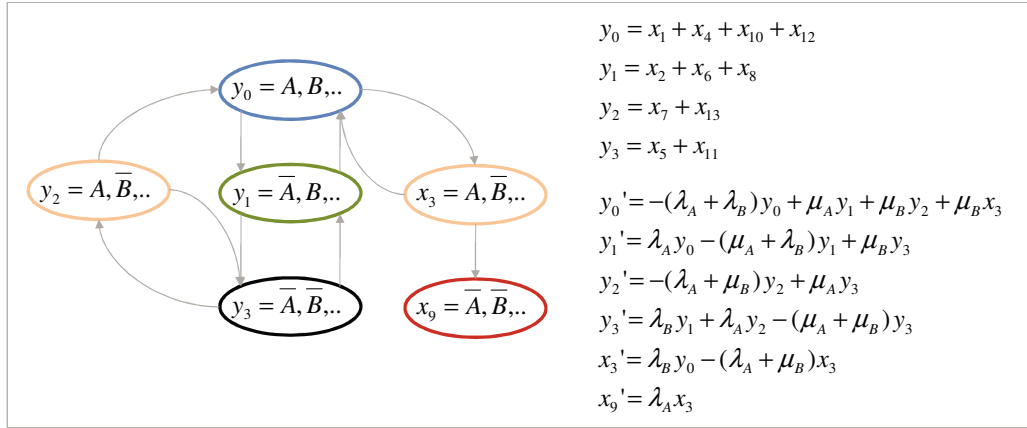


Fig. 5.13. Lumped Markov Chain and differential equations of the Markov Chain in Figure 5.12.

5.7 DISCRETE EVENT SIMULATION OF ATS

Simulation of ATS can be used to solve complex system with general distributions and for which the state-space results to big or analytical methods are not applicable. Simulation of an ATS consists of an execution of an ATS as described in Section 5.5. The algorithm that permits the simulation of ATSs can be defined as:

Until ending conditions met (e.g., #batches)

Initialize time to complete of transitions

Initialize system variables

Evaluate reward variables

$T_{prog} = 0$

While $T_{prog} < T_{max}$

Evaluate transition variables

Update time to complete

Choose the firing transition

Update system variables

Update reward variables

end

end

In order to make the algorithm execution faster, updating commands can be directed only to the subset of involved variables. Expected values of reward variables can be evaluated on the fly, as the simulation evolves, or analyzing the generated evolution traces. In bold are reported those commands that require user inputs.

As we can see the only initial information needed to simulate the model are the initial values of the time to occur of transitions, generally set to $+\infty$ and the initial value of the system variables. In general \mathbf{s} are defined in such a way that the initial occupied state in each ATS is defined and \mathbf{t} and $\boldsymbol{\pi}$ are set to 0. Thus only the vales of s need to be given as input of the model.

Other inputs of the model are the structure of ATS and the transition and reward functions. In particular, transition and reward functions are evaluated on the basis of the initial value of the system variables.

In the next subsection we present a case study of a system specified in terms of an ATS model that was resolved by simulation using a simulator created in Matlab®.

5.7.1 THE ROME POWER-TELCO NETWORK SYSTEM

The Rome Power-Telco system is a case study based on the Integrated Risk Reduction of Information-based Infrastructure Systems (IRIS) project (project co-funded by the European Commission within the Sixth Framework Programme 2002-2006) [17,18]. The objective of the study is to model and analyze interdependencies existing between two different critical infrastructure, a power and telecommunication networks. In this study we focus only on the dependencies directed from the power network to the Telco network. The context is that of the online risk estimator: given an initial state of the system, retrieve dependability measure about the probability of first failure of Telco node due to the lack of energy.

Elements of the systems are 117 power elements and 48 batteries. Power elements are of two kinds: nodes (cabins) and links (cables) between nodes.

5.7.1.1 ATS MODEL OF POWER NETWORK ELEMENTS

Power elements are represented in terms of two ATSs, one capturing aspects related to the working/failed dimension and the second the powered/unpowered dimension.

Let $E_i \in PN$ be the i -th element of the power network PN . For each element we build two ATSs. In particular we have:

- an ATS with two states W_i and F_i that denote the working and failed states, respectively. Transitions connecting these two states are exponential and always active. Parameters (and reactivation) depend on the states of parent nodes.
- An ATS with two states P_i and U_i that denote the powered and unpowered states, respectively. Transitions connecting these two states are instantaneous and the activation depends on the voltage associated with the element.

System variables associated with the two ATSs are:

- W_i and F_i , are the state indicator variables of W_i and F_i , respectively (the context specifies the meaning, i.e., states or state indicator variables).
- w_i and f_i , are the transition indicator variables of $T_{i,W \rightarrow F}$ and $T_{i,F \rightarrow W}$, respectively.
- P_i and U_i , are the state indicator variables of P_i and U_i , respectively (the context specifies the meaning, i.e., states or state indicator variables).
- p_i and u_i , are the transition indicator variables of $T_{i,P \rightarrow U}$ and $T_{i,U \rightarrow P}$, respectively.
- Transition timing functions are not defined.

Parameter and reactivation functions of transitions between $T_{i,W \rightarrow F}$ and $T_{i,F \rightarrow W}$ are subjected to stochastic association existing between elements.

Stochastic associations can be modelled in terms of parent and child elements. For an element E_i , let $P(E_i)$ denote the set of its parent elements. Parent elements influence the parameter of transitions by scaling the relative transition rate.

In particular the transition rate, λ_i (μ_i), of $T_{i,W \rightarrow F}$ ($T_{i,F \rightarrow W}$) is scaled by a factor α (β) n times depending on the number of unpowered parent elements. This is modelled by a f -T with top node representing the parameter of the transition and a gate Π with inputs the state indicator functions relative to the unpowered state of parent elements ($\forall j: U_j \in P(E_i)$).

We reactivate a transition when a parent element switches (in both directions) between the powered and unpowered states. In this case we specify this condition by an OR relation between the transition indicator functions p_j and u_j , where $U_j \in P(E_i)$.

The ATS model is shown in Figure 5.14. Dashed lines resemble the formalism used in Parametric Fault Trees [21].

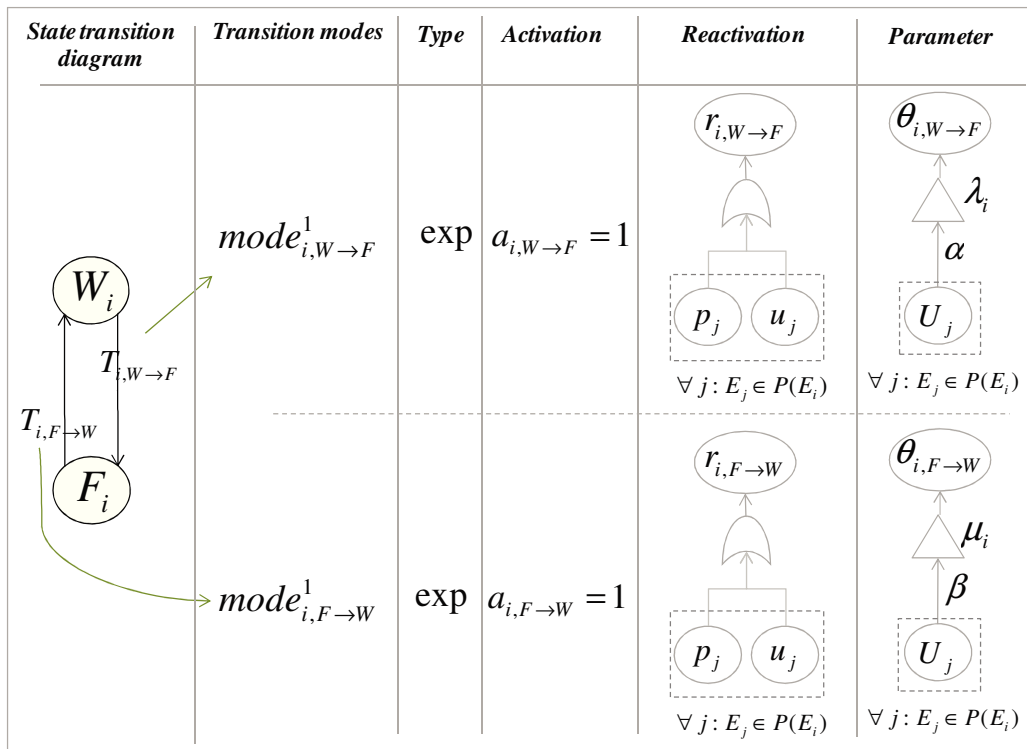


Fig. 5.14. ATS model of the working/failed dimension of power elements (E_i).

The ATS of the powered/unpowered dimension of power elements is shown in Figure 5.15. Transitions connecting these two states are instantaneous with activation depending on the value of the voltage associated with the element. In particular $T_{i,P \rightarrow U}$ is active if V_i is equal to 0 or greater of the threshold v_i and $T_{i,U \rightarrow P}$ is active if $0 < V_i \leq v_i$.

The voltage associated with a power element is function of the working/failed elements of the network and is given solving the balance equation of the network considering the remaining elements, i.e., elements in the working state. Thus the voltage of elements is a particular kind of reward function with inputs state indicator functions W_i .

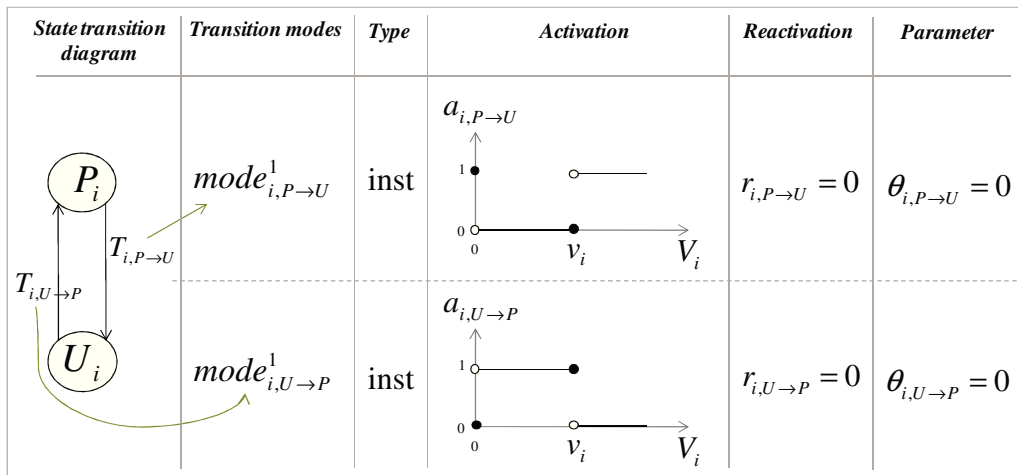


Fig. 5.15. ATS model of the powered/unpowered dimension of power elements (E_i).

Figure 5.16 shows three reward functions associated with the ATS model of the power network. The number of working and powered elements is retrieved using the gate Σ with inputs the working and powered states, respectively, considering an unitary weight for the input links. The voltage level associated with elements are retrieved as described above (gate K stands for Kirchhoff balance equations).

The system behaviour depends on the specific network configuration and the values of the parameters defined in the model. The following attributes impact on the system behaviour:

- physical connections between power elements have effect on the voltage associated with the nodes (Kirchhoff balance equations);

- stochastic associations have effect on the failure and repair rates of the elements. In this context both the connections defined by the parent child relationship and the values of the scaling factor α and β are of influence.
- Parameter like failure and repair rates as well as voltage thresholds of elements.

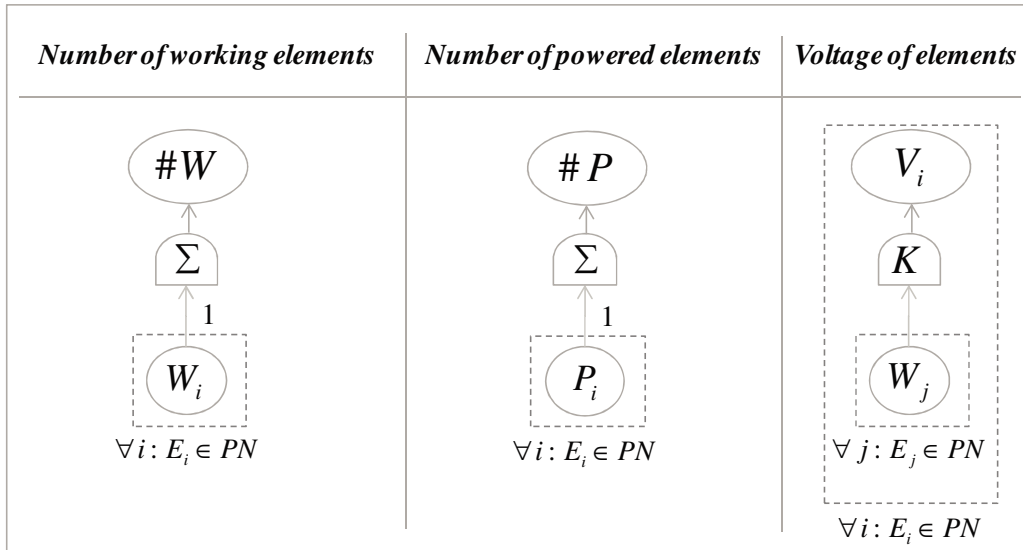


Fig. 5.16. Reward function of the ATS model of the power network.

Figure 5.17 and 5.18 show three possible evolutions of the power network in terms of number of working and powered components, respectively, with parameters $\alpha = 500$ and $\beta = 1$, with initial conditions “all working and powered”. The values of other parameters (failure and repair rates, voltage thresholds, parent nodes, physical connections) were defined in the IRIIS project [18]. Results were obtained through a simulative model created in Matlab®.

In order to speed up the simulation we consider a synchronization mechanism between instantaneous transitions of the kind:

- when two or more transitions are synchronized and their time to complete is identical and one of them is chosen to be triggered, they occur simultaneously, i.e., state change are updated simultaneously.

Synchronization allows to fire more than a transition in a single time step. In this way the variables of the systems are updated only after all synchronized transitions

are fired. The main advantage is to avoid loops of instantaneous activities that update the state of systems made of several components. On the other hand, synchronization can be used as a modelling mean in some scenario. In the present case study we synchronized all the transitions of the powered/unpowered ATs models.

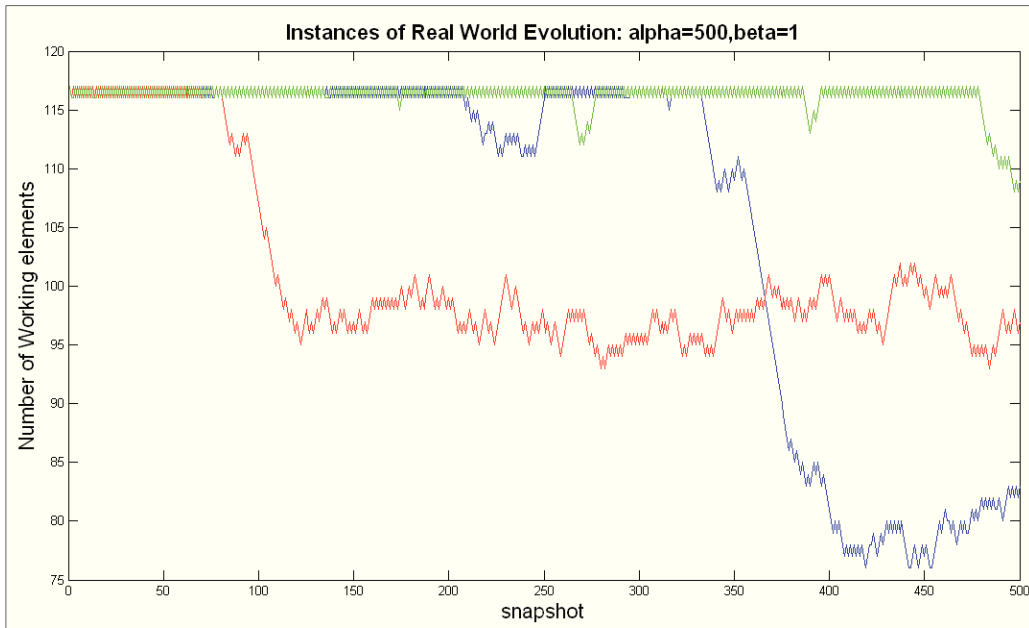


Fig. 5.17. Instance of evolution of the power network. Number of working elements.

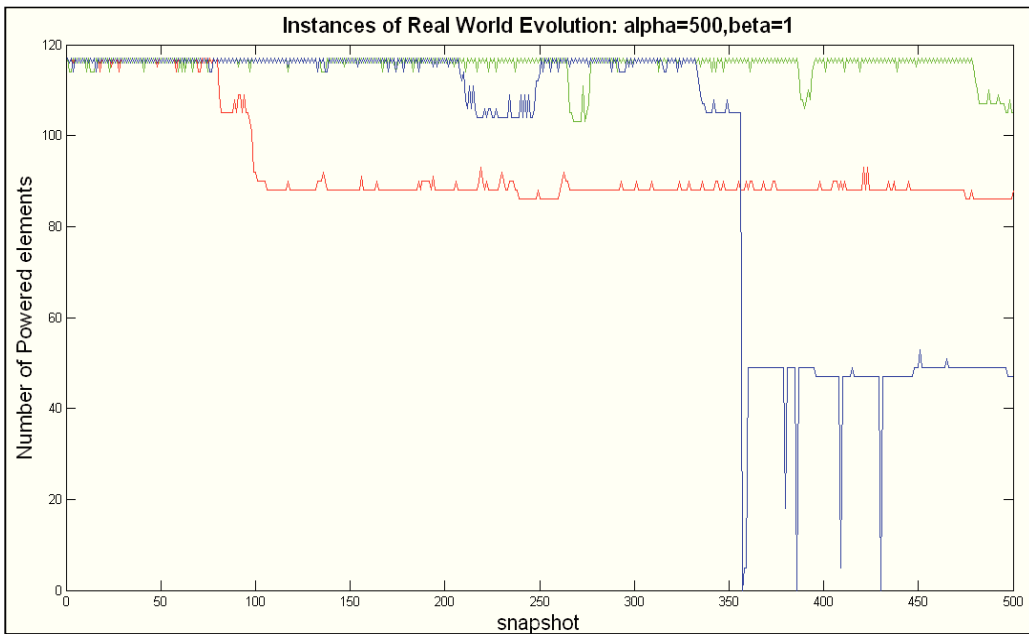


Fig. 5.18. Instance of evolution of the power network. Number of powered elements.

5.7.1.2 ATS MODEL OF BATTERIES

Batteries are used to supply energy when a Telco element lacks of electricity due to a interruption of service of the power network. A number of 48 batteries are used to supply 48 distinct Telco elements. Each of these Telco elements can receive electricity by one or more power elements. When all of these elements are unpowered, batteries switches to the active state. At the same way, when at least one of these elements are powered again, batteries switch back to the stand-by state. Since there is a battery for each Telco element that receive energy from the power network we can consider batteries connected directly to the network.

State transition diagram	Transition modes	Type	Activation	Reactivation	Parameter
	$mode_{k,C \rightarrow F}^1$	exp	 $\forall i: E_i \in C(B_k)$	$r_{k,C \rightarrow F} = 0$	$\theta_{k,C \rightarrow F} = \lambda_k$
	$mode_{k,F \rightarrow C}^1$	exp	 $\forall i: E_i \in C(B_k)$	$r_{k,F \rightarrow C} = 0$	$\theta_{k,F \rightarrow C} = \mu_k$
	$mode_{k,F \rightarrow C}^2$	inst	 $\forall i: E_i \in C(B_k)$	$r_{k,F \rightarrow C} = 0$	$\theta_{k,F \rightarrow C} = 0$

Fig. 5.19. ATS model of batteries (B_k).

Let $B_k \in B$ be the k -th element of the set of batteries B and let C_k and F_k denote the charged and flat state, respectively. Furthermore, let $\mathcal{C}(B_k)$ be the set of power elements that supply power to B_k (or, similarly, to the Telco node whose B_k is the battery).

Transition $T_{k,C \rightarrow F}$ is exponential and active if all the power elements of $\mathcal{C}(B_k)$ are unpowered. Transition $T_{k,F \rightarrow C}$ has two modes: an exponential mode that is active if all the power elements of $\mathcal{C}(B_k)$ are unpowered and instantaneous if at least one of the elements of $\mathcal{C}(B_k)$ is powered. The exponential mode model the time needed to change a flat battery, while the instantaneous mode is an approximation that reflects the fact that when the power is restored into the power network, the battery becomes charged instantaneously (approximation reasonable due to the small failure rates of power elements). The ATS model of batteries is shown in Figure 5.19.

5.7.1.3 THE ONLINE RISK ESTIMATOR

Defined the model of the real world, we simulate it and take snapshots of the values of its variables at different time steps. These values are used in the initialization of the risk estimator model. In particular beside the state of the modelled elements at the taken snapshot, inputs of the risk estimator are residual life of batteries given as the difference between the time to complete of transition and the value of the time step of the considered snapshot. Residual life times are used as initialization values of the time to complete of transitions.

The property of interest is the reliability of batteries, that is, in the case of repairable components the probability of first occurrence of the failure of batteries.

Thus the reward variable of which we want to estimate the expected value is $v_R = F_k$. However, in order to evaluate the probability of first occurrence we need to stop the evolution of the system as the system failure occurs. Thus in the Risk estimator model all transition have assigned an activation variable that is true only when the considered battery has not failed. However, in this way we need to simulate the model for each battery.

Another way (instead of blocking the system evolution) is to simulate it all at once and retrieve off line the quantities of interest trough trace analysis.

We evaluate the reliability of the batteries in relation with the size of the black out of the power network. Figure 5.10 shows the generated histograms for $t = 0, 1, \dots, 10$ and $\Delta t = 1$. In the second dimension four classes of black out sizes are defined: low 0-5, medium 6-25, high 26-50 and very high 51-117 (with 117 the total number of power elements). To retrieve these information we made use of the expected value of the reward function $\#P$. In figure there are 48 histograms, one for each battery.

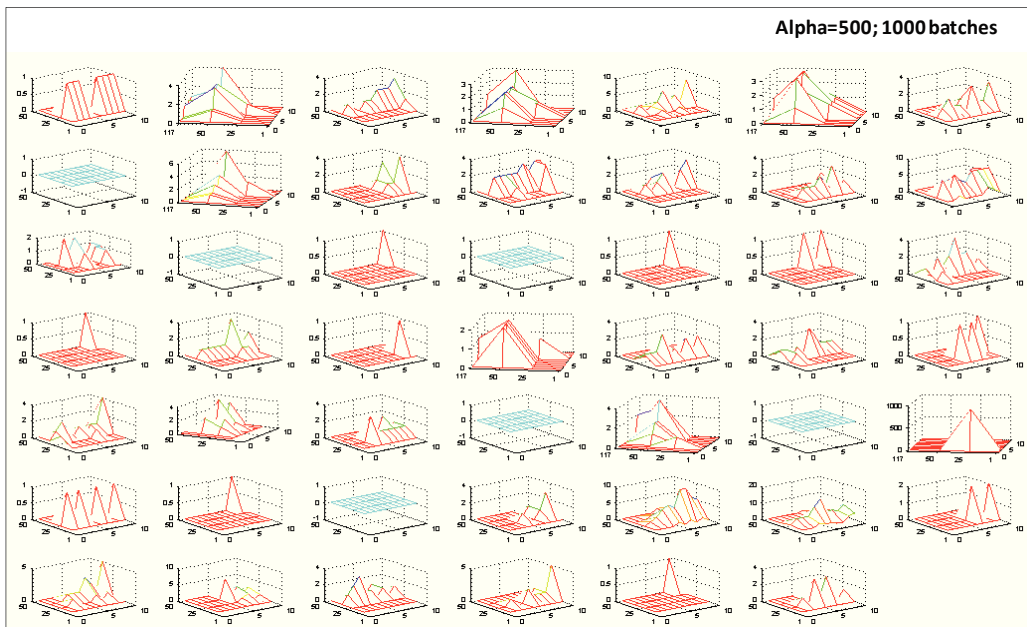


Fig. 5.20. Output of the risk estimator. Each histogram represent the reliability (pdf) at various times (x-axis) of a battery in relation with the impact size (y-axis).

5.8 CONCLUSION

In our work we tried to alleviate some typical problems emerging during modelling systems for dependability studies. In the dependability community the need for modelling temporal aspects in the definition of performance measures has been widely recognized.

Initially we have been concerned with limitations of Dynamic Fault Trees (DFTs) that arise when considering repairable components or complex behaviour due to substitution logics and load-sharing systems that cannot be tackled by the dynamic gates of the tree [23].

In this context related works are Dynamic Reliability Block Diagrams (DRBDs) and Boolean driven Markov Processes (BDMPs) [23,24]. They try to alleviate problems of DFTs by the definition of state machines related to the blocks of the diagram or to the leaf of a Fault Tree. However, in DRBD the class of finite state machines is limited to three states and the relation between state machines are defined by fixed sender/receiver relations. In ATS, the state machines can have an arbitrary number of states with an arbitrary meaning and relations can involve the occurrence of more events in given sequences. The same consideration are true for BDMP where, though give more classes of possible state machines, their structure is defined and regulated by input Boolean functions that will start a state machine instead of another. There are some similarities between BDMP and ATS, but ATS extends BDMP because can have a general reward structure, while the reward structure of BDMP is related to a FT with triggers of the activation of state machines. Thus, also the relations that can be modelled in an ATS are more general than the ones of BDMPs.

Another class of related works are extension to Petri nets [12] and especially Stochastic activity networks (SAN) [13]. Many of the objects defined in ATS are similar to those of SAN. In SAN for an activity is defined an activation and a reactivation. Parameter of activities can be marking dependent. Thus, modes of transitions in ATS are similar to activities in SAN. However, transitions in ATS are more general because the kind of distribution can vary too. In SAN this can be achieved considering a group of activities. SANs are also more general than ATS. Activities are assigned cases that probabilistically choose the next marking. In ATS probabilistic choice have not yet been defined. Moreover the structure of the network is more general of the structure of ATS. However, we believe that this is the major advantage that ATS have on SANs with respect to dependability studies: although, not general like SAN, they furnish a state space representation that can be used to

model many classes of systems. For instance, they can be applied in queuing systems with finite number of possible arrives.

Another class of related research is the one of model checking and related methodologies [6-10]. In particular the PRISM language presents the possible states of a state machine by a variable that can take on a finite number of values and transitions between states as functions of these variables. Each function is assigned a guard condition (or an activation) and a rate that define its time to complete in the case the underlying process is Markov. One of the advantage of ATS is that the parameter of transitions can be state dependent. Moreover, the introduction of transition timing variables facilitates the definition of temporal dependencies based on sequence of events. In PRISM this may be modelled introducing the ranking variables and function defined for ATS.

Another related methodology is that of interactive Markov chains (IMC) where a stochastic process algebra is used [2]. They use a completely different approach to define the communication between transition models: instantaneous transitions are labelled transitions that complete synchronously among sub models. In this way a communication mechanism is defined. Thus, they integrate interactive processes with Markov chains and define an algebraic semantic for the definition of the associated Markov chain. Central concept of IMC are synchronization and non determinism. In ATS we do not model synchronous transitions since transition system communicate by a set of defined variables that are updated at each state change. I/O IMC extends this concept with the introduction sender and receiver [22]. In our approach we treat non determinism with equi-probability.

As an example let consider stand-by component that can replace two different components belonging to two different subsystems. A subsystem is considered failed if there are not active components. That is: the primary component has failed; and the stand-by component has either failed or has been already used in the other subsystem. If one want to model the system by the mean of IMC, one approach would be to define a IMC for each component of the system as well as of the subsystems (which could be an indication of which component is active). If components are not repairable the resulting model can be found in [22]. A quick look

at the model and one can see that there are many transitions (both timed and synchronizing). The complexity of the modelling activity lays in the definition of these transition systems that can be very large in size. In ATS the same system can be modelled by a transition system for each component. The size of the ATS models is smaller compared to the one of IMC and one can abstract more easily from the models of interdependent parts.

Generalised Semi-Markov Processes (GSMPs), i.e. probabilistic timed systems where durations of delays are expressed by means of random variables with a general probability distribution, have been defined in terms of discrete event systems and discrete event simulation in [11] and in terms of stochastic process algebra in [4]. A GSMP describes the temporal behaviour of a system by using elements, which act similarly as clocks of a Timed Automata. In particular the temporal delays in the evolution of a system are represented by clocks (elements) whose duration is determined by an associated generally distributed random variable. In this way the temporal behaviour of the system is guided by the events of start and termination of clocks (elements). An ATS is a formal specification of GSMP where dependencies are modelled explicitly by the mean of a set of functions defined over the evolution of the system.

With ATS we have developed a formal method to model dependencies in interdependent systems effective for dependability studies. ATS generalize the class of dependencies and behaviours captured by an high level modelling formalism like FT, DFTs, DRBD, BDMP. Moreover ATS can be seen as an high level model specification of stochastic extension of Petri nets.

We consider the framework an important addition to the well known formalisms for stochastic modelling of complex systems because it offers a standardized way of building models of complex systems. We believe that standardization will alleviate the difficulties currently experienced by modellers to maintain each other's or even own complex stochastic models caused by the modellers' preferences in constructing models, which may be either unfamiliar to others or difficult to understand. We believe that promoting standardization and visualization in constructing models is also likely to improve model comprehension, make the model validation easier and

even make model maintenance easier: changes in most cases will be applied at an higher level of abstraction to the graphical representations of the transition attributes and various predicates. Hence, no intimate knowledge of the underlying implementation details (e.g. knowledge of the C++ classes used by the tool) is required.

Finally, improved usability of the tools, in our opinion, will make it much more likely for domain experts to engage more closely to scrutinize the models and, thus, improve their quality.

Exploiting the full potential of the framework will require tool support, more specifically for visually modelling ATSs and f -T and for automatic transformation of ATS to the chosen target modelling environment, e.g. Mobius, Matlab or any other tool with suitable functionality. These are concerns which we intend to address in our future work.

Other future developments concern the introduction of probabilistic choice, the formalization of synchronizing mechanism between concurrent transitions, application of ATS to model checking, i.e., by the definition of a temporal logic suitable for the formalism, and the definition of a process algebra semantic for ATS.

Finally we believe that multi-formalism models is one of the main achievements that must be reached in dependability modelling. The Mobius [14,26] tool for instance allows the composition of PEPA, SAN, ADVICE and FT models. SHARPE [27] allows multi-composition of Stochastic Reward Nets [25], Fault Trees, Markov chains, etc. In the next chapter we give a SAN representation of ATS. This conversion, beside the fact of allowing the resolution of ATS by a computer based tool, shows that ATS could be implemented as one of the available formalisms in the Mobius tool.

BIBLIOGRAPHY

- [1] Clark, A., Gilmeore, S., Hillston, J. & Tribastone, M., 2007. *Stochastic Process Algebras*. Lecture notes in computer science, Springer. Vol. 4486, pp. 132-179.

- [2] Hermanns, H., 2002. *Interactive Markov Chains*. Lecture notes in computer science, Springer. Vol. 2428.
- [3] Hoare, C.A.R., 1985. *Communicating Sequential Processes*. Prentice-Hall.
- [4] Bravetti, M. & Gorrieri, R., 2002. *The theory of interactive generalized semi Markov processes*. Theoretical computer science. Vol. 282(1), pp. 5-32
- [5] Strulo, B., 1993. *Process Algebra for Discrete Event Simulation*. PhD Thesis, Imperial College.
- [6] Kwiatkowska, M., Norman, G. & Parker, D., 2009. *PRISM: probabilistic model checking for performance and reliability analysis*. ACM Sigmetrics Performance evaluation review. Vol. 36(4), pp. 40-45.
- [7] Baier, C. & Kaoten, J.P., 2008. *Principles of Model Checking*. MIT Press.
- [8] Kwiatkowska, M., Norman, G. & Parker, D., 2002. *PRISM: probabilistic symbolic model checker*. Lecture notes in computer science, Springer. Vol. 2324, pp. 113-140.
- [9] Ballarini, P., Djafri, H., Duflot, M., Haddad, S. & Pekergin, N., 2011. *HASL: An Expressive Language for Statistical Verification of Stochastic Models*. In VALUETOOLS'11.
- [10] Younes, H.L.S. & Simmons, R.G., 2006. *Statistic probabilistic models checking with focus on time-bounded properties*. Information and computation. Vol. 204(9), pp. 1325-1345.
- [11] Glynn, P. W., 1983. *On the role of generalized semi-Markov processes in simulation output analysis*. In Proceedings of the 15th conference on Winter simulation. Vol. 1, pp. 38-42.
- [12] Marsan, A., Balbo, G., Conte, G., Donatelli, S. & Franceschinis, G., 1995. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons.
- [13] Sanders, W. H. & Meyer, J. F., 2002. *Stochastic activity networks: Formal definitions and concepts*. Lectures on Formal Methods and Performance Analysis. Springer Verlag.

- [14] Mobius. <http://www.mobius.illinois.edu/>.
- [15] De Alfaro, I., 1998. *Stochastic Transition Systems*. Lecture notes in computer science. Vol. 1466, pp. 423-438.
- [16] Vesely, W.E., Goldberg F.F., Roberts N.H. & Haasl D.F., 1981. *Fault tree handbook*. U.S. Nuclear Regulatory Commission, Washington DC.
- [17] Bloomfield, R. , Buzna, L., Popov, P., Salako, K. & Wright, D., 2009. *Stochastic Modelling of the Effects of Interdependencies between Critical Infrastructure*. Critical Information Infrastructures Security, 4th International Workshop, CRITIS 2009. Lecture notes on computer science. Vol. 6027, pp. 201-212.
- [18] IRRIS project. <http://www.irriis.org/>
- [19] Sanders, W.H., 1988. *Construction and Solution of performability models based on stochastic activity networks*. PhD Thesis, University of Michigan.
- [20] Reibman, A., Smith, R. & Trivedi, K.S., 1989. *Markov and Markov reward models transient analysis: an overview of numerical approaches*. European journal of operational research. Vol. 40(2), pp. 257-267.
- [21] Bobbio, A. & Raiteri, D.C., 2004. *Parametric fault tree with dynamic gates and repair boxes*. Annual symposium on RAMS, pp. 459-465.
- [22] Boudali, H., Crouzen, P. & Stoelinga, M., 2007. *Dynamic Fault Trees analysis using Input/Output Interactive Markov chains*. 37th IEEE international conference in Dependable systems and Networks, pp. 708-717.
- [23] Distefano, S., & Puliafito, A., 2007. *Dynamic reliability block diagrams vs dynamic fault trees*. In Proceedings Annual Reliability and Maintainability Symposium RAMS '07; 71-76.
- [24] Bouissou, M. & Bon, J.L., 2003. *A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes*. Reliability Engineering and System Safety; 82:149-163.

- [25] Muppala, J.K., Ciardo, G. & Trivedi, K.S., 1994. *Stochastic reward nets for reliability prediction*. Communication in Reliability, Maintainability and Serviceability.
- [26] PERFORM, 2006. *Möbius: Model Based Environment for Validation of System Reliability, Availability, Security and Performance*. User's Manual, v. 2.0 Draft.
- [27] Sahner, R.A. & Trivedi, K.S., 1987. *Reliability modelling using SHARPE*. *IEEE Transactions on Reliability*. Vol. R-36(2), pp. 186-193.

CHAPTER 6

6.1 INTRODUCTION

In this chapter we present a procedure to implement Adaptive Transition System (ATS) into Stochastic Activity Network (SAN) [1]. The choice of SANs follows from the fact that is possible to specify in a straightforward manner quantities like system and transition variables introduced in previous chapter.

The remainder of the chapter is structured as follows: in Section 2 we give a brief introduction of the Mobius tool [2], a tool that allow graphical implementation of SAN models. In Section 3 we illustrate the procedure for the conversion of ATS model to SAN models. In Section 4 the Heat-Power system presented in Chapter 5 is used as a tutorial example for the construction procedure and in Section 5 we report some conclusion.

6.2 MOBIUS MODELLING TOOL

Mobius [2] is a software tool, developed at the University of Illinois, for modelling the behaviour of complex systems. The first version was released in 2001 as a successor to UltraSAN. Although Mobius was originally developed for studying the reliability, availability, and performance of computer and network systems, its use has expanded rapidly. The flexibility and power found in Mobius comes from its

support of multiple high-level modelling formalisms and multiple solution techniques.

This flexibility allows engineers and scientists to represent their systems in modelling languages appropriate to their problem domains, and then accurately and efficiently solve the systems using the solution techniques best suited to the systems size and complexity. Time and space efficient distributed discrete event simulation and numerical solution are both supported.

A model in Mobius is represented by a Join/Rep composition of atomic models. Atomic models can be SAN models, PEPA models, FT models, etc. [3,4]. In our case each atomic model is a SAN [1]. In particular for our purpose an atomic model is the SAN representation of a single ATS. Atomic models can be joined together via the *REP* construct, i.e., an atomic model is replicated N times or by the *JOIN* construct where variables can be shared across atomic models.

Mobius allows to specify a reward function based on the variables (places) defined in the model that can be evaluated via numerical solution or discrete event simulation.

6.3 SAN IMPLEMENTATION OF ATS

We represent ATS in terms of SAN atomic models that are composed together by the *JOIN* construct. Modelling concerns are: (i) the data structure of the variables of ATS; (ii) the model of transitions and (iii) the updating structure.

6.3.1 SYSTEM AND REWARD VARIABLES DATA STRUCTURE

The three classes of *system variables* of an ATS are: state indicator, transition indicator and transition timing variables. We use a SAN representation based on extended places. In this way is possible to organize the data in two-dimensional matrices, one for each class of variables.

Let consider a ATS model with NS transition systems. Let S_i ($s = 1, 2, \dots, NS$) be the i -th ATS, $Q_{i,j} \in Q_i$ ($j = 1, 2, \dots, NQ_i$) denote the j -th state of S_i and $T_{i,k} \in T_i$ ($k = 1, 2, \dots, NT_i$) be the k -th transition of S_i .

For each ATS, S_i , are defined the following variables:

- $s_{i,j}$, is the state indicator variable of $Q_{i,j}$;
- $t_{i,k}$, is the transition indicator variable of $T_{i,k}$; and
- $\pi_{i,k}$, is the transition timing variable of $T_{i,k}$.

The extended place Is is a two-dimensional data structure for state indicator variables. The first dimension specifies the ATS model S_i ; while the second dimension states of S_i , such that:

$$mark(Is(i, j)) = s_{i,j}(n) \quad (i = 1, 2, \dots, NS; \quad j = 1, 2, \dots, NQ_i), \quad (6.1)$$

The extended place It is a is a two-dimensional data structure for transition indicator variables. The first dimension specifies the ATS model S_i ; while the second dimension transitions of S_i , such that:

$$mark(It(i, j)) = t_{i,k}(n) \quad (i = 1, 2, \dots, NS; \quad j = 1, 2, \dots, NT_i), \quad (6.2)$$

The extended place Tt is a is a two-dimensional data structure for transition timing variables. The first dimension specifies the ATS model S_i ; while the second dimension transitions of S_i , such that:

$$mark(Tt(i, j)) = \pi_{i,k}(n) \quad (i = 1, 2, \dots, NS; \quad j = 1, 2, \dots, NT_i), \quad (6.3)$$

Finally, the extended place R is a is a vector for reward variables. Given NR reward variables, R is NR dimensional vector, such that:

$$mark(R(i)) = v_R(n) \quad (i = 1, 2, \dots, NR), \quad (6.4)$$

In order to allow to store non integer values, Tt and R are extended places with data type *double*. However, in the case reward variables specify measures of interest like reliability, availability etc., the extended place can be of the type *int*. Also Tt can

be of the type *int* if transition timing variables are replaced by ranking variables as in Ordered-ATS (OATS). The use of *int* values allows one to transform the SAN model in a *base model* that can be solved numerically.

Finally, if reward variables are not integer variables, a numerical solution can be obtained only if those rewards are not inputs of transition functions. In this case reward variables do not need to be specified as a part of the net but can be specified in the Mobius interface *Reward* and used to obtain a Markov Reward Model [5].

In this section we show how to build a SAN model derived by an ATS model that is suitable for discrete event simulation. SAN models of ATS suitable for numerical evaluation are not considered but their construction can be easily implemented following the rules given in this chapter and the ones defined in Chapter 5 about the conversion of transition timing variables to ranking variables.

Figure 6.1 shows the extended places used to represent system and reward variables of ATS in SAN. The extended places presented here are specified in each SAN atomic model of ATS and are shared via the *JOIN* construct.

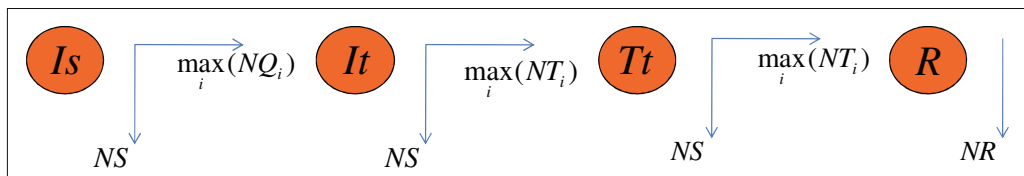


Fig.6.1. Extended places of system and reward variables and data structure.

6.3.2 SAN REPRESENTATION OF ATS TRANSITIONS

Having defined the SAN representation of system variables let describe the SAN implementation of ATS transitions.

Each mode of an ATS transition is represented by a SAN activity. Given the transition $T_{i,k}$ with $NM_{i,k}$ modes $mode_{i,k}^m$ we define an activity $ACT_{i,k}^m$ ($m = 1, 2, \dots, NM_{i,k}$) for each of them.

The type of activity is specified by the type of the mode $type_{i,k}^m$ (timed, instantaneous, deterministic, etc.).

Parameter and reactivation functions of modes of transitions are specified as the attributes of the respective activity in the Mobius graphical interface. In particular the reactivation variable is specified in the *execution policy* as the *reactivation predicate*.

The *activation function* of modes of transitions are specified by the mean of input gates. In particular an input gate, $IG_{i,k}^m$ is defined for each activity $ACT_{i,k}^m$ representing a mode of transitions. The *input predicate* of these input gates specifies the activation function.

Finally, the input gate $IG_{i,k}$ is connected to each $ACT_{i,k}^m$ in order to enable the activity if the system is in the state where the transition departs from. If $T_{i,k}$ is a transition departing from $Q_{i,j}$, we define the input predicate of $IG_{i,k}$ by the relation $mark(Is(i,j))$.

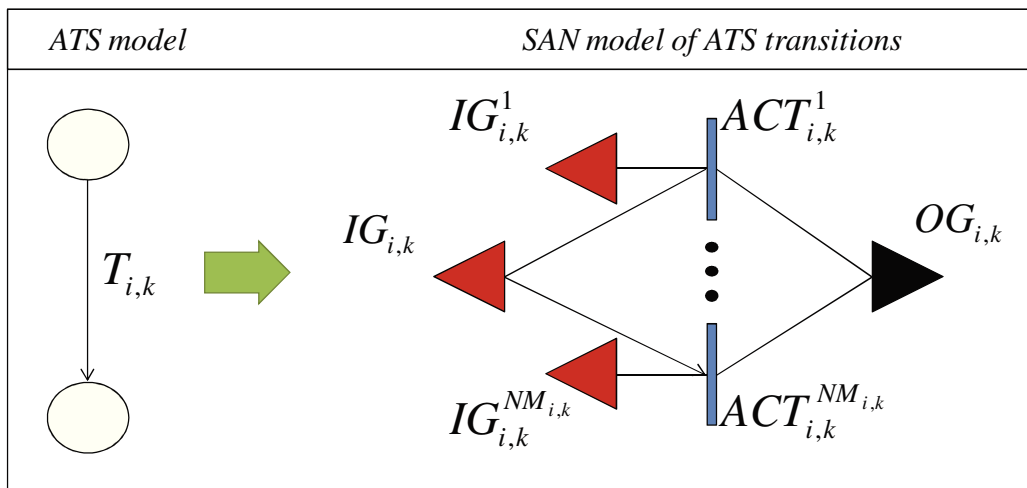


Fig. 6.2. SAN model of ATS transitions ($T_{i,k}$).

6.3.3 ATS-SAN UPDATE STRUCTURE

For each transition is specified an output gate connected to each SAN activity that models the transition. $OG_{i,k}$ is the output gate connected to each $ACT_{i,k}^m$. The output gate is used to set the new conditions after the firing of an activity (or the completion of a transition). Given $T_{i,k}$, the transition that connects $Q_{i,j}$ to $Q_{i,j+1}$, the following code is specified in $OG_{i,k}$:

- state indicator variables:
 $\mathbf{mark(Is(i, j)) = 0;}$
 $\mathbf{mark(Is(i, j + 1)) = 1;}$
- transition indicator variables:
 $int\ x = mark(A(1));$
 $int\ y = mark(A(2));$
 $mark(It(x, y)) = 0;$
 $\mathbf{mark(It(i, k)) = 1;}$
 $\mathbf{mark(A(1)) = i;}$
 $\mathbf{mark(A(2)) = k;}$
- transition timing variables
 $\mathbf{mark(Tt(i, k)) = \tau_n;}$
- reward variables
 $R(z) = f_{R,z}(\cdot); \forall z = 1, 2, \dots, NR.$

A is a support place used to store the memory address of the last completed transition. Reward functions can be different. The update of reward variables must be done after the update of system variables.

In bold are reported the part of the code that is different for each transition of the model. An automated conversion is possible and would simplify the task of writing the appropriate code.

Transition variables are updated automatically by Mobius by the evaluation of input predicates of input gates and parameters and execution policies of activities.

6.4 ATS-SAN MODEL OF THE HEAT-POWER SYSTEM

In this section we show how to build the ATS.SAN model of the Heat-Power system introduced in Section 5.4 following the guidelines presented above.

Parameters of the Heat-Power systems that are needed to be considered for the construction of the ATS-SAN model are:

- S_0, S_1, S_2, S_3 , the ATS representing P0, P1, P2 and G, respectively;
- $NS = 4$, the number of ATS;
- $NQ_0 = NQ_1 = NQ_2 = NQ_3 = 2$, the of states in each ATS;
- $NQ_{max} = 2$, the maximum number of states among ATSS;
- $NT_0 = NT_1 = NT_2 = NT_3 = 1$, the number of transitions inn each ATS;
- $NT_{max} = 1$, the maximum number of transitions among ATSS;
- $NR = 1$, the number of reward variables;

Given the information above, the structure of the extended places is:

- $Is (NS \times NQ_{max})$, is a 4x2 matrix;
- $It (NS \times NT_{max})$, is a 4x1 matrix;
- $Tt (NS \times NT_{max})$, is a 4x1 matrix;
- $R (NR \times 1)$, is a 1x1 matrix.

The ATS-SAN model is built implementing an SAN atomic model of each ATS model of the components of the system. The ATS-SAN model of P0 is shown in Figure 6.3. The ATS is made up only of one transition, thus the ATS-SAN model is composed of the following elements: the model of the transition of S_0 ; and the extended places used to store the variables of the model.

Since $T_{0,0 \rightarrow 1}$ (or $T_{0,0}$) has two modes, two activities are present in the ATS-SAN model. $ACT_{0_0_1}$ is the activity representing the first mode of the transition. It is a timed exponential activity, with fixed parameter. No reactivation predicates is assigned to the activity. $ACT_{0_0_2}$ is the activity that represents the second mode of the transition. Since the mode of transition is instantaneous, an instantaneous activity is used to represent this mode. Each activity is assigned an exclusive input gate and a shared input gate. $IG_{0_0_1}$ is the exclusive input gate assigned to $ACT_{0_0_1}$. The input predicate of the gate is a specification of the activation function of the mode of the transition in terms of marking of the involved variables defined in the form of the extended places introduced above.

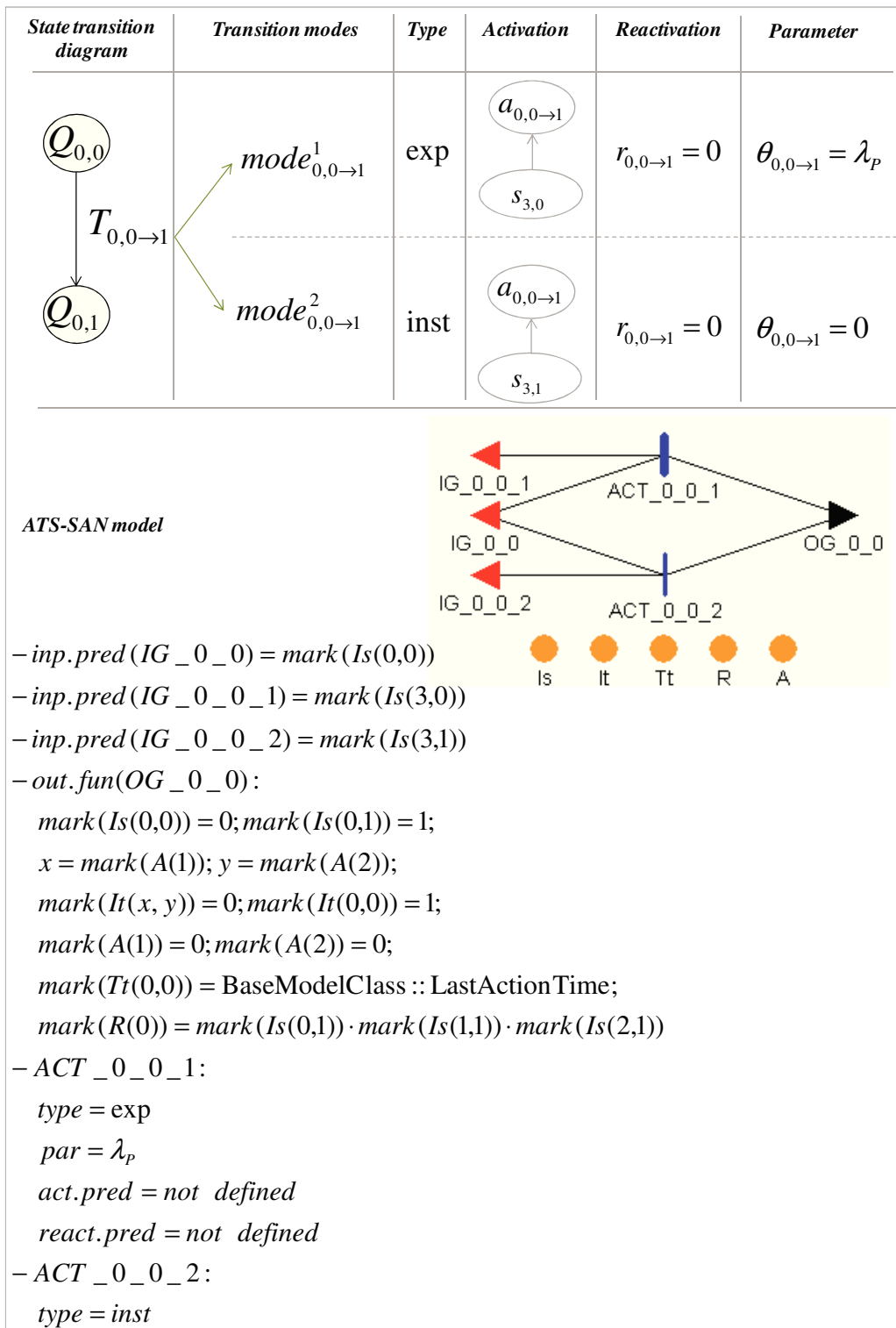
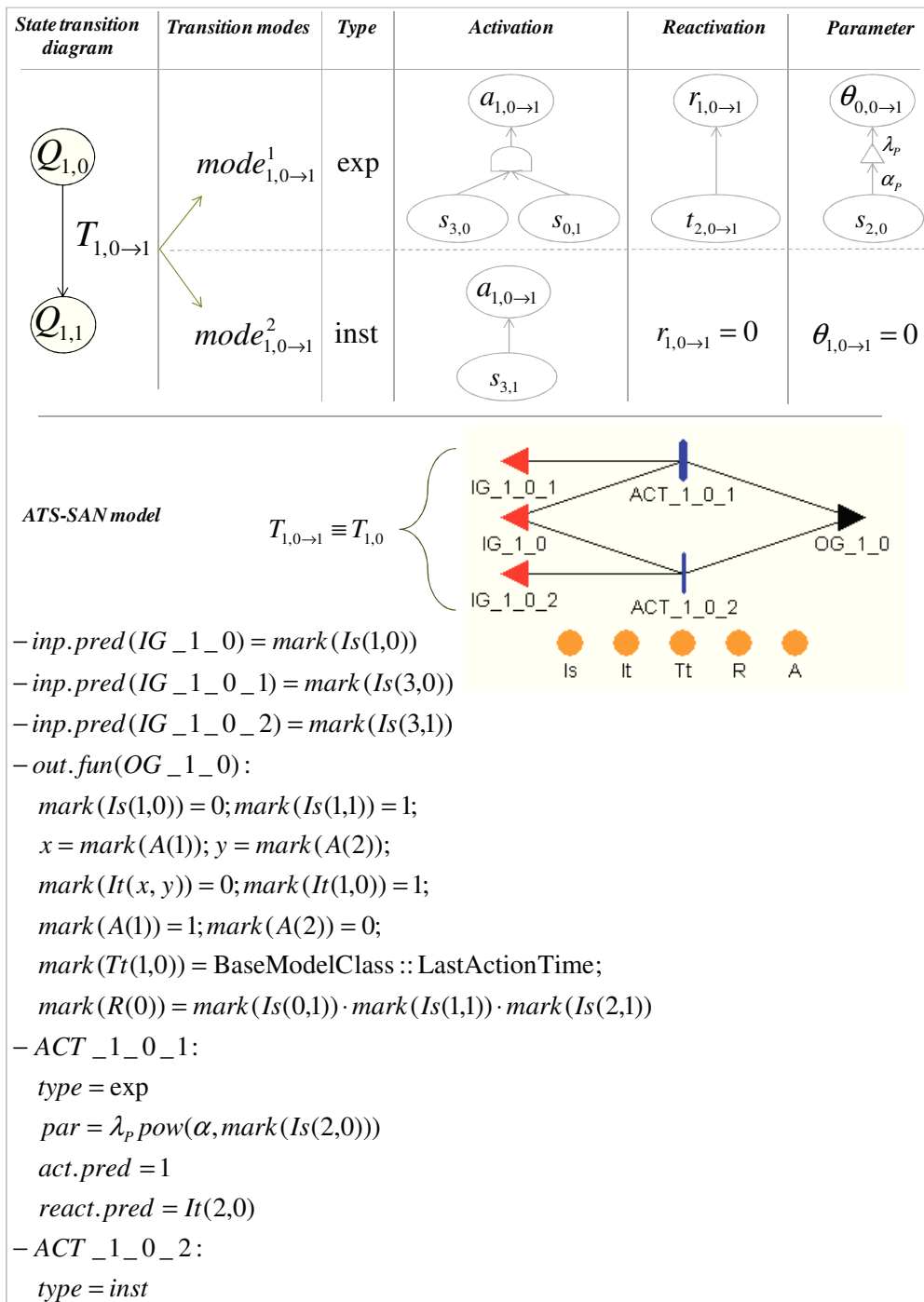


Fig. 6.3. ATS-SAN model of S_0 (or of the pump P0).


 Fig. 6.4. ATS-SAN model of S_1 (or of the pump P1).

Since the activation variable of the considered mode of transition takes on value 1 if the generator G is in the working state, the input predicate of the gate is set to $mark(Is(3,0))$, where $Is(3,0)$ represent the state indicator variable $s_{3,0}$. At the same

way is defined the input predicate of $IG_{0_0_2}$, the exclusive input gate of $ACT_{0_0_2}$, the activity that represents the second mode of the considered transition.

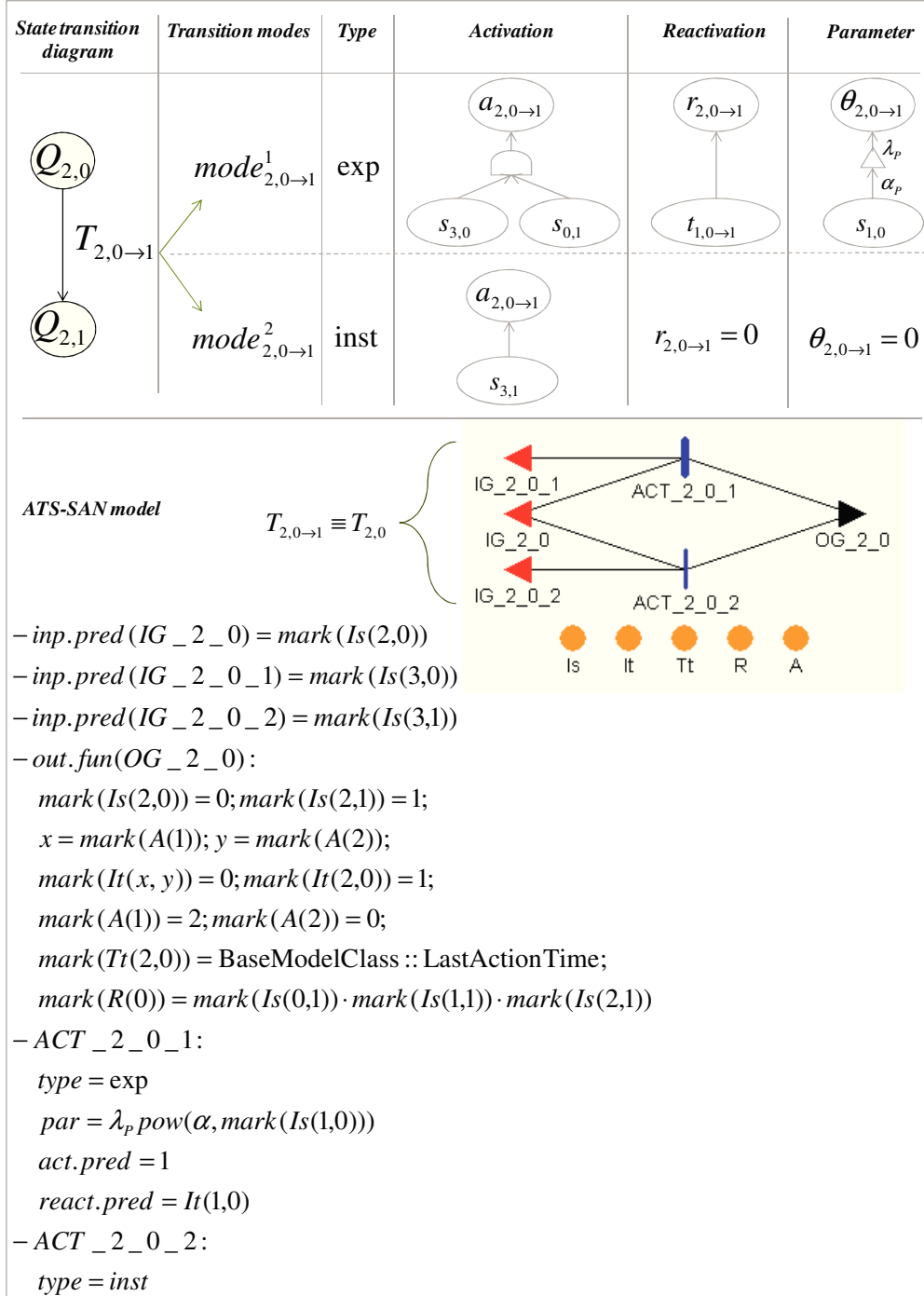


Fig. 6.5. ATS-SAN model of S_2 (or of the pump P2).

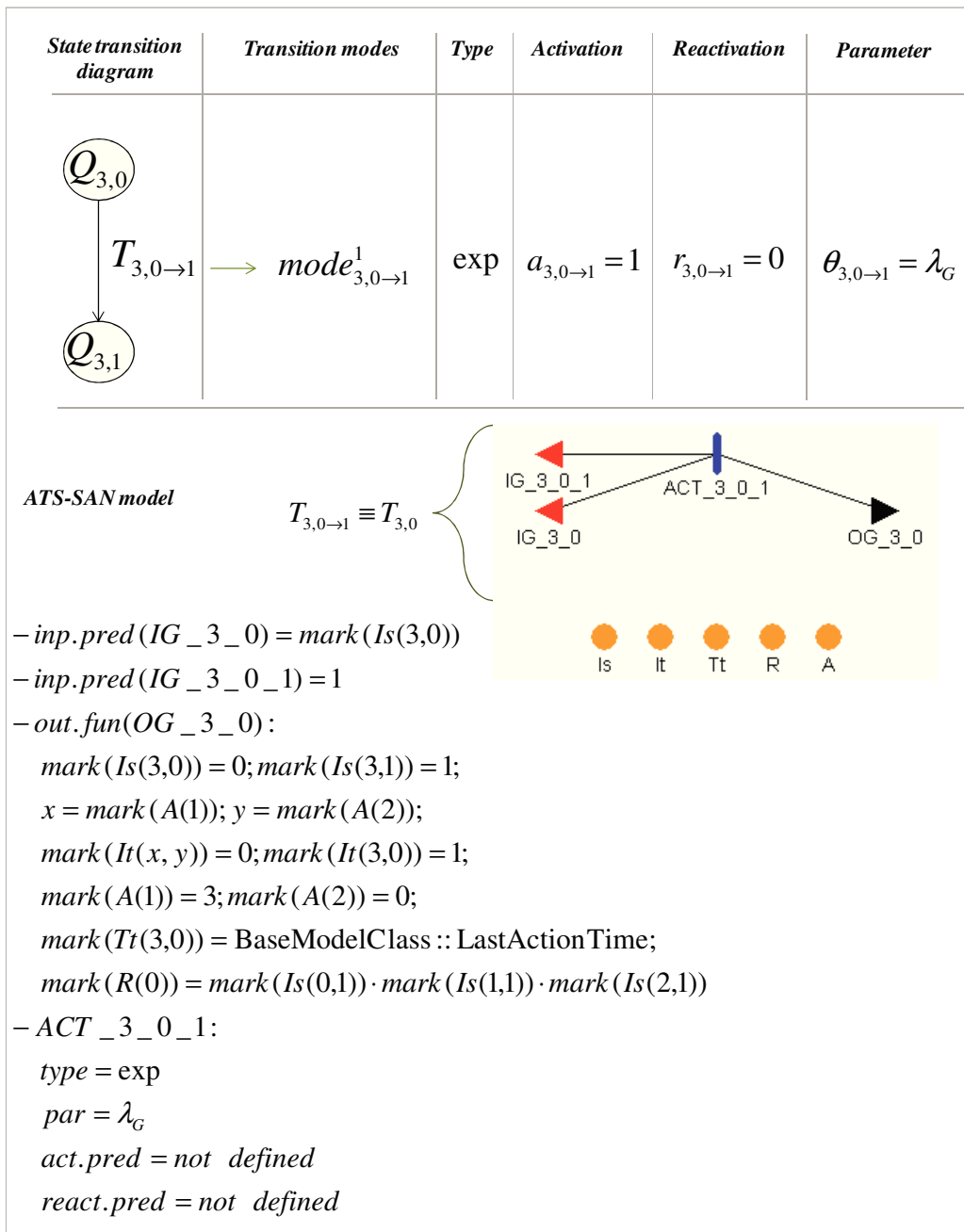


Fig. 6.6. ATS-SAN model of S_3 (or of the generator G).

Finally, the output function of the output gate OG_0_0 updates the values of the marking of the extended places representing the system and reward variables.

Figure 6.4 and 6.5 show the ATS-SAN model for the pump P1 and P2. Here differently than the preceding case, the parameter of the timed activity is state dependent and the reactivation predicate must be specified.

Finally the ATS-SAN model for the generator G is shown in Figure 6.6. Here, the transition has only one mode, thus only one activity is used to model the ATS transition.

SAN atomic models are composed into a single model by the *JOIN* construct (Figure 6.7).

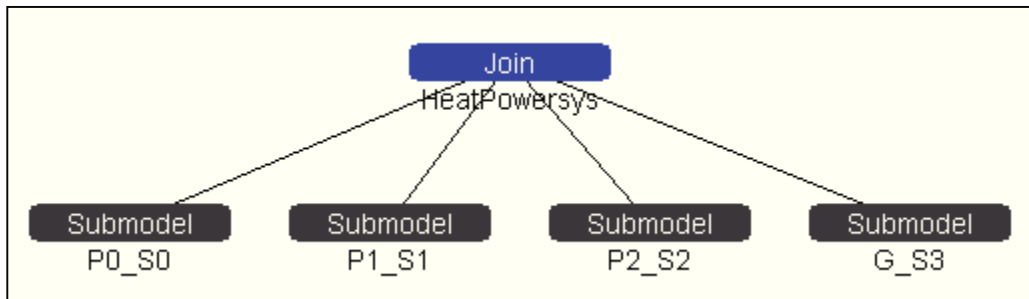


Fig. 6.7. Model composition of ATS-SAN models for the Heat Power system.

6.5 CONCLUSION

In this chapter we have present a conversion procedure of ATS models into SAN model. We have seen that this approach leads to SAN models with a standard structure. In fact while tool's flexibility is a great asset in building a model, the existence of many ways of constructing the same logic may become a problem. The subjective preferences of the modeler dictate which of the many modeling alternatives will be taken, but this option may be difficult for the others to understand. This may result in poor maintainability of the models over time, e.g. high propensity of introducing errors when modifying existing models which could have been avoided had the modeling options been more limited and well known to the modeler who has developed the model and those in charge of its maintenance.

We believe that in this dissertation we make a small step towards alleviating both problems.

First, we propose a set of graphical notations for modeling the predicates common for the SAN models, i.e., the f -T model of transition functions. We also advocate the use of these graphical notations for modeling explicitly the dependencies between the modeled SAN elements: e.g. instantaneous state change or change of model parameters as a function of the model overall state.

Second, we show that the proposed graphical notations can be *automatically converted* into equivalent SAN fragments. The advantage of so doing is not merely streamlining the construction of the models and communicating more easily to the domain experts what the model is doing, but also simplifying the models' debugging – replacing the manually constructed code with a tool generated, ideally highly optimized code.

We consider this framework an important addition to the well known formalisms for stochastic modelling of complex systems because it offers a standardized way of building models of complex systems. We believe that standardization will alleviate the difficulties currently experienced by modellers to maintain each other's or even own complex stochastic models caused by the modelers' preferences in constructing models, which may be either unfamiliar to others or difficult to understand. We believe that promoting standardization and visualization in constructing models is also likely to improve model comprehension, make the model validation easier and even make model maintenance easier: changes in most cases will be applied at a higher level of abstraction to the graphical representations of the transition attributes and various predicates.

Hence, no intimate knowledge of the underlying implementation details (e.g. knowledge of the C++ classes used by the tool) is required.

Finally, improved usability of the tools, in our opinion, will make it much more likely for domain experts to engage more closely to scrutinize the models and, thus, improve their quality.

Exploiting the full potential of the framework will require tool support, more specifically for visually modelling ATS and for automatic transformation to the

chosen target modelling environment, e.g. Mobius. These are concerns which we intend to address in our future work.

BIBLIOGRAPHY

- [1] Sanders, W. H. & Meyer, J. F., 2002. *Stochastic activity networks: Formal definitions and concepts*. Lectures on Formal Methods and Performance Analysis. Springer Verlag.
- [2] PERFORM, 2006. *Möbius: Model Based Environment for Validation of System Reliability, Availability, Security and Performance*. User's Manual, v. 2.0 Draft.
- [3] Clark, A., Gilmeore, S., Hillston, J. & Tribastone, M., 2007. *Stochastic Process Algebras*. Lecture notes in computer science, Springer. Vol. 4486, pp. 132-179.
- [4] Vesely, W.E., Goldberg F.F., Roberts N.H. & Haasl D.F., 1981. *Fault tree handbook*. U.S. Nuclear Regulatory Commission, Washington DC.
- [5] Reibman, A., Smith, R. & Trivedi, K.S., 1989. *Markov and Markov reward models transient analysis: an overview of numerical approaches*. European journal of operational research. Vol. 40(2), pp. 257-267.

CHAPTER 7

7.1 INTRODUCTION

In this Chapter we present the conversion procedure to generate an Adaptive Transition System (ATS) model of Repairable Dynamic Fault Trees (RDFT) [1-3]. Due to the complexity induced by repairable components no much attention has been given to the extension of DFT to the repairable case. We show that ATS offers a effective way to model RDFT.

In fact, the extension to repairable elements involves the definition of new rules for dynamic gates. We show that several scenarios can be considered depending on the substitution logic of spare components, the effect of triggers and maintenance management.

In order to model RDFT we propose to decompose behavioural aspects from failure logics. Behavioural aspects concern the definition of substitution logics between spare components, trigger effects, maintenance policies, etc. while the failure logic is responsible to define the system configurations that leads to the occurrence of the top event of the DFT, i.e., the structure function [4]. This is done converting the DFT to its related static Fault Tree [4]. Behavioural aspects are tackled by the definition of opportune ATS models that will be attached to the leaf of the static FT derived from the original DFT.

To this end, after that we have chosen a specific set of logics for dynamic gates with repairable components, we propose a classification of the basic event (BE) of DFT. BE will be regarded as primary or spare. Depending on the class of the BE, an ATS with a standard structure can be defined. The ATS model will carry in all the dynamic (behavioural) aspects of the system, modelled by an opportune definition of the transition functions of the model.

On the other hand, the part of the DFT representing the fault logic of the system, free of any dynamic aspect, will be used to define the reward function of the model. The reward function will have the form of a static FT derived from the original DFT.

In the following we will refer to a DFT from a case study taken from the literature [5]. The case study was chosen due to the presence of SPARE gates, FDEP gates and shared spare components.

The remainder of this chapter is structured as follow: in Section 7.2 we introduce the DFT of the case study and show of to convert it into its static representation. In Section 7.3 we classify DFT components and define, on the basis of this classification the ATS models that will be attached to the leaf of the static representation of the DFT. In Section 7.4 the DFT introduced in Section 7.2 is solved through an ATS-SAN [6] implementation as described in Chapter 6. In Section 7.5 we introduce a tool that is currently under development. The tool is an extension of the MatCarloRe tool presented in Chapter 4 to solve repairable DFTs. Finally in Section 7.6 are reported some conclusions.

7.2 STATIC REPRESENTATION OF DFT

Figure 7.1 shows the DFT model of a system taken from the literature [10]. In [10] is assumed that all components can be only in two states (working/failed), they are non-repairable and characterized by a time to failure exponentially distributed. Here we relax the assumption that components are not repairable showing how they can included in the DFT model.

The components of the DFT of Figure 7.1 are:

- basic events, A1, A2, B1, B2, S, T1, T2, T3;
- gate A, SPARE gate with an active component, A1, and two spares, A2, S;
- gate B, SPARE gate with an active component, B1, and two spares, B2, S;
- gate F1, FDEP gate with trigger T1 and components A1 and B2;
- gate F2, FDEP gate with trigger T2 and components B1 and A2;
- gate F3, FDEP gate with trigger T3 and component S,
- gate TE, AND gate with gates A, B, F1, F2, F3 as inputs.

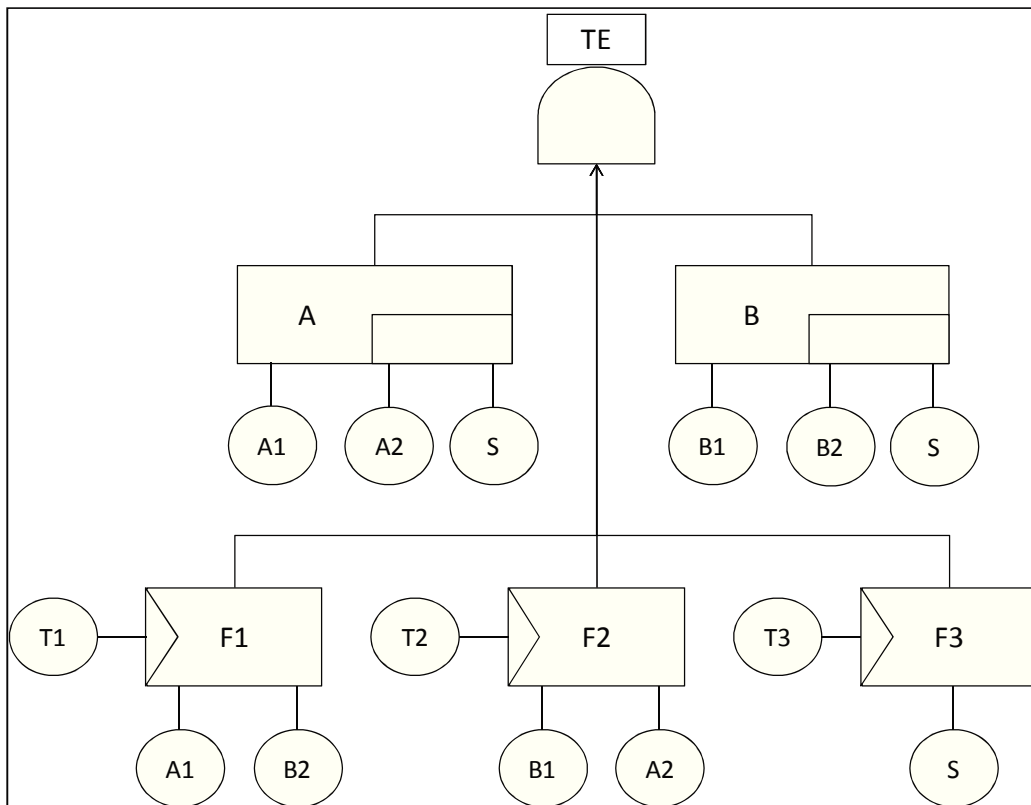


Figure 7.1. Selected DFT.

In order to build an ATS model of the system we start replacing dynamic gates with static gates.

7.2.1 STATIC REPRESENTATION OF SPARE GATES

SPARE gates are converted into AND gates. Here the challenge is to model the behaviour of spare components. In case of repairable components we should ask the following questions:

- given that a spare component is active, when a component to the left-hand side of the component becomes again available, does the active component switch to the stand-by state or remains active?
- if a spare component is shared by more gates, in the case the spare at the moment of its restoring has got more simultaneous calls, which of these will be satisfied?

Although in ATS we may respond to these questions in different ways and in different ways for different subsystems, we choose here to employ a specific configuration. In our setting we will give priority to elements of SPARE gates according to their position as inputs of the gate (from left to right) and we will give priority to SPARE gates according to the temporal ordering of request.

Finally, a shared spare will be named differently in the converted static FT. This because we need to take into account the fact that it can be unavailable for a given subsystem either because failed or because active elsewhere.

7.2.2 STATIC REPRESENTATION OF FDEP GATES

In the case of FDEP gates the following actions are taken:

- said X an input of an FDEP gate that is not a trigger, we consider an OR gate with inputs the trigger of the gate and X ;
- the OR gate replaces X in all positions where X is present in the tree;
- when all inputs components of a FDEP gate (that are not triggers) are replaced by the OR gate defined above, the FDEP gate is eliminated from the tree;

- when a component X is input of more FDEP gates the procedure is repeated iteratively, giving rise to a cascade of OR gates (that can be merged successively in a single OR gate).

With this transformation the assumption we make is that a trigger has not direct influence on the state of the component. Thus the component can still fail or being repaired regardless of the state of the trigger component. The effect on the trigger component is to block the component to fulfil its requirements. The OR gate, indeed, registers a fail when the trigger or the component X have failed.

Other configurations are also possible. For instance, the failure of a trigger can cause the instantaneous failure of its subordinated components. We have shown in Chapter 5 how this can be modelled in terms of ATS (see the Heat-Power systems). However, in this other scenario mechanisms due to the restoring of components must be considered. Some questions may be:

- can a component be restored if its trigger has not been previously restored?
- assuming that a component can be restored even if the trigger is in the failed state, what happen to the component? It fails again instantaneously, or the failure of the trigger has effect on the component only at the moment it fails?

This questions can be addressed by the implementation of an ATS model. However, we will not give more details about these other possible configurations.

7.2.3 STATIC REPRESENTATION OF SEQ GATES

SEQ gates are not present in the DFT in Figure 7.1. The model of SEQ gates is very simple. A SEQ gate is replaced in the static converted FT by a basic event. In fact in the ATS model of a SEQ gate will be represented by a transition system with a number of states equal to the number of inputs of the gate (or the degradable states of the component). The failure state is the one that will be considered in the static converted FT.

7.2.4 PAND GATES

PAND gates are not present in the DFT in Figure 7.1, too. PAND gates do not carry in any behavioural aspect, thus, in the converted static Fault Tree they cannot be eliminated. The conditions that bring to the trigger of the gate are the same as in the not-repairable case. Non-trivial configurations, however, could be defined. An example a possible extension of the logic of the gate can be found in Chapter 5.

7.2.5 STATIC DFT

With the rules stated above the static FT derived from the DFT in Figure 7.1 is shown in Figure 7.2.

SPARE gates A and B are converted into AND gates. Component S is shared among A and B. Thus in the static DFT representation, we label it as S-A and S-B. As we will see, there will be a single ATS model of S. However the variables that will be used into the FT will be different depending on the gate where the component is attached.

FDEP gates are converted in OR gates. For instance F1 is converted in F1-A1 and F1-B2. This because inputs of F1 are A1 and B2. Thus two OR gate will be used to substitute A1 and B2 in the static representation of the DFT.

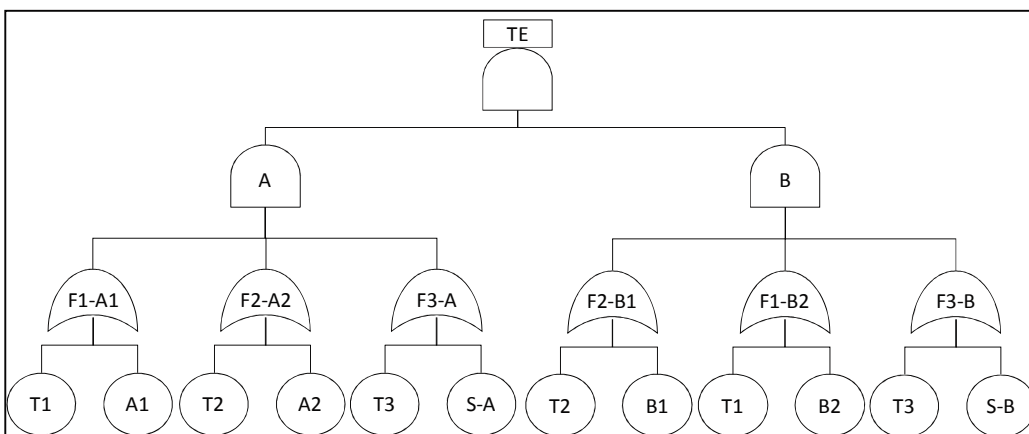


Fig. 7.2. Static Fault Tree of the DFT in Figure 7.1.

The task now is to associate to each Basic Event of the tree an ATS model and define which variables must be used to evaluate the occurrence of the Top Event.

7.3 ATS MODEL OF DFT COMPONENTS

In the previous section we have seen how to convert a DFT into a static FT. In this definition, however, are not considered all the dynamic aspects of dynamic gates. These aspects will be included in the ATS models of components of the system modelled by the DFT. To this end we distinguish those components that are not dependent on any other components, namely the primary components, and those whose behaviour depended on the state of other, namely the spare components.

7.3.1 PRIMARY COMPONENTS

Primary components are those components that are not spare. In the example of Figure 7.1 primary components are: A1, B1, T1, T2, T3. The ATS model of primary components is made of a single transition system with two states representing the working and failed condition. Since we assume that components are repairable, two transitions connect the two states of the ATS model. A transition represents the failure or the repair of a component.

Figure 7.3 shows the ATS model of A1. The remain primary components have the same structure. Here the two state W and F represent the working and failed condition, respectively. $T_{A1,W \rightarrow F}$ and $T_{A1,F \rightarrow W}$ are transitions between these two states.

System variables are:

- *state indicator variables*, $s_{A1,W} \in \{0,1\}$ and $s_{A1,F} \in \{0,1\}$ (representing the fact that the component is in state W or F , respectively);
- *transition indicator variables*, $t_{A1,W \rightarrow F} \in \{0,1\}$ and $t_{A1,F \rightarrow W} \in \{0,1\}$ for $T_{A1,W \rightarrow F}$ and $T_{A1,F \rightarrow W}$, respectively (registering the fact a transition is the one that most recently completed);
- *transition timing variables*, $\pi_{A1,W \rightarrow F} \in \mathfrak{R}$ and $\pi_{A1,F \rightarrow W} \in \mathfrak{R}$ for $T_{A1,W \rightarrow F}$ and $T_{A1,F \rightarrow W}$, respectively (registering the most recent time to complete of a transition).

Since primary components are not dependent on any other component the value of transition variables are fixed. However generalizations that include stochastic associations or different distributions of the time to complete are possible. For instance (in a simulation model) one would take into account that the failure of a component is exponentially distributed until a certain time and Weibull distribute afterwards.

Here we present a model with exponential distribution of time to failure and time to repair. Thus, transitions have only one mode, the exponential one. Activation variables, $\mathbf{a}_{A1,W \rightarrow F} \in \{0,1\}$ and $\mathbf{a}_{A1,F \rightarrow W} \in \{0,1\}$ take on value 1; reactivation variables $\mathbf{r}_{A1,W \rightarrow F} \in \{0,1\}$ and $\mathbf{r}_{A1,F \rightarrow W} \in \{0,1\}$ take on value 0 and parameter variables $\boldsymbol{\theta}_{A1,W \rightarrow F} \in \mathfrak{R}$ and $\boldsymbol{\theta}_{A1,F \rightarrow W} \in \mathfrak{R}$ take on value λ_{A1} and μ_{A1} , respectively.

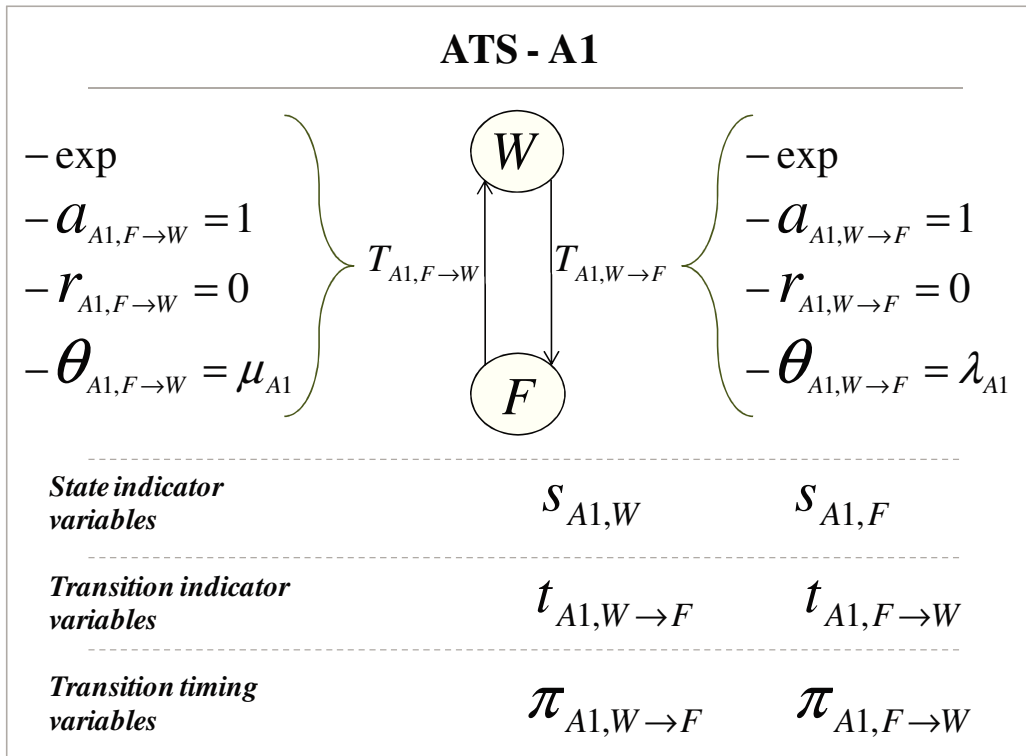


Fig. 7.3. ATS model of primary components (model for A1).

Finally we need to define which variables will be used as inputs (i.e., BEs) of the static representation of the DFT. Since the FT is a model of the system failure we will use state indicator variables of the failure state of components. In particular: $A1 = s_{A1,F}$, $B1 = s_{B1,F}$, $T1 = s_{T1,F}$, $T2 = s_{T2,F}$, $T3 = s_{T3,F}$.

7.3.2 SPARE COMPONENTS

Spare components are those components inputs of spare gates (not primary, i.e., starting from the second to the left in the graphical notation). We model spare components in terms of two ATS models, one capturing aspects related to the working/failed condition and one capturing aspects related to the stand-by/active condition. ATS models of spare components are similar in the structure. The main difference lays in the number of SPARE gates where these components are shared.

A2 and B2 are two spare components that are similar since they are not shared among several SPARE gates and are both of order 1 in the respective gates, i.e., they are the first spare components on the left-hand side of the graphical notation of the gate. Component S, on the other hand is a shared component between the two SPARE gates A and B.

Figure 7.4 and 7.5 shows the ATS models of A2. The model for B2 is similar. In Figure 7.4 is shown the ATS model of the stand-by/active dimension of the state-space of the spare component. Here the two state S and O represent the stand-by and the active, i.e., operative, condition, respectively. $T_{A2,S \rightarrow O}$ and $T_{A2,O \rightarrow S}$ are transitions between these two states.

System variables are:

- *state indicator variables*, $\mathbf{s}_{A2,S} \in \{0,1\}$ and $\mathbf{s}_{A2,O} \in \{0,1\}$ (representing the fact that the component is in state S or O , respectively);
- *transition indicator variables*, $\mathbf{t}_{A2,S \rightarrow O} \in \{0,1\}$ and $\mathbf{t}_{A2,O \rightarrow S} \in \{0,1\}$ for $T_{A2,S \rightarrow O}$ and $T_{A2,O \rightarrow S}$, respectively (registering the fact a transition is the one that completed most recently);
- *transition timing variables*, $\boldsymbol{\pi}_{A2,S \rightarrow O} \in \mathfrak{R}$ and $\boldsymbol{\pi}_{A2,O \rightarrow S} \in \mathfrak{R}$ for $T_{A2,S \rightarrow O}$ and $T_{A2,O \rightarrow S}$, respectively (registering the most recent time to complete of a transition).

Spare components are dependent on other components, thus transition variables are a function of system variables. Here, $T_{A2,S \rightarrow O}$ and $T_{A2,O \rightarrow S}$ are modelled as

instantaneous transitions responsible of the switching of spare components between the two logical states, stand-by and active (here generalizations may include the possibility of a delay for a component to become active). Thus it is by the definition of the transition activation functions that these relations are modelled by the ATS formalism.

In particular a spare component of order 1 switches to the active state in the case of failure of the primary component of the SPARE gate plus the additional condition that the spare element must be itself in the working state. In this case A2 switches to the active state when A1 is failed. In terms of system variables this is achieved when both $s_{A1,F}$ and $s_{A2,W}$ take on value 1. Thus $a_{A2,S \rightarrow O} = AND(s_{A1,F}, s_{A2,W})$.

On the other hand, A2 switches back to the stand-by state when it fails or in the case A1 is restored. Thus $a_{A2,O \rightarrow S} = OR(s_{A1,W}, s_{A2,F})$.

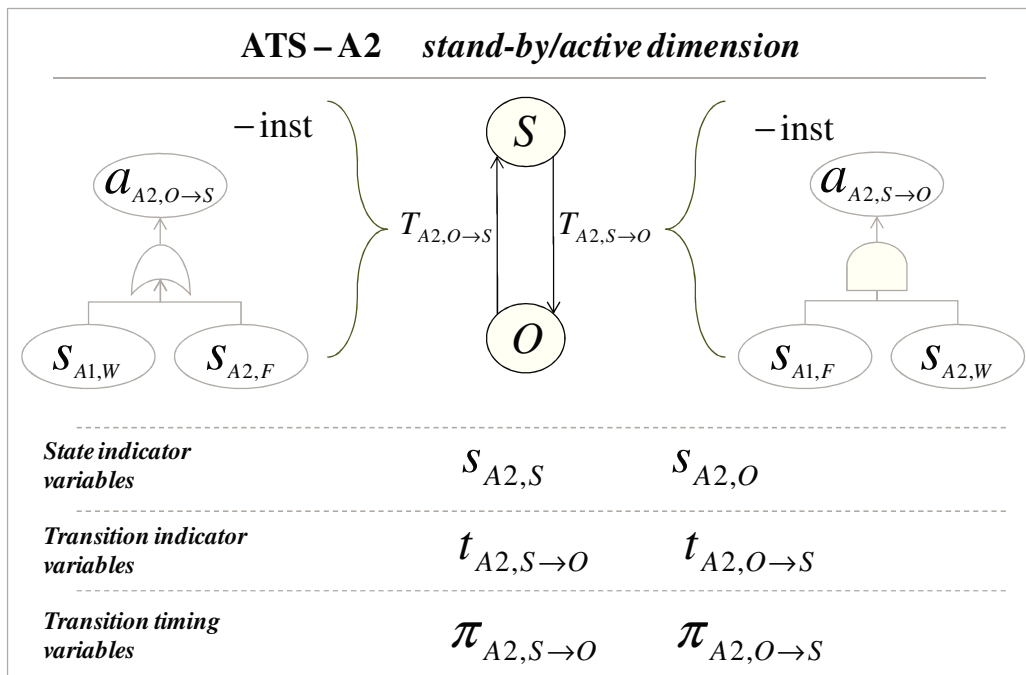


Fig. 7.4. ATS model of the stand-by/active dimension of not shared spare components of order 1 (model for A2).

Defined the model of the stand-by active dimension we can introduce the model of the working/failed dimension of the spare components. This model depends on the fact that a spare component may be shared among more SPARE gates but does not

depend on the order of the spare element in the SPARE gate. This because all the logical relations are captured by the stand-by/active ATS model. Figure 7.5 shows the ATS model of the working/failed dimension of A2.

System variables are similar to the case of primary components. Thus, *state indicator variables* are $s_{A2,W} \in \{0,1\}$ and $s_{A2,F} \in \{0,1\}$; *transition indicator variables* are $t_{A2,W \rightarrow F} \in \{0,1\}$ and $t_{A2,F \rightarrow W} \in \{0,1\}$ for $T_{A2,W \rightarrow F}$ and $T_{A2,F \rightarrow W}$, respectively; and *transition timing variables* are $\pi_{A2,W \rightarrow F} \in \mathfrak{R}$ and $\pi_{A2,F \rightarrow W} \in \mathfrak{R}$ for $T_{A2,W \rightarrow F}$ and $T_{A2,F \rightarrow W}$, respectively.

Variables associated with the transition that represents recovery activities are similar to the one defined for primary components, too. Thus, the activation variable of the “repairing transition” is fixed to 1, the reactivation to 0 and the parameter to the value of the repair rate associated with the component.

On the other hand, in the model of the “failure transition” we must distinguish the case in which the component is a cold or a warm stand-by.

The case of warm stand-by is represented at the top of Figure 7.5. Here we see that the activation variable is fixed to 1, while its the parameter and the reactivation variable depend on the state of stand-by/active dimension.

In particular the value of the failure rate is scaled by a dormancy factor when the component is the stand-by condition. This is modelled by the SL gate defined in Chapter 5. We have that $\alpha_{A2,W \rightarrow F} = SL(\lambda_{A2}, \alpha_{A2}, s_{A2,S}) = \lambda_{A2,F} \alpha_{A2}^{s_{A2,S}}$.

Here, differently from DFT we could define, instead of a dormancy factor, an “active factor”. In this case we could vary the failure rate of a component in dependence of the subsystem where it is employed, i.e., different working loads for different subsystems.

The reactivation variable takes on value 1 whenever there is a change of the parameter. That is when the component enters or exits the stand-by state. To this end the reactivation function is given by OR relation between the transition indicator variables of $T_{A2,S \rightarrow O}$ and $T_{A2,O \rightarrow S}$. We have $r_{A2,W \rightarrow F} = OR(t_{A2,S \rightarrow O}, t_{A2,O \rightarrow S})$.

The case of cold stand-by is shown at the bottom on Figure 7.5. In this case only the activation function of the “failure transition” must be defined, the other variables remain fixed to 0, in the case of the reactivation variables, and to the value of the failure rate, in the case of the parameter.

The condition for the activation of the failure transition is that the component must be not in the stand-by state. Thus we have that $\mathbf{a}_{A2,W \rightarrow F} = \text{NOT}(s_{A2,S})$.

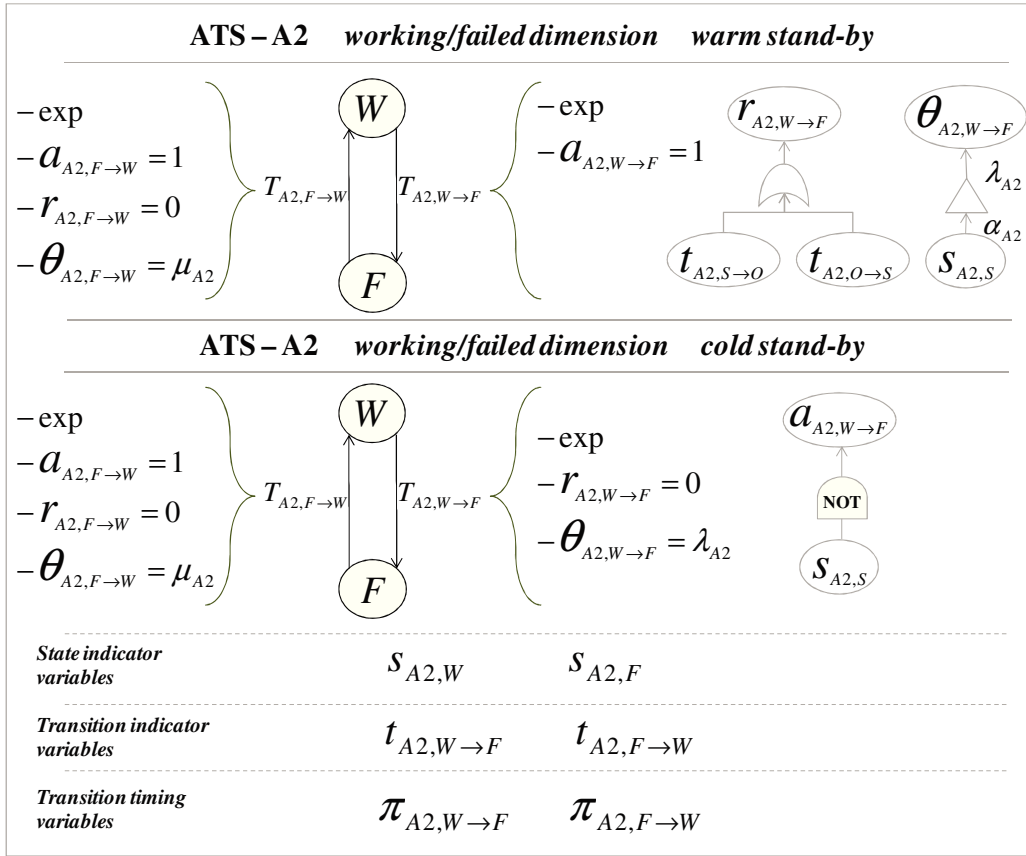


Fig. 7.5. ATS model of the working/failed dimension of not shared spare components (model for A2).

Inputs of the static representation of DFT of Figure 7.2 for A2 and B2 are defined by imposing that the components are failed. Thus, we that $A2 = s_{A2,F}$, $B2 = s_{B2,F}$.

The ATS model of S is different from the model of A2 and B2 in that S is a shared spare among more SPARE gates. In this case the ATS model of the stand-by/active dimension must include the notion of which SPARE gate is served by the component. Thus, two active states are defined for S, OA and OB , the first meaning

the spare is active in the SPARE gate A and the second that it is active in B. Figure 7.6 shows this model. In figure only the transition functions relative to those transitions related to OA are showed, being symmetric the relations on the transitions involving OB .

Transition $T_{S,S \rightarrow OA}$ models the switching of S from the stand-by state to the active state in A. The conditions that allow S to do so are defined by the transition activation function $\mathbf{a}_{S,S \rightarrow OA}$. The output of this function takes on value 1 only if:

- S is in the working state ($\mathbf{s}_{S,W}$) and;
- or A requires S when B does not $\left(AND \left(\mathbf{s}_{A1,F}, \mathbf{s}_{A2,F}, OR(\mathbf{s}_{B1,W}, \mathbf{s}_{B2,W}) \right) \right)$; or
- in the case when both SPARE gates require S, A requested S before B $\left(AND \left(\mathbf{s}_{A1,F}, \mathbf{s}_{A2,F}, \mathbf{s}_{B1,F}, \mathbf{s}_{B2,F}, \leq \left(MAX(\boldsymbol{\pi}_{A1,W \rightarrow F}, \boldsymbol{\pi}_{A2,W \rightarrow F}), MAX(\boldsymbol{\pi}_{B1,W \rightarrow F}, \boldsymbol{\pi}_{B2,W \rightarrow F}) \right) \right) \right)$.

The gate \leq allows a non deterministic choice (if the model of $T_{S,S \rightarrow OB}$ is symmetric to the one of $T_{S,S \rightarrow OA}$) in the case that both A and B requires S at the same time (however this cannot happen if all “failure and repair transitions” are exponential).

Transition $T_{S,OA \rightarrow S}$ models the switching of S to the stand-by state out of state OA . The conditions that allow this change are modelled by the transition activation function $\mathbf{a}_{S,OA \rightarrow S}$. The output of this function takes on value 1 only if:

- S has failed ($\mathbf{s}_{S,F}$); or
- S is not requested anymore from A and it is not requested from B, too $\left(AND \left(OR(\mathbf{s}_{A1,W}, \mathbf{s}_{A2,W}), OR(\mathbf{s}_{B1,W}, \mathbf{s}_{B2,W}) \right) \right)$.

The structure of $\mathbf{a}_{S,S \rightarrow OA}$ may be different if the transition between the two active state $T_{S,OA \rightarrow OB}$ were not present in the model. The choice to include $T_{S,OA \rightarrow OB}$ is to present a model as more general as possible. In fact, $T_{S,OA \rightarrow OB}$ allows not to resample the time to failure of a spare component when switching between two active states.

While in the case of exponential distribution this has no effect, it is important to address this issue when other distributions are used.

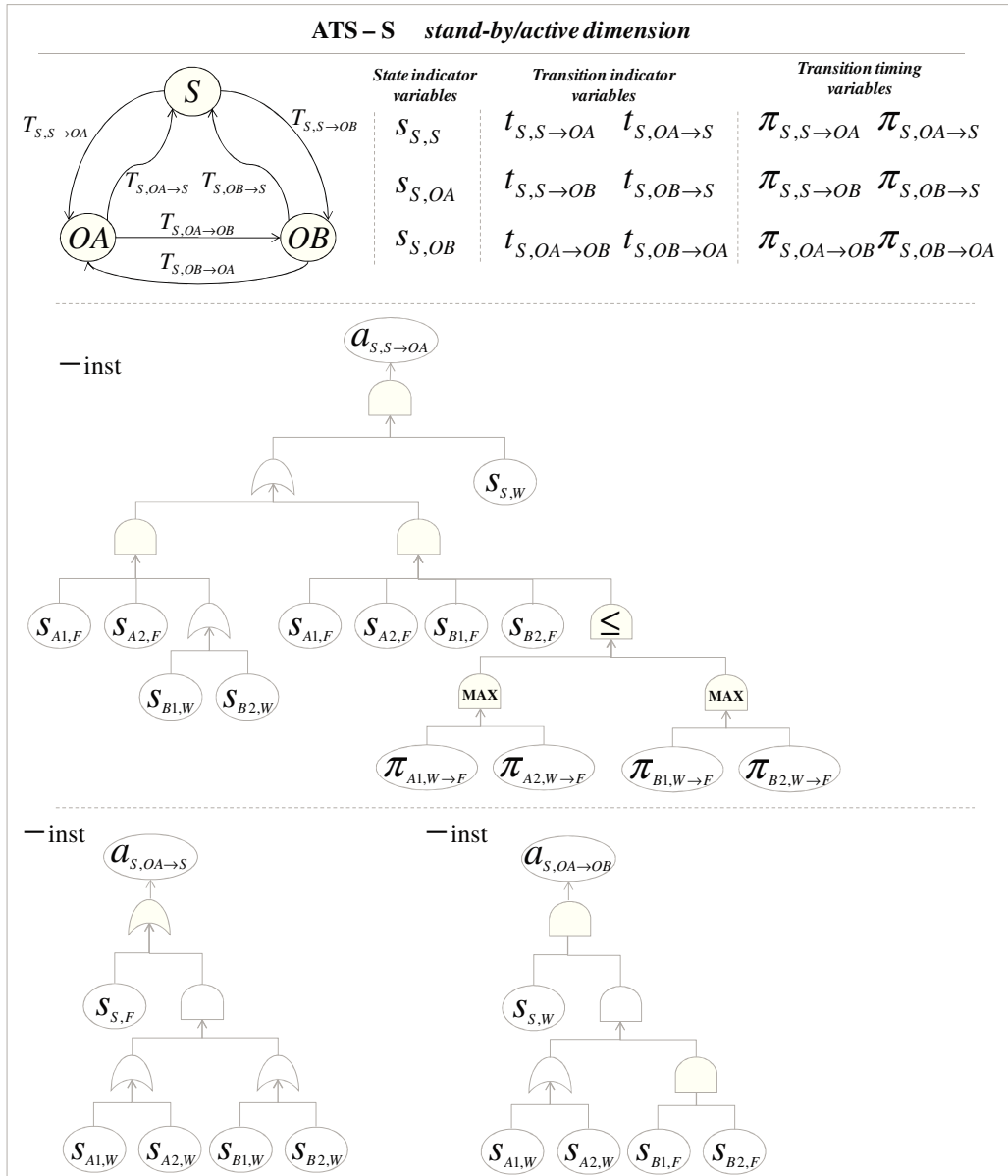


Fig. 7.6. ATS model of the stand-by active dimension of shared spare components (model for S).

The conditions that allow S to switch between two active states are given by the transition activation function $a_{S,OA \rightarrow OB}$, whose output takes on vale 1 only if:

- S is in the working state ($s_{S,W}$); and

- it is not requested anymore from A ($OR(s_{A1,W}, s_{A2,W})$); and
- it is requested from B ($AND(s_{B1,F}, s_{B2,F})$).

The relation presented here for $a_{S,S \rightarrow OA}$, $a_{S,OA \rightarrow S}$, $a_{S,OA \rightarrow OB}$ and symmetrically for $a_{S,S \rightarrow OB}$, $a_{S,OB \rightarrow S}$, $a_{S,OB \rightarrow OA}$ must be generalized in the case that components of lower order in the SPARE gates where S is input, i.e., component A2 and B2, are shared among more SPARE gates. In this case, in fact, the above relations for the transition activation functions must include the possibility that not only a component can be failed but it can also be occupied in another subsystem.

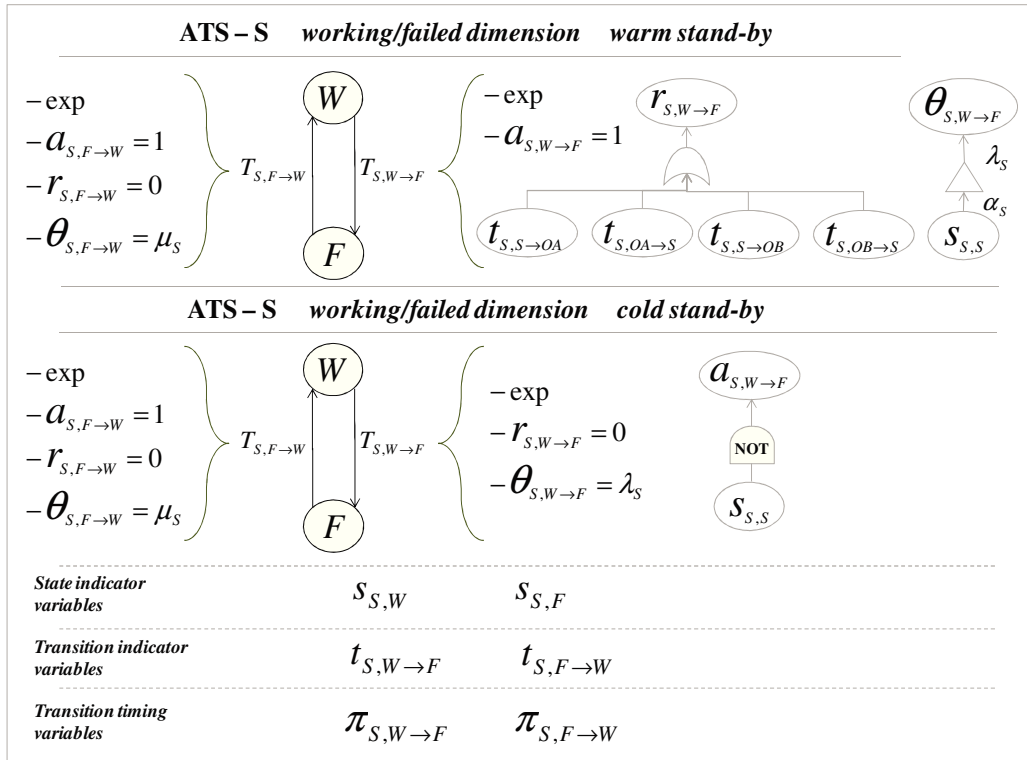


Fig. 7.7. ATS model of the working/failed dimension of shared spare components (model for S).

The ATS model of the working/failed dimension of S is shown in Figure 7.7. The ATS model is similar to the one of A2 presented in Figure 7.5. Here the only difference lays in the definition of the reactivation function of the “failure transition” in the warm stand-by case. In this case in fact the component can enter and leave the

stand-by state by four transitions, thus all the transition indicator variables relative to these transitions must be included as inputs of the function.

Finally, we must define the inputs of the static representation of the DFT of Figure 7.2. We do it imposing the following relations: $SA = 1 - s_{S,W} \cdot s_{S,OA}$ and $SB = 1 - s_{S,W} \cdot s_{S,OB}$. Thus SA takes on value 1 only if either S has failed or it is occupied in another SPARE subsystem. The symmetric condition applies for SB.

7.3.3 REWARD FUNCTION SPECIFICATION

The reward function of the ATS model (Figure 7.8) is retrieved directly from the static representation of the DFT in Figure 7.2 applying the input specified above for the BEs.

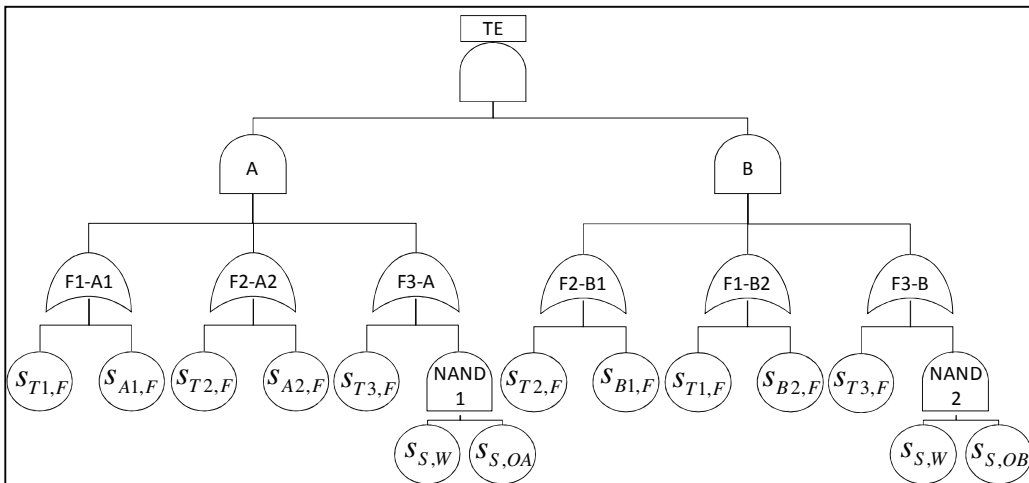


Fig. 7.8. Reward function of the ATS model of the DFT of Figure 7.1.

7.4 SAN IMPLEMENTATION AND EVALUATION

The RDFT presented in the previous sections was solved through its implementation into a SAN model following the indications stated in Chapter 6. The ATS-SAN model of A1 and S is shown in Figure 7.9.

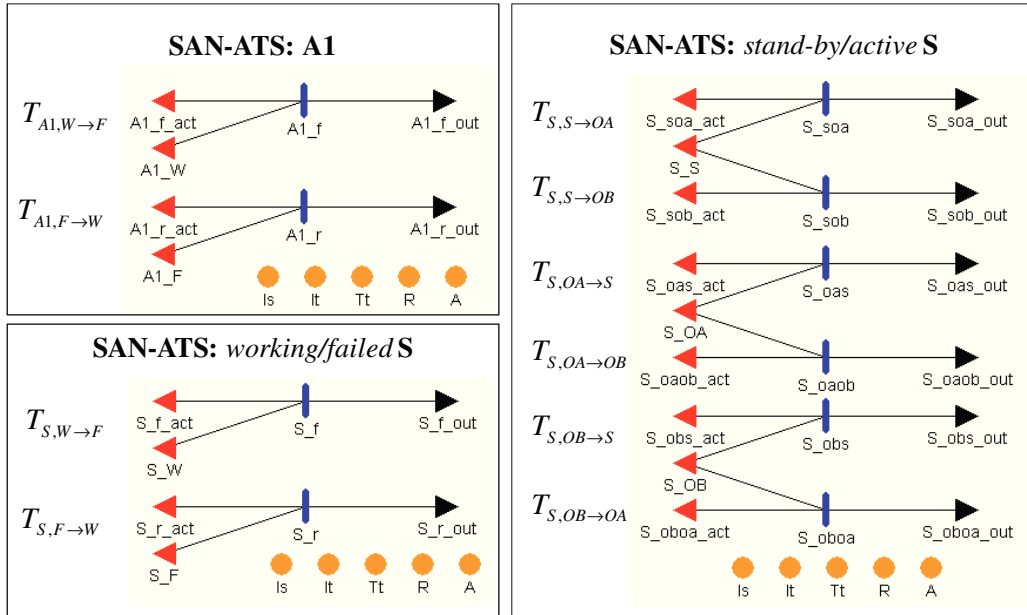


Fig. 7.9. Atomic SAN models of the ATS of components A1 and S.

The time to failure and the time to repair of all components are exponentially distributed and the failure and repair rate values are reported in Table 7.1. From reference [5] the unreliability value of the system at 100 time units is 0.03126.

 Table 7.1. Components failure and repair rates [$time\ unit^{-1}$].

Component	Failure rate λ	Repair rate μ
A1	0.0010	0.025
A2	0.0050	0.025
B1	0.0020	0.025
B2	0.0035	0.025
S	0.0050	0.025
T1, T2, T3	0.0030	0.025

The model was resolved via simulation with the aid of the Mobius simulator [7].

Table 7.2. Simulator Solver results.

Simulation	Batches	Mean	Conf. Int.	CPU time
SAN	59000	6.47e-03	+/-6.47e-04	54.56 sec

The Simulator results are reported in Table 7.2. As expected, the unavailability value is less than the unreliability value reported in [5]. The experiment were carried

out by a laptop with the following characteristics: CPU, Intel Core 2 Duo 1.83 GHz; RAM, 1.99 GB.

7.5 MATCARLOAV: AN EXTENSION TO SOLVE DRFT

An extension of MatCarloRE, the simulative tool for DFT presented in Chapter 4 is currently under development to solve RDFT. MatCarloAv will be a tool for discrete event simulation of RDFT that employs a conversion of the DFT to the related ATS model and solve it via simulation. The tool performs the following steps:

- conversion of the DFT into an ATS model;
- definition of the algorithm for the simulation of the ATS model;
- use of the functions defined in MatCarloRe for the gates of the static FT derived from the original DFT to evaluate the availability (or the reliability with repair) of the RDFT.

The conversion procedure of the DFT into an ATS model follows the same rules stated in this chapter. Once that the model has been converted, a Matlab code for the simulation of the ATS is generated on the basis of the execution logic described in Chapter 5. Finally the reward function is evaluated at every state change through the use of the adapted function of the MatCarloRe tool.

At the moment the tool is still under development. An accepted abstract of a paper describing the tool is attached in Appendix A.

7.6 CONCLUSION

In this chapter we have introduced Repairable Dynamic Fault Tree, a formalism that extend DFTs with repairable components. We have shown what the main issues are when restoring actions are taken on components of a DFT. In particular we have introduced new logics for dynamic gates.

Successively we have shown how ATS can be used to model and solve repairable DFTs. Due to the capability of ATS to tackle the characteristics of single components on the basis of their nature and their configuration into the system, DFT models can be extended to tackle more complex behaviours.

In order to create an ATS model of a RDFT we need first to convert the DFT to a static FT where basic events are modelled in terms of ATS models. In this way the failure logic of the system is separated from the substitution logic of spare components. The FT is used then as a specification of the reward function of the ATS model. It follows that a dynamic behaviour of a system can be modelled in terms of a FT and an ATS. However, dynamic gates represent useful high level representation of a kind of behaviour that is useful to maintain.

However, we showed that with the implementation of f -T, the Fault Tree like graphical representation of transition and reward functions; dependencies existing among components are easily modelled and debugging of the model can be done with the advantages of a graphical support.

Future work regards the construction of a software tool that automatically convert a DFT into an ATS model. The ATS model can be then solved via simulation or via its conversion into a Markov model. In fact when ATS are applied to model DFT, the class of dependencies involved in the model make the ATS an OATS. Furthermore, if only exponential distributions of the time to failure and to repair are used into the model, then the OATS model is a MOATS and can be solved through conversion to a Markov chain.

BIBLIOGRAPHY

- [1] Bobbio, A. & Raiteri, D.C., 2004. *Parametric fault tree with dynamic gates and repair boxes*. Annual symposium on RAMS, pp. 459-465.
- [2] Boudali, H., Crouzen, P. & Stoelinga, M., 2007. *Dynamic Fault Trees analysis using Input/Output Interactive Markov chains*. 37th IEEE international conference in Dependable systems and Networks, pp. 708-717.

- [3] Distefano, S., & Puliafito, A., 2007. *Dynamic reliability block diagrams vs dynamic fault trees*. In Proceedings Annual Reliability and Maintainability Symposium RAMS '07; 71-76.
- [4] Vesely, W.E., Goldberg F.F., Roberts N.H. & Haasl D.F., 1981. *Fault tree handbook*. U.S. Nuclear Regulatory Commission, Washington DC.
- [5] Boudali, H., & Dugan, J., 2005. *A New Bayesian Network Approach to Solve Dynamic Fault Trees*. In Proceedings Annual Reliability and Maintainability Symposium, Jan. 451-456.
- [6] Sanders, W. H. & Meyer, J. F., 2002. Stochastic activity networks: Formal definitions and concepts. Lectures on Formal Methods and Performance Analysis. Springer Verlag.
- [7] PERFORM, 2006. Möbius: Model Based Environment for Validation of System Reliability, Availability, SEcurity and Performance. User's Manual, v. 2.0 Draft.

CONCLUSION

In this thesis we have addressed the issue of reliability modelling of complex interdependent systems.

In Chapter 1, 2, 3 we have introduced the main concepts, models, and techniques of reliability engineering. Chapter 4 shows a Matlab tool for the evaluation of Dynamic Fault Tress via simulation.

In Chapter 5 we have presented the major achievement of this doctoral work: Adaptive Transition Systems (ATS). ATS is an hybrid formalism that integrates concepts deriving from different fields: transition systems, stochastic extension of Petri nets and Fault Trees.

ATS are substantially different from other approach of modelling distributed systems due to the model of communication mechanism that is defined between parallel models. The formalism allows to represent a system in terms of parallel transition systems and a Fault tree-like structure of the model of the communication mechanism.

The communication mechanism defines the dependencies existing between parallel models in the natural way of defining dependencies between system subparts. The model construction results facilitate due to the separation of concerns, i.e., the modeller can abstract from the complexity of the whole (sub-)system when modelling specific system sub-parts. He can focus on the “state-space” of the modelled elements and then define the dependencies among them.

Moreover, debugging and maintenance of the model results also improved because of the “state-space” structure of the model and because changes can be often defined only at the communication mechanism level without the necessity of reconsidering the structure of the “state-space” of the modelled elements.

Some comparison with formalisms like Stochastic Process Algebra and Model checking are also reported. Solutions methods have been proposed, too. We have shown that when an ATS is an Ordered-ATS (OATS) with exponential and instantaneous transition type the OATS is a Markov-OATS (MOATS) and can be resolved through its conversion into a Markov chain. We have shown, that the class of models representable by a Markov chain is extended by an opportune concept of state that include the ordering of the most recent completion of transitions. Finally, being the execution logic of ATS defined on the completion of events, ATS resemble generalized semi Markov process, thus ATS adopts an “educated” approach to simulation.

In Chapter 6 a conversion ATS-to-SAN (Stochastic Activity Network) model is presented showing that the resulting SAN model has a standardized structure where the logic expression of the Boolean predicates of the network are graphically defined by the FT-like formalism used to define the communication mechanism between ATSSs.

Finally in Chapter 7 we introduce Repairable Dynamic Fault Tree (RDFT) and show a lower level conversion method of the RDFT to a an ATS model. In practice by the definition of a specific ATS model is possible to extend DFT to the class of repairable components.

In addition ATS have been applied to model a real system defined as an Interdependent Critical Infrastructure (ICI). The application shows the capability of ATS to deal with interdependent large systems. The main achievement lays in the support to the modelling activity of ATS. We have shown that the task of modelling complex interdependencies results facilitate by the structured approach that ATS provides.

Future developments of ATS regard both theoretical and practical, i.e., tool implementation, fields. From a theoretical point of view it is interesting to exploit the conversion of ATS to Stochastic Process Algebra models like Performance Evaluation Process Algebra (PEPA) or Interactive Markov Chain (IMC). It is also interesting to extend the kind of properties that can be investigated by a n ATS model employing some kind of temporal logic used in Model Checking, e.g., Continuous temporal logic (cTL). We have indicated, in Chapter 5, that ATS can be implemented in the PRISM formalism (a Model Checking tool) if the ATS is an Ordered-ATS.

We have shown that an ATS model can be converted to a Stochastic Activity Network model. In this context it would be beneficial to define a multi formalism language that make use of both ATS and SAN. For instance, the Mobius tool allows to model a system making use of both formalism SAN and PEPA. An extension of the tool to include ATS thus, would be straightforward having defined the conversion procedure ATS-to-SAN.

On the other hand, a standalone tool, e.g., defined in Matlab with a Java interface support, can be defined in a way that both analytical and simulation evaluation techniques can be used in the most general setting for ATS and allowing “exotic” extension.

Finally, from a point of view of ATS development, we should consider the introduction of case probabilities applied to transition, i.e., a transition can be directed to more states with different probabilities, and also among transitions, i.e., different instantaneous transitions with probabilities. Another future development can be directed to the formalization of synchronization procedures between transitions. We have show in Chapter 5 an application of synchronization but has not been formally defined. We remind, in this context, that ATS allows nondeterministic choices and, thus, the definition of synchronizations procedures and case probabilities should take into account also this fundamental aspect often present in distributed systems.

APPENDIX A

In this appendix are reported the following accepted papers in international journals:

- Chiacchio F., Compagno L., D'Urso D., Manno G. & Trapani N., (2011). Dynamic fault trees resolution: A conscious trade-off between analytical and simulative approaches. *Reliab Eng Syst Safety, under press.*
- Manno G. & Chiacchio F., (2011). MatCarloRe: an integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree. *Expert System With Applications, under press.*

and in the proceedings of international conferences:

- Popov P. & Manno G., (2011). The effect of correlated failure rates on reliability of continuous time 1-out-of-2 software. *Lecture notes in computer science, vol. 6894, pp. 1-14.*
- Chiacchio F., Compagno L., D'Urso D., Manno G. & Trapani N., (2011). An open source application to model and solve dynamic fault tree of real industrial systems. *IEEE Proceedings of the 5th International conference on Software, Knowledge Information, Industrial Management and Application (SKIMA).*
- Chiacchio F. & Manno G., (2011). MatCarloAV, an extensible Matlab library for the simulative evaluation of dynamic fault trees. *PSAM 11 & ESREL. Accepted abstract.*



Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Dynamic fault trees resolution: A conscious trade-off between analytical and simulative approaches

F. Chiacchio^{a,*}, L. Compagno^b, D. D'Urso^b, G. Manno^a, N. Trapani^b^a Dipartimento di Matematica e Informatica—DMI, Università degli Studi di Catania, Italy^b Dipartimento di Ingegneria Industriale e Meccanica—DIIM, Università degli Studi di Catania, Italy

ARTICLE INFO

Article history:

Received 4 August 2010

Received in revised form

7 June 2011

Accepted 30 June 2011

Keywords:

Risk assessment

Combinatorial models

Markov chains

Hierarchy

Spreadsheet modeling

ABSTRACT

Safety assessment in industrial plants with 'major hazards' requires a rigorous combination of both qualitative and quantitative techniques of RAMS. Quantitative assessment can be executed by static or dynamic tools of dependability but, while the former are not sufficient to model exhaustively time-dependent activities, the latter are still too complex to be used with success by the operators of the industrial field.

In this paper we present a review of the procedures that can be used to solve quite general dynamic fault trees (DFT) that present a combination of the following characteristics: time dependencies, repeated events and generalized probability failure.

Theoretical foundations of the DFT theory are discussed and the limits of the most known DFT tools are presented. Introducing the concept of weak and strong hierarchy, the well-known modular approach is adapted to study a more generic class of DFT. In order to quantify the approximations introduced, an ad-hoc simulative environment is used as benchmark.

In the end, a DFT of an accidental scenario is analyzed with both analytical and simulative approaches. Final results are in good agreement and prove how it is possible to implement a suitable Monte Carlo simulation with the features of a spreadsheet environment, able to overcome the limits of the analytical tools, thus encouraging further researches along this direction.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

The RAMS techniques offer qualitative analyses and quantitative techniques for risk assessment. The former (such as HAZOP and FMEA [1]) concern the context analysis (kind of process, geographic

issues, internal specifications and rules, etc.) and are used to reveal potential hazards and consequences. The latter concern the risk assessment, computed as the probability of occurrence of undesired events (which are often highlighted by the qualitative analyses).

Two main classes of analytical stochastic models are used for quantitative evaluations:

- combinatorial models (also known as static) that are straightforward, but unable to describe dynamic dependencies among the components of the system and
- state-space models, mostly based on the Markov Chain representation (DTMC, CTMC, MRM, MRGP and GSMP), that overcome many of the limits of the static models but can become too large to be handled [2–4].

In the last years researchers have proposed several techniques, which combine the best properties of the previous models [4,7,8] such as the BDMP [5,6], the DRBD [9], the DFT [10], the SPN [11], etc. These powerful techniques of modeling are implemented using many reliability tools [2,5,12,13,14,17] that can be used according to their own hypotheses and features, which, often, are not suitable to design and solve any possible type of model.

Abbreviations: BDD, Binary Decision Diagram; BDMP, Boolean logic Driven Markov Process; BE, Basic Event; CDF, Cumulated Distribution Function; CTMC, Continuous Time Markov Chain; DAG, Direct Acyclic Graph; DCS, Decision Support System; DFT, Dynamic Fault Free; DRBD, Dynamic Reliability Block Diagram; DTMC, Discrete Time Markov Chain; ExpD, Exponential Distribution of Probability; FDEP, Functional Dependency; FMEA, Failure Mode and Effects Analysis; FT, Fault Tree; FT-A, Fault Tree Analysis; GD, Generalized Distribution of Probability; GSMP, Generalized Semi-Markov Process; HAZOP, Hazard and Operability Study; MCS, Minimal Cut Sets; MOE, Multiple Occurring Event; MOE-FT, Fault Tree with repeated events; MRGP, Markov Regenerative Process; MRM, Markov Rewards Model; (N)HCTMC, (Non) Homogenous Continuous Time Markov Chain; PAND, Priority AND; RAMS, Reliability, Availability, Maintainability and Safety; RBD, Reliability Block Diagram; SEQ, Sequence Enforcing; SFT, Static Fault Tree; SPN, Stochastic Petri Net; TE, Top Event; UnMOE-FT, Fault Tree with no repeated events; Wysiwyg, What you see is what you get

* Corresponding author. Tel.: +39 95 7382412; fax: +39 95 337994.

E-mail addresses: chiacchio@dmi.unict.it (F. Chiacchio), lco@diim.unict.it (L. Compagno), ddurso@diim.unict.it (D. D'Urso), gmanno@dmi.unict.it (G. Manno), ntrapani@diim.unict.it (N. Trapani).

0951-8320/\$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2011.06.014

Please cite this article as: Chiacchio F, et al. Dynamic fault trees resolution: A conscious trade-off between analytical and simulative approaches. *Reliab Eng Syst Safety* (2011), doi:10.1016/j.ress.2011.06.014

In this paper we focused on the Fault Tree analysis because nowadays it is the most used quantitative technique for accident scenario assessment in the industry. The aim of this paper is to review briefly the improvements of the DFT over the SFT and provide a useful scheme to approach the resolution of a quite general class of DFT that includes nested dynamic gates, events with generalized distributed time to failure and MOE [40] (also known as repeated events). Intentionally, we will not cover other approaches (i.e. SPN, SAN, BDMP, etc.) because they are too general [5,6] and their use requires notions that go over the capability covered by the DFT approach.

A significant part of this work is devoted to reason about the hierarchical approach for DFT [15,16,18,19]. The concepts of weak and strong hierarchy are introduced and used to estimate what approximations arise when DFT with nested dynamic gates are analyzed.

This paper is organized as follows: in the first part we present an overview of the fault tree analysis, introducing the SFT of the presented case of study and its enhanced model by the mean of the DFT technique. In the second part, the most common analytical techniques of resolution are discussed, in particular the state-space models and an adapted modular approach for general DFTs. The aim of this section is to provide a reference framework to analyze a generic DFT, what techniques apply and what software uses (or combine) to obtain reasonable results.

In the final section, the case of study is solved in several manners, according to the scheme of resolution suggested. Among the traditional analytical tools, a novel simulative approach—developed under a well-known commercial spreadsheet [44]—is used as a benchmark to compare the final results. In the end conclusions are drawn and future works are indicated.

2. Research framework

The study is developed with reference to the FT model of Fig. 1: an accident scenario in an alkylation plant, as it is reported in the Safety Report required by the Seveso Directive. The SFT was designed by the experts according to the HAZOP report. Although SFTs are very common in the industrial field, DFTs are desirable because some reliability schemes and the integration with real time technology of monitoring (like the DCS [20]) introduce temporal dependencies that the static models are unable to treat.

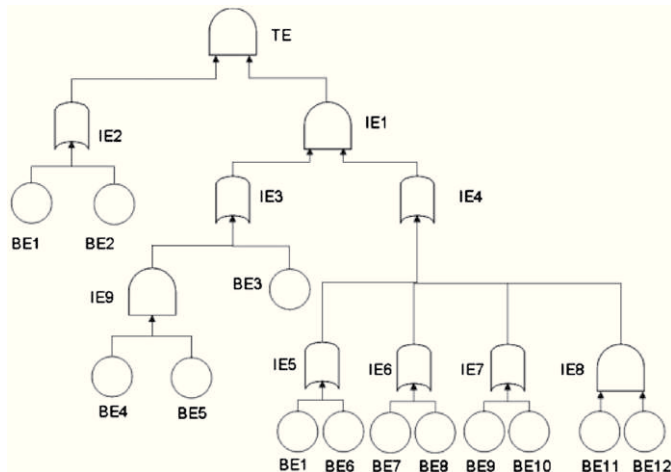


Fig. 1. SFT of a real industrial plant (alkylation plant).

2.1. Static Fault Tree (SFT)

The TE of a SFT [21] is described through the well-known structure function:

$$\phi(t) = f(\underline{X}(t)) = \begin{cases} 1, & \text{if the system is working} \\ 0, & \text{if the system is failed} \end{cases} \quad (1)$$

where $\underline{X}(t) = [X_1(t), X_2(t), \dots, X_n(t)]$ is the vector of the states of the system and $X_i(t)$ represents the i th component that can be in a working or in a failed condition. Several methods of resolution exist and their usability depends on the complexity of the tree. In fact, a simple model without repeated events can be solved with the equivalent RBD [22]. Nevertheless, in the industrial applications it is usual to deal with large SFT composed by BEs characterized by a very low probability of occurrence. In these cases, exact methods such as factorization [23] or BDD [24] can be unfeasible; therefore the MCS technique is combined with the rare event approximation renouncing to exact results. The choice of what is the optimal truncation limit is discussed in many regulatory guides of PRA and it has been the objective of further elaborations through a technique that considers the important measures and the sensitivity of the CDF [25]; however, there is no certainty about the accuracy that can be reached and this can cause the underestimation (or the overestimation) of the sources of risk [26] and consequently can invalidate the safety or optimization strategies, which are based on these evaluations. The SFT models are constrained to the following assumptions [27]:

- binary nature of the components, which can only be in the operative or in the failure state;
- BEs are independent;
- transition between the working and the failed state is instantaneous;
- maintenance restores components as good as new and
- if the failure of a component influences other events on superior levels, its repair restores these events to the normal operative condition.

The algorithms for the resolution of the SFT are easy to implement because they make use of the Boolean algebra.

2.2. Dynamic Fault Tree (DFT) and analytical resolution

State-space models have been used to overcome the limits of the SFT, but

- unlike the FT, they are not systemic oriented;
- construction of the schema can become difficult and error prone;
- readability of the model is less intuitive than the combinatorial representation and
- complexity of the model can make the analytical resolution hard (or even unfeasible).

DFT methodology is a technique for the reliability assessment that was born to overcome the state-space complications but keeps the powerful representation of the SFT. In fact, the structure function of these models is time dependent since the dynamic gates (Fig. 2) establish interactions among the components (FDEP, PAND) and modify their failure attitude (SPARE, SEQ) [4], but the resolution of a DFT is not as simple as in the SFT because it cannot be performed with the rules of the Boolean algebra.

After a careful review of the most important literature about DFT models, we have realized the need to list and discuss the

Name	Graphical Representation	Description (N input)
SPARE		It triggers only after the primary if all the N spares occur. Spares can be shared with other spare gate.
PAND		It behaves like an AND gate but it triggers only if the input events occur in the order from the left to the right.
SEQ		It forces the input events to occur from the left to the right order. It can model the gradual degradation of a system.
FDEP		This gate models the failure of the dependent input events if the primary occurs. The output is a dummy.

Fig. 2. Most frequently used dynamic gates.

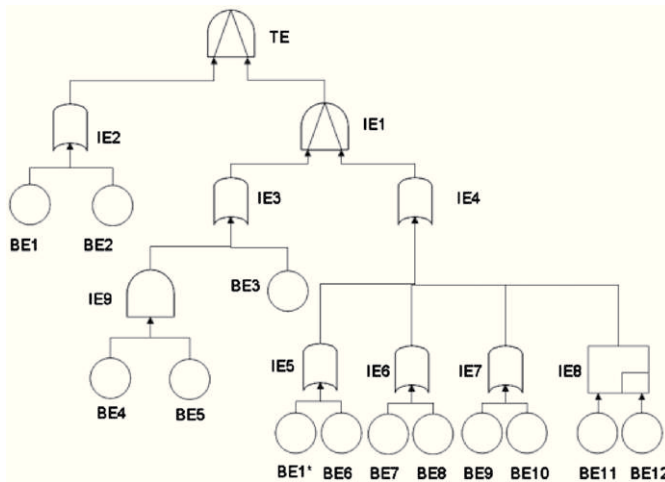


Fig. 3. DFT of the real industrial plant in Fig. 1.

method proposed in these previous works, as it came out that an established procedure for the resolution of a DFT does not exist.

Fig. 3 shows the dynamic version of the static schema of Fig. 1. The DFT of Fig. 3 was obtained, implementing the following important rearrangements that consider the real behavior of the safety systems of the alkylation plant: the static AND gates (IE1 and TE) were replaced by two PAND gates (as the alarm equipment considers the time of occurrence of a fault) and the IE8 with a SPARE in order to model the cold stand-by between two pumps.

In Fig. 4 we present a breakdown for the classification of a fault tree. The procedure for the resolution of the fault tree is chosen according to the following main characteristics: the distribution function of the events time to fail, the presence of repeated BEs and the type of tree.

If the BEs of the DFT are characterized by an exponential distributed time to fail, the most used domain of the resolution of a DFT is the HCTMC (or CTMC). The memory-less property (which characterizes the HCTMC) makes the resolution of the problem straightforward. In fact, the mapping of a DFT into a Markov chain can be performed with the direct mapping, recursively constructing the matrix of the infinitesimal generator Q . Hence, the closed

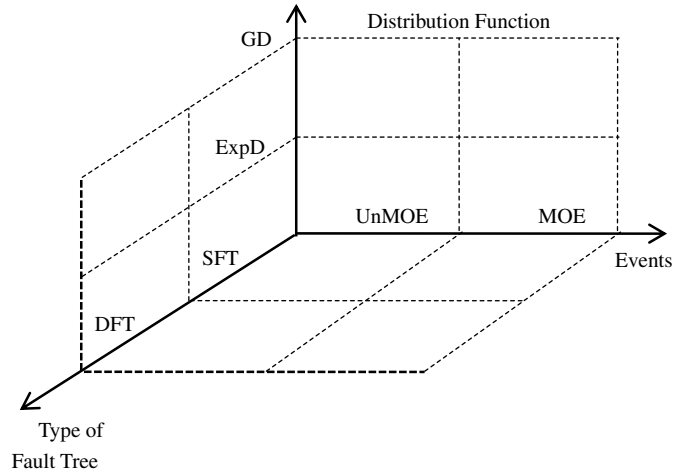


Fig. 4. FT breakdown structure.

analytical solution for the instantaneous reliability is obtained by solving the set of differential equations [28]:

$$\frac{dP(t)}{dt} = P(t)Q \tag{2}$$

where $P(t)$ is the vector of the state probabilities and $P(0)$ the vector of the initial probabilities.

The main disadvantages of this approach is the constraint to use only the exponential distribution and the state-space explosion that can affect even small DFTs. In fact, if we assume a DFT with n basic events, the matrix Q can be larger than $2^n \times 2^n$.

If the state-space model is available, there are techniques that can be used to reduce its largeness and improve the computation performance: in [29] stochastic algebra it is used to find bisimilarity among the states and favor the lumping, while in [30] a simple technique is shown for the aggregation of the states that permits to find the equivalent transition rates also for repairable systems.

In [34,35] DFTs are solved by the conversion into a Bayesian network, a formalism based on the explicit dependencies among the gates and the BEs of the fault tree. This method mitigates the state-space explosion as it creates a DAG with a number of nodes given by the sum of the number of the BEs (that are the leaves of the DAG) and the gates.

In [37] an elegant solution, based on a temporal Boolean logic, allows the resolution of a cascade of PAND gates for the reliability computation of DFT with repeated events. However, this is just a small set of all the classes of DFT that we are considering.

Even though the previous techniques are quite dynamic, industrial applications make use of repairable components, which are not described only by the exponential distribution; therefore HCTMCs and Bayesian network are too limited. Non-homogeneous CTMCs (NHCTMCs) are more powerful but only simple models can be analytically treated with MRGPs and GSMPs [31]. The continuous phase-type distribution [32] is an approximated solution for solving the GSMP: a generalized distribution function can be approximated by the use of one or more interrelated Poisson processes that occur in sequence. But its application is not feasible even for ordinary cases because a Markov chain that approximates a single generalized distribution function of a BE can be too large; with the increasing of the model the final representation of the GSMP can suffer from the state-space explosion problem [33] (Table 1).

In Table 2 a synthesis of the main features and limits of the analytical techniques is presented.

2.3. Mitigation technique of large DFT: weak and strong hierarchical approach

The issue of the state-space explosion can be mitigated with the hierarchical approach [15,16,18,19]. This method (also known as “modularization”) was developed to reduce large combinatorial models.

The hierarchical technique is performed recursively in two phases:

1. the decomposition phases, ‘Phase 1’, detect the independent parts (sub-models) of the original FT that can be solved a part (i.e. repeated events cannot be split into different sub-modules) and
2. the composition phases, ‘Phase 2’, perform the aggregation of the sub-models which are substituted with an equivalent BE.

At the end of this recursive process, the original FT is collapsed in a smaller but equivalent FT*, which can be solved with the methods of the Boolean algebra.

Hierarchy can be applied in a similar fashion for DFT (Fig. 5): the aim is to deal with a simpler aggregated DFT (in the following indicated as DFT*) in order to mitigate the state-space explosion of the equivalent CTMC.

In the following we will refer to

- *composed sub-models* to indicate the parts of the DFT, which are solved as a part during the decomposition phase and
- *hierarchical model* to indicate the final representation of the aggregated DFT*, built with the previous composed sub-models.

Unfortunately, the dependencies of a DFT model are also caused by the temporal interactions among different BEs through the dynamic gates. Therefore, modularization can become less

Table 1
Parameters of the BEs for the FT-A (λ =failure rate [1/h]; Q=constant) probability of failure.

ID	Description	λ [1/h]	Q
BE1	Human error	-	1.0×10^{-3}
BE2	Breakage HV72 failure	9.1×10^{-4}	-
BE3	No operative intervention	-	1.0×10^{-3}
BE4	LAHH78 failure	1.7×10^{-4}	-
BE5	LAHH failure	7.5×10^{-4}	-
BE6	HV75 failure	9.1×10^{-4}	-
BE7	Flow level control failure	4.5×10^{-3}	-
BE8	FV72 failure	8.6×10^{-4}	-
BE9	Level control system failure	4.5×10^{-4}	-
BE10	LAHH78 failure	7.9×10^{-3}	-
BE11	Pump G17 failure	1.5×10^{-4}	-
BE12	Pump G17S failure	9.5×10^{-4}	-

effective. According to past literature [15,16], we focused on two kinds of hierarchical approaches that we will refer to as

1. *strong approach*: it synthesizes a DFT*, which keeps the temporal dependencies of the original DFT, providing exact results and
2. *weak approach*: it disregards the temporal dependencies and provides an approximated DFT*.

Fig. 6 shows a class of DFT that inspired many reliability models [15,18,19].

In this kind of DFT, dynamic modules are solved as a monolithic block; the white blocks can be thought as the intermediate layers of the FT and can contain only static gates; gray blocks are the lowest levels for which no more decompositions are performed (sub-models of SFT and DFT are completely reduced and solved). For these cases, the strong hierarchy is enabled. In fact, unmanageable temporal dependencies do not arise because these are all treated internally in the sub-models. The final DFT* is solved with the techniques of the SFT, getting an exact result.

The use of the hierarchy inside a pure dynamic sub-module was suggested in [16]: we have noticed that it can turn into a strong or weak approach, depending on the structure of the sub-models, as in Fig. 7, where two similar DFTs are shown. Through the modularization of the sub-models (inside the circle), the original DFTs are converted into an equivalent

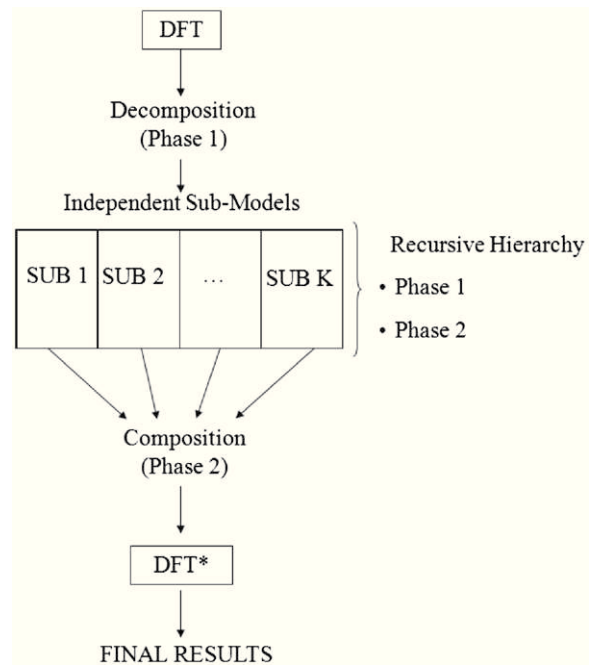


Fig. 5. Recursive two-phase procedure of the hierarchical technique.

Table 2
Synthesis of the features and limits of the main analytical stochastic models.

Stochastic model	Hypotheses	Mitigation technique	Overall limits
HCTMC	Exponential distribution	Stochastic algebra Lumping Hierarchy	State-space explosion Ineffective modeling of real industrial systems
GSMP/MRGP Bayesian networks	Non-exponential distributions Reliability computation	Continuous phase-type approximation	Complex models are not solvable No cyclic dependencies

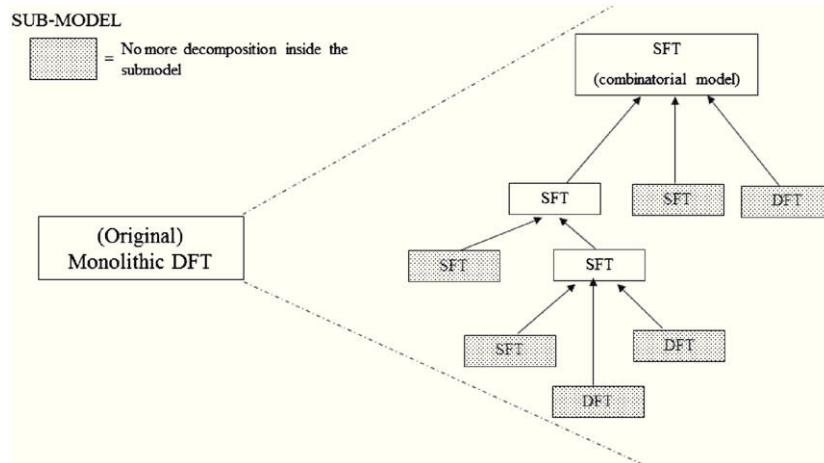


Fig. 6. Hierarchical decomposition of a DFT, which retrieves exact results.

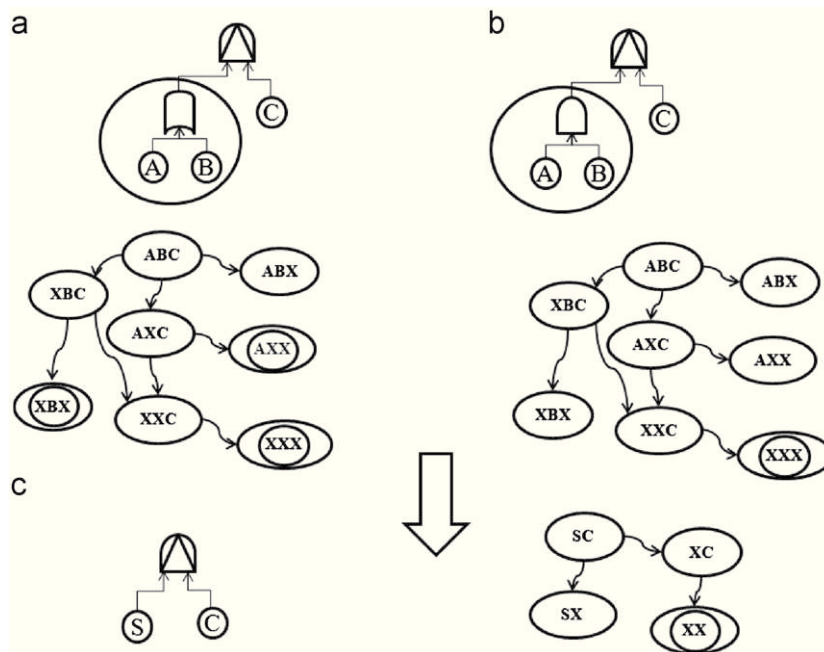


Fig. 7. Two different DFTs (a) and (b) with the equivalent hierarchical model (c). Double circles are the state of failure for the system. DFT (a) enables the strong hierarchy and DFT (b) the weak one.

DFT* (Fig. 7c): the CDFs that describe the composed sub-models 'S' can be a GD.

In particular, the model of Fig. 7a can enable a straight strong hierarchy as the CDF of the composed OR gate is still an exponential distribution with $\lambda_{eq} = \lambda_S = \lambda_A + \lambda_B$.

In the case of Fig. 7b, the CDF of the AND gate is an exponential; hence the hierarchical DFT* is no longer equivalent to a CTMC.

In this last case, the use of a weak hierarchy can be applied to get back a CTMC, finding an equivalent constant failure rate, $h(t) = \lambda_{eq}$, which synthesizes the dynamic of the gate. Two approaches can be followed:

W1. the computation of an instantaneous equivalent failure rate [16], from the reverse law of the exponential distribution, that depends on the precise instant of time T^* (where T^* is the mission time of the original model);

W2. the second weak approach exploits the property of the exponential distribution by the use of the inverse MTTF as the equivalent failure rate of the combined sub-model. In this case:

$$\lambda_{eq} = \frac{1}{MTTF_{AND}} = \frac{\lambda_A^2 \lambda_B + \lambda_B^2 \lambda_A}{\lambda_A^2 + \lambda_B^2 + \lambda_A \lambda_B} \quad (3)$$

We have quantified the quality of the approximation for the example of Fig. 7 (Table 3), assuming all the BEs to have the same failure rate: low values of the parameter λT_m (i.e. 10^{-2} , 10^{-3}) correspond to higher relative errors. Nevertheless, for high reliable systems these approximations are more tolerable than the ones associated to larger λt . These approximations can grow according to the complexity of the sub-model: final results may not be significant in terms of both logical modeling and numerical evaluations.

In conclusion, the hierarchical approach results are exact (or strong) in the following cases:

1. DFT is structured like in Fig. 6;
2. under any dynamic gate if the modularization involves OR gate sub-modules without repeated events.

In all the other cases, the hierarchical approach will be feasible in terms of weak hierarchy, providing an approximated DFT, which keeps the advantage to be equivalent to a CTMC.

2.4. Software tools analysis and application of the hierarchical techniques

Automated tools can be classified (and used) according to the way they solve a DFT, with respect to the engine used (analytical or simulative), the domain of resolution (CTMC, Bayesian Space, BDMP, SPN, etc.), the hierarchical algorithms and the measures retrieved.

The software tools reported in Table 4 have been tested. Some considerations can be important:

1. TOOL_1 [42]: it provides an intuitive high level framework to model a DFT. The engine of the TOOL_1 can retrieve the reliability and the availability of the system and it can deal with repeated events; only exponential distributions are permitted and the hierarchy has to be handled manually (other kind of reliability models can be assembled such as RBD, FT and CTMC).
2. TOOL_2 [13]: this software provides a user friendly graphic interface to construct a DFT. It can compute only the reliability of a system; repeated events are not permitted but, unlike TOOL_1, it can deal also with the Weibull and lognormal distributions through a simulative engine. Modularization is handled automatically but only the exact (strong) hierarchy is performed. Moreover, the algorithm is not optimized as it cannot solve simple DFT as in [34].
3. TOOL_3 [43] is more than a reliability tool. It offers a practical interface to solve SFT, Reliability Graph, RBD and HCTMC as well as GSMP and MRGP. DFTs are not implemented; therefore

dynamic models have to be solved constructing manually the equivalent state-space model. Hierarchy is allowed and reliability, availability and other importance measures can be automatically retrieved.

For the models of Fig. 6, TOOL_1 and TOOL_2 can process the DFT as no nested PAND gates appear; instead, TOOL_3 needs to solve the dynamic models a part—through the equivalent CTMC—and embeds these results into the hierarchical SFT model. But, if dynamic gates are nested along the structure of the tree the previous software can get stuck due to the state-space explosion.

The solution that we want to suggest in order to mitigate the state space is based on an adapted combination of strong and weak hierarchy inside any dynamic sub-model of the original DFT. In this way the original DFT can be drastically reduced. For instance, with the strong hierarchical approach the hypothetical example of [35] is reduced from the initial 16 BEs to 10. In this configuration TOOL_2 can easily solve the model. Applying another weak hierarchy for the AND gate, the DFT is finally reduced to three BEs and also TOOL_1 can process the computation with an error of 5.5×10^{-10} .

Among the proposed reliability software only TOOL_3 can deal with GD once the DFT* is completely represented in terms of the state-space model. However, it can solve only particular subclasses of non-Markovian models; therefore a generalization is not easily feasible. In fact, GDs can be used only in the initial state of the state-space model. The example of Fig. 8 explains this statement: a 2-input PAND gate is modeled according to the two possible permutations of the input of the gate (Fig. 8a and b). Let us also assume that one basic event, BE_G , is characterized by a

Table 3 Analytic unreliability for the DFT of Fig. 7b and for its weak hierarchical model (Fig. 7c).

λT_m	Unreliability		
	Analytic	W1	W2
10	3.33×10^{-1}	5.51×10^{-1}	3.99×10^{-1}
1	8.39×10^{-2}	1.98×10^{-1}	1.45×10^{-2}
10^{-1}	2.87×10^{-4}	4.52×10^{-3}	3.05×10^{-3}
10^{-2}	3.28×10^{-7}	4.95×10^{-5}	3.31×10^{-5}
10^{-3}	3.31×10^{-10}	4.94×10^{-9}	3.33×10^{-7}

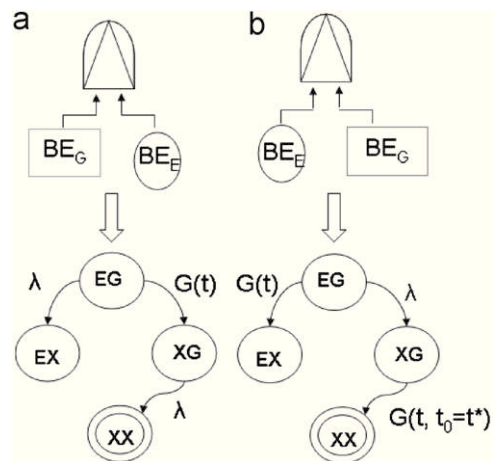


Fig. 8. (a) GSMP for a 2-input PAND gate; (b) GSMP for a 2-input PAND gate with the permutation of the input of (a).

Table 4 Main features of the reliability automated tools for DFT.

Tool	Dynamic modeling	Hierarchy	Measures computed
TOOL_1 [42]	DFT WIZARD HTMC	Manual	Reliability/availability importance measures Other measures of interest
TOOL_2 [13]	DFT WIZARD	Automated	Reliability (no repeated events) Sensitivity Other measure of interest
TOOL_3 [43]	HTMC GSMP MRGP SPN	Manual	Reliability/availability Importance measures Other measure of interest Sensitivity

generalized CDF, $G(t)$, and the other basic event, BE_E , has time to fail exponentially distributed with parameter λ :

1. in Fig. 8a the BE_E is the first input of the PAND and in that way TOOL_3 is able to perform the “competing process” [31] from the initial state of the GSMP and
2. in the second modeling (Fig. 8b) the BE_E is put as the second input of the PAND. Under this condition, the sojourn time t^* of the initial state “EG” is not determined and the CDF of the state “XG”, $G(t, t_0=t^*)$ is unknown.

2.5. Simulative approach

In the previous sections we discussed the lack of the analytical approaches, which are generally caused by the state-space explosion of the equivalent CTMC. When the hierarchical approach is not of help and the events of the model are not exponentially distributed, a comparative analysis based on a simulative approach can be performed [36]. Its implementation requires the following steps:

- a) identification of the simulative horizon T_m (time of mission) and of its time step of discretization;
- b) definition of the stochastic behaviors of each BE (nature of failures—described by a distribution function) and the codification of their random sampling;
- c) implementation of the interrelationships among the events, modeled with the static and dynamic gates and
- d) tracking of the results at any iteration and verification of the convergence through the error test.

For our purposes, the simulative environment has been coded with a commercial spreadsheet [44].

With this approach, the only informations needed to compute the reliability of the system are the time to fail and the state of each component (inputs and gates). Therefore, for a DFT with n inputs and k gates, these data can be stored in a structure of $n \times 2k$ elements.

The main advantage of the spreadsheet environment is that the logic relationships of the simulative engine can be implemented quickly with its standard functions, realizing a meta-structured language that is universally used and natively integrated within many office suites of the same type.

The logic of any gate can be implemented in a master cell (i.e. n-PAND, C-SPARE, W/H-SPARE, FDEP, etc.) and, thanks to the “copy and paste” function, copied at need into other cells in order to build up the DFT. This characteristic is valuable because it allows the implementation of new logics and can simplify the evaluation of large DFT, no matter the number of repeated events.

In fact, for the inputs of the tree (the BEs) any kind of stochastic failure behavior is allowed; the sampling process of the CDF can be performed with reference to any generic law (for example with a statistical inference of experimental data), as shown in Table 5, through the RAND function (implemented with the algorithm of [38] and tested in [39]).

Table 5
Simulative code with the standard functions of the spreadsheet.

GD	F(TBE _i)	TBE _i =F(TBE _i) ⁻¹
Exponential	+RAND()	-LN(1-RAND())/TBE _{im}
Gaussian	+RAND()	+NORMINV(RAND();TBE _{im} ;σ _{TBE_i})
Constant	+RAND()	+RAND()*TBE _{imax}

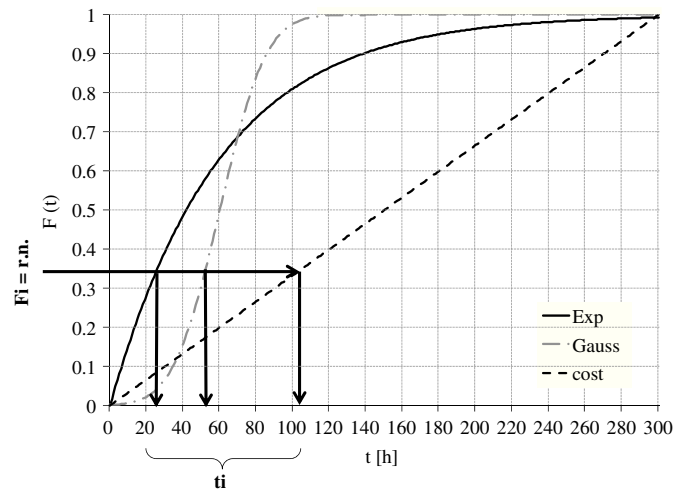


Fig. 9. Discrete event simulation engine that determines the time of failure according to the CDF of the BE.

Fig. 9 explains the discrete event approach: a random number in [0,1] is extracted and used as the value of the CDF (the codomain). The inverse function (as in Table 5) retrieves the time of failure for each BE, realizing the engine of the simulating process.

For what concerns the logic of the gates, Fig. 10 shows the relationships to put into the master cells for any dynamic 2-input gate (the SEQ works exactly as a C-SPARE gate) and the model of two C/W/H-SPARE gates with only one shared spare component. The contribution of any FDEP gate can be considered preparing the spreadsheet in a way that all the BEs that depend on a FDEP gate are updated according with the time of the correspondent failure of the trigger event. The relation is very simple as it corresponds to the static OR logic. This way to code a model is based on the Wysiwyg property of the spreadsheet, which favors the knowledge sharing among the end users. In fact, it allows to look inside any cell and understand what is the logic implemented in the fault tree.

In the schema of Fig. 10 the algorithm wants to compute the time to fail of the i th component (T_i) for all the BEs of the DFT, which are used for the other conditional statements like the state of the components, the time of triggering of the gates and of activation of the dependent logics (see the SPARE).

In point of fact, the scalability and the flexibility to develop any kind of logic with the conditional statements, the primitive functions and all the amount of cells of the spreadsheet environment can turn into an unreadable and bad structured spreadsheet. In this sense, it would be worth understanding the limits of the dependent behaviors to implement and the number of components to treat in one sheet. For instance, as it is shown in Fig. 10, the dependency between two spare gates that share one spare component can be tackled with two additional conditions. These conditions may easily become larger as the number of spare components increases.

Another interesting characteristic is that with this environment RAMS analysis is not limited only to the reliability computation since cells are “almost” unlimited and data (also from other sheets) can be linked with few clicks. For instance, the “failure criticality index” [41] of each component can be retrieved counting the number of system failures that occur in $(0, T_m]$ due to the failure of the same component.

In a similar way it is possible to attach to additional cells other kind of data like real time measures from the field [20] and retrieve evaluations about the current state of the system.

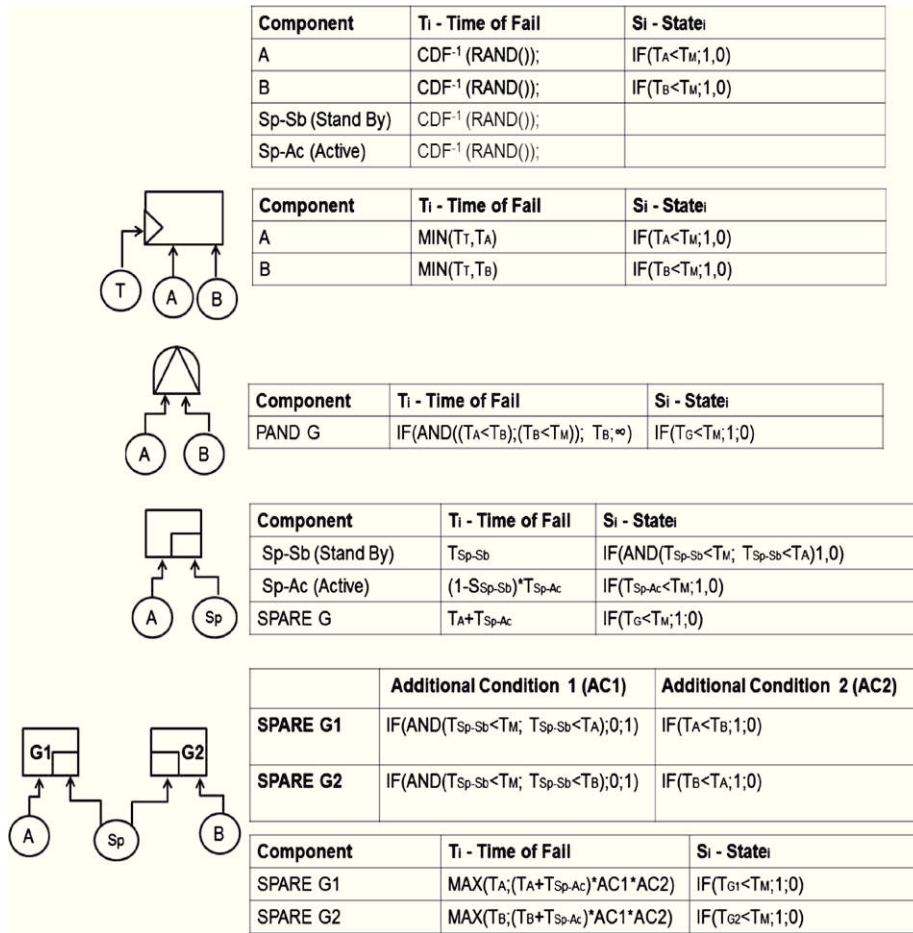


Fig. 10. Implementation of the dynamic 2-input gates under the spreadsheet environment of [44]. CDF^{-1} is not a function of the spreadsheet and represents the inverse function of the CDF of the generic BE.

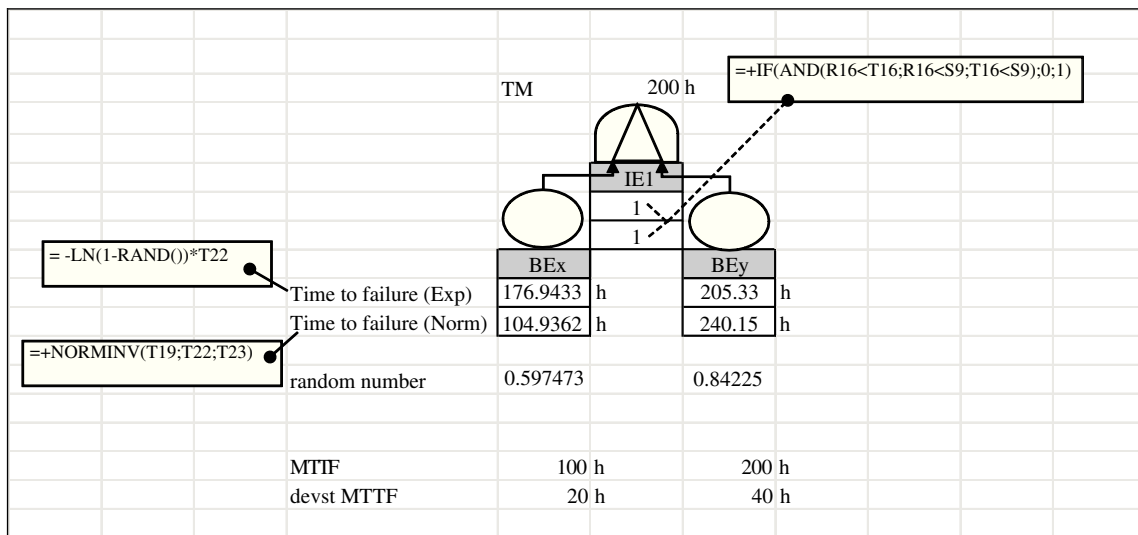


Fig. 11. Implementation of a 2-input PAND under the spreadsheet environment.

For instance, Fig. 11 shows a simple model of PAND built from the master cell and the computation of the MTTF attached to any BE.

The model of Fig. 7b was developed and solved quickly: in this case errors are negligible with respect to the weak hierarchy (Table 6).

Table 7 shows the results collected for the case of study of another complex model, a multiprocessor system described in [34]. This case of study was developed with the master cells of Fig. 10. Results prove the effectiveness of the spreadsheet.

Table 6

Analytic unreliability for the DFT of Fig. 7(b) and for the simulating model.

λT_m	Analytic unreliability	Simulated unreliability
10	3.33×10^{-1}	3.35×10^{-1}
1	8.39×10^{-2}	8.38×10^{-2}
0.1	2.87×10^{-4}	2.96×10^{-4}
0.01	3.28×10^{-7}	3.31×10^{-7}
0.001	3.31×10^{-10}	3.37×10^{-10}

Table 7

Simulated unreliability for the DFT of a multiprocessor system [34].

Mission time T_m [h]	Unreliability (10^4 iterations)	Unreliability (10^5 iterations)	Unreliability (10^6 iterations)
1000	5.40×10^{-3}	5.90×10^{-3}	6.00×10^{-3}
2000	1.10×10^{-2}	1.26×10^{-2}	1.23×10^{-2}
3000	1.88×10^{-2}	1.91×10^{-2}	1.91×10^{-2}
4000	2.71×10^{-2}	2.73×10^{-2}	2.73×10^{-2}
5000	3.42×10^{-2}	3.74×10^{-2}	3.72×10^{-2}

3. Case of study solution

The simulations were executed with a standard laptop with the following technical characteristics: a dual processor of 1.86 GHz and 2 GB of Ram.

According to the statistic parameters of Table 1, the DFT of the case study (Fig. 3) would be unfeasible with the previously described analytical techniques as it contains

- constant probabilities (BE1 and BE3) that invalidate the use of the CTMC and
- a repeated event (BE1) at the lowest branch of the DFT that does not verify the assumption of the hierarchical approach.

In order to test all the approaches reviewed in the paper, several scenarios were designed as follows according to the breakdown of Fig. 4:

1. **Test 1:** all the BEs are described by exponential distributions.
2. **Test 2:** this test represents the original input configuration of Table 1 according to the industrial application.
3. **Test 3:** BE1 and BE3 are described by exponential distributions (like in Test 1) and all the other BEs are characterized by a Weibull distribution with a scale parameter equal to the failure rate λ_i (of the respective BEi) and a shape equals to 3.

The previous scenarios were evaluated twice to consider the FT with (MOE-FT) and without (UnMOE-FT) the repeated event.

The results are shown in Tables 8 and 9, respectively, for the static and the dynamic FT: not feasible computations (n.f.) happen when the tool is not able to perform the resolution of the model, not performed (n.p.) are the computations that would need more investigations and no results (–) when the computation got stuck.

For what concerns the results in terms of reliability, it is possible to notice that in both the STF and the DFT the repeated BE1 does not contribute significantly to the TE occurrence.

For the monolithic UnMOE-FT, TOOL_1 got stuck while TOOL_2 and the simulator retrieved comparable results: the former was computed in about 30 min, and the latter—with a number of 10^8 iterations—took about 9 h.

The computations with the strong and weak hierarchical approaches (possible only for Test 1) simplified the model; hence both TOOL_1 and TOOL_2 were able to perform the computation very quickly, retrieving the same results.

Table 8

Static unreliability model results.

FT type	Software application	SFT unreliability		
		Test 1	Test 2	Test 3
MOE-FT (BE1=BE1*)	TOOL_1	7.73×10^{-1}	7.73×10^{-1}	9.48×10^{-1}
	TOOL_2	n.f.	n.f.	n.f.
	TOOL_3	7.73×10^{-1}	7.73×10^{-1}	9.48×10^{-1}
	Simulative approach	7.73×10^{-1}	7.73×10^{-1}	9.42×10^{-1}
UnMOE-FT (BE1 ≠ BE1*)	TOOL_1	7.73×10^{-1}	7.73×10^{-1}	9.63×10^{-1}
	TOOL_2	7.73×10^{-1}	7.73×10^{-1}	9.63×10^{-1}
	TOOL_3	7.73×10^{-1}	7.73×10^{-1}	9.63×10^{-1}
	Simulative approach	7.73×10^{-1}	7.73×10^{-1}	9.50×10^{-1}

Table 9

Dynamic unreliability model results.

FT type	Software application	DFT unreliability		
		Test 1	Test 2	Test 3
MOE-FT (BE1=BE1*)	TOOL_1	–	n.f.	n.f.
	TOOL_2	n.f.	n.f.	n.f.
	TOOL_3	n.f.	n.f.	n.f.
	Simulative approach	5.35×10^{-5}	5.49×10^{-5}	$< 10^{-7}$
UnMOE-FT (BE1 ≠ BE1*)	TOOL_1	–	n.f.	n.f.
	Monolithic			
	TOOL_1 Strong Hierarchy	5.32×10^{-5}	n.f.	n.f.
	TOOL_1 Weak Hierarchy	4.47×10^{-4}	n.f.	n.f.
	TOOL_2	5.31×10^{-5}	n.f.	7.93×10^{-9}
	Monolithic			
	TOOL_2 Strong Hierarchy	5.32×10^{-5}	n.f.	n.p.
	TOOL_2 Weak Hierarchy	4.47×10^{-4}	n.f.	n.p.
	TOOL_3	n.f.	n.f.	n.f.
	Simulative Approach	5.45×10^{-5}	5.07×10^{-5}	$< 6.00 \times 10^{-8}$

In order to use the strong (Fig. 12a) and the weak hierarchy (Fig. 12b) few other simplifications were needed. In fact, the gate IE8 was elided (its contribution to the failure of the IE4 is negligible) and the BE3 was cut off (its failure rate is much lower than the equivalent failure rate of the gate IE9). In this way, the original OR gate IE4 was replaced with a strong equivalence characterized by a failure rate computed as the sum of the failure rates as follows:

$$\lambda_{IE4} = \lambda_{BE1*} + \lambda_{BE6} + \lambda_{BE7} + \lambda_{BE8} + \lambda_{BE9} + \lambda_{BE10} = 2.235 \times 10^{-2} [1/h]$$

For the weak configuration of Fig. 11b, another composition was added through the weak equivalence of the AND gate IE9:

$$\lambda_{IE3/IE9} = \frac{1}{MTTF_{IE9}} = \frac{\lambda_{BE4}^2 \lambda_{BE5} + \lambda_{BE5}^2 \lambda_{BE4}}{\lambda_{BE4}^2 + \lambda_{BE5}^2 + \lambda_{BE4} \lambda_{BE5}} = 1.6316 \times 10^{-4} [1/h]$$

As shown in Table 9, the strong hierarchy retrieved results that are very close to the ones obtained with the monolithic model. The weak hierarchy, instead, introduced an error of two orders of magnitude, not far from the one computed for the same DFT structure (Fig. 7c) with respect to the parameter λT_m (Table 6).

Test 2 and Test 3 were solved only with the simulative approach and it is interesting to notice how the Weibull

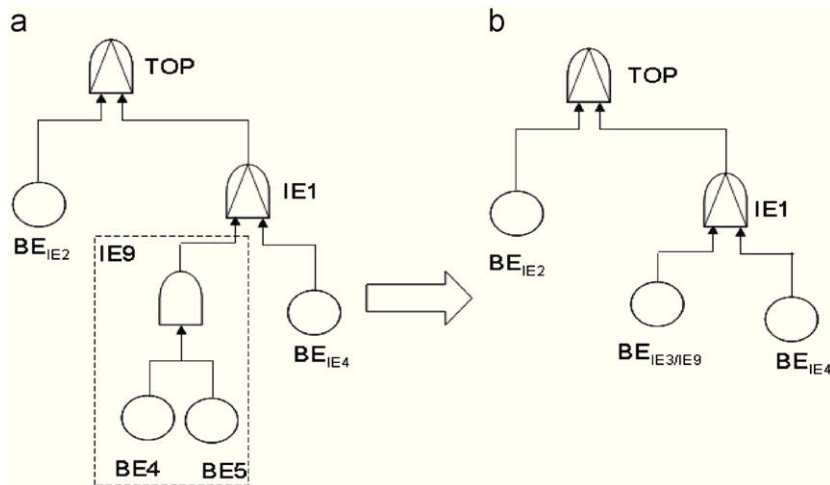


Fig. 12. (a) Strong and (b) weak hierarchy representation for the model of Fig. 3.

Table 10
Resolution tools for each kind of DFT model.

MOE-DFT	UnMOE-DFT
ExpD	
Simulative application	TOOL_2 Simulative application TOOL_1 (if strong hierarchy is permitted)
GD	
Simulative application	Simulative application

distributions affect in a very different way the behavior of the SFT and the DFT. In fact, for the former model the TE occurrence increases whereas for DFT it is drastically reduced to values (10^{-8} – 10^{-9} order of magnitude), which can be considered negligible for safety purposes. This last result offers an important cue about the meaning of the risk assessment when the components of these models have a time to failure characterized by GDs.

In Table 10 the resolution tools of this study are classified according to the type of DFT that they can resolve.

In the appendix, the results of the Monte Carlo simulating processes for the DFT are shown.

4. Conclusions and future works

This study has been motivated by the need to improve the quality and the performance of the risk assessment of industrial plant, with the aim to find an environment for the reliability assessment with the following requirements:

- easy to use by the actors of the industry;
- able to match the safety logic and dependability schemes of real plant;
- able to provide results with a reasonable time of computation and
- suitable to retrieve on-line risk assessment using data reported in real time from the field.

The choice of the DFT technique as an instrument for the risk assessment met the first and the second requirement, due to the intuitiveness and the power of the modeling approach.

In this paper, the stochastic analytical techniques to solve DFT were discussed with reference to the presence of repeated events

and generalized probability distribution function, enlarging the usual domain of application of DFT models often restricted within the Markov domain.

Therefore, the first result highlighted a breakdown for the classification of industrial FT; the scheme was used to study the suitability of three of the most known automated tools for DFT models [13,42,43] and classify these software applications according to their capabilities and limits.

An adapted hierarchical technique for DFT, based on an exact (strong) and approximated (weak) approach, was experimented in order to improve the performances of the previous tools. This offered the clue for a second result that revealed what kind of approximations are carried by the weak approach (that works in the scope of the continuous time Markov chain) and what class of DFT the strong hierarchy turns in semi-Markov processes.

After these results, we could claim that the power of the DFT is not exploited fully because no precise procedures of resolution exist and the ordinary reliability tools for DFT are neither satisfactory nor easy to handle.

In this context, we suggested the possibility to develop ad-hoc simulative models to use at least as benchmarks for other analytical evaluations. Therefore, the other contribution of this research was to show how to implement a simulative environment with a commercial spreadsheet [44], as it is a well-known software adopted in many business companies.

The solution seems valuable since the spreadsheet environment carries the following interesting characteristics:

- 1) produced files are easy to distribute and improve the information, sharing among the risk analysts, the managers and employers who are involved in the risk evaluations;
- 2) Wysiwyg property of the spreadsheet allows access simultaneously to the logics and the results of the DFT, favoring the understanding of the process, which is an added value of the risk analysis;
- 3) dynamic model can be customized to build up the most known dynamic gates, design other kind of dependencies logics and implement several other means, like the importance measures and
- 4) it can be integrated with the DCS in force of the plant in order to process data in real time and retrieve fresh information on the real state of the system.

In this work, we showed how to prepare the master cells of the spreadsheet environment, which embed the generalized

distributions for any BE and the logic of the most used dynamic gates of a DFT; afterwards, they were used to construct a complete model, with quite complex dependencies.

Thanks to the framework developed for the case of study, we recognized the existence of singular results when the BEs of the plant are not described by the common exponential distributions.

In our opinion, these results offer the cue for new studies since the characteristics of the components and processes inside the industrial field can be quite general.

Following this path, in future works, we aim to develop a more flexible and faster simulative environment able to merge the intuitiveness of the DFT representation with the power of the simulative approach and able to support data in real time. Finally, we want to emphasize that quantitative risk assessment, based on any technique, is not significant without an accurate identification of hazards and assessment of scenarios and of their root

causes. However, the results provided by quantitative risk assessment are normally used by the supervisory authorities to assess the safety of facilities and therefore it becomes important to provide results that take into account the actual system and its operating modes. In addition, these results may be useful to the plant manager in order to assess the effectiveness of technical solutions that affect the reliability and the safety of the plant.

Appendix

Simulated=unreliability of the DFT (performed with the simulated process);

Analytic=unreliability of the DFT (performed with TOOL_2 and TOOL_1 using the strong hierarchical approach); α =significance level (Figs. A1 and A2).

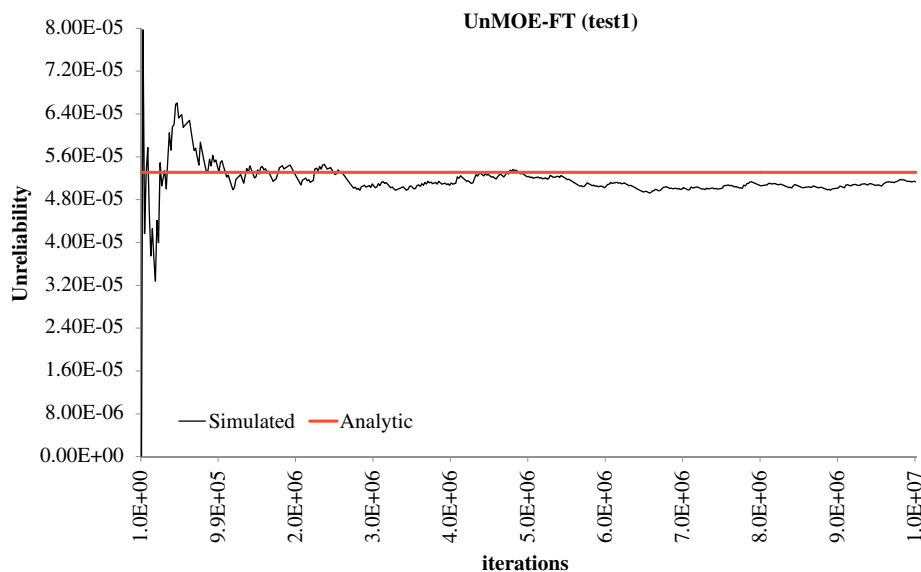


Fig. A1. Comparison between the simulated and the analytic results for the UnMOE-FT of test 1. This is the only scenario that can be solved with the analytical approach ($T_M=8760$ h, $\alpha=0.01$; confidence= 9.62×10^{-10}).

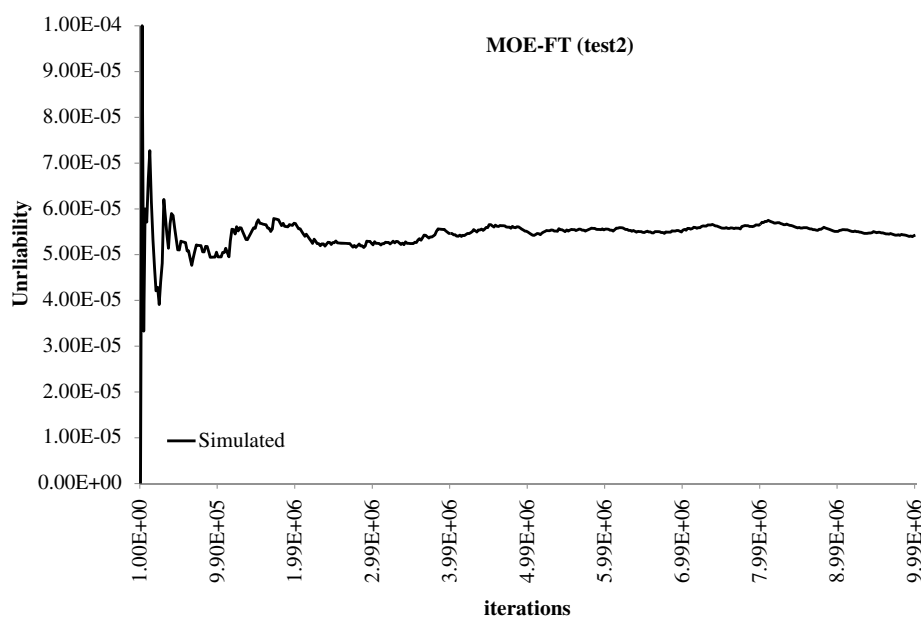


Fig. A2. Simulative process for the MOE-FT of test 2 ($T_M=8760$ h, $\alpha=0.01$; confidence= 5.35×10^{-10}). For test 2, analytic resolution of DFT with fixed probability is not feasible (n.f., see Table 9).

References

- [1] Khan FI, Abbasi SA. Techniques and methodologies for risk analysis in chemical process industries. *Journal of Loss Prevention in the Process industries* 1998;11:261–77.
- [2] Trivedi KS, Sahner RA. Reliability modeling using Sharpe. *IEEE Transactions on Reliability* 1987;R-36:186–92.
- [3] Dugan JB, Bavuso SJ, Boyd MA. Fault Trees and sequence dependencies. In: *Proceedings of the annual reliability and maintainability symposium*; 1990. p. 286–93.
- [4] Boudali H, Crouzen P, Stoelinga M. Dynamic fault tree analysis using input/output interactive Markov chains. In: *Proceedings of the 37th annual IEEE/IFIP international conference on dependable systems and networks, DSN '07*. p. 708–17.
- [5] Bouissou M, Bon JL. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. *Reliability Engineering and System Safety* 2003;82:149–63.
- [6] Bouissou M. A generalization of dynamic fault trees through Boolean logic driven Markov processes (BDMP)[®]. EDF R&D, MRI Department, Clamart, France.
- [7] Cepin M, Mavko B. A dynamic fault tree. *Reliability Engineering and System Safety* 2002;75:83–91.
- [8] Amari S, Dill G, Howald E. A new approach to solve dynamic fault trees. In: *Proceedings of the annual reliability and maintainability symposium*; 2003. p. 374–9.
- [9] Distefano S, Puliafito A. Dynamic reliability block diagrams vs dynamic fault trees. In: *Proceedings of the annual reliability and maintainability symposium, RAMS '07*; 2007. p. 71–6.
- [10] Dugan JB, Bavuso JS, Boyd MA. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability* 1992;41(3):363–77.
- [11] Ciardo G, Muppala JK, Trivedi KS. SPNP: stochastic petri net package. In: *Proceedings of the third international workshop on petri nets and performance models*; 1989. p. 142–51.
- [12] Dugan JB, Venkataraman B, Gulati R. DIFTree: a software package for the analysis of dynamic fault tree models. In: *Proceedings of the annual reliability and maintainability symposium*; 1997. p. 64–70.
- [13] Sullivan KJ, Dugan JB, Coppit D. The Galileo fault tree analysis tool. In: *Proceedings of the 29th annual international symposium on digest of papers fault-tolerant computing*; 1999. p. 232–5.
- [14] Dugan JB, Sullivan KJ. Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transactions on Reliability* 2000;49(1):49–58.
- [15] Gulati R, Dugan JB. A modular approach for analyzing static and dynamic fault trees. In: *Proceedings of the annual reliability and maintainability symposium*; 1997. p. 57–63.
- [16] Anand A, Somani AK. Hierarchical analysis of fault trees with dependencies, using decomposition. In: *Proceedings of the annual reliability and maintainability symposium*; 1998. p. 69–75.
- [17] Montani S, Portinale L, Bobbio A. A tool for automatically translating dynamic fault trees into dynamic Bayesian networks. In: *Varesio M, editor. Proceedings of the RAMS'06*; 2006.
- [18] Sun H, Andrews JD. Identification of independent modules in fault trees which contain dependent basic events. *Reliability Engineering and System Safety* 2004;86:285–96.
- [19] Ou Y, Dugan JB. Modular solution of dynamic multi-phase systems. *IEEE Transactions on Reliability* 2004;53:499–508.
- [20] Compagno L, et al. An on-line fault tree analysis for the continuous monitoring of the industrial plant accidents. Pisa: VGR; 2008. p. 51–60.
- [21] Vesely W. Fault tree handbook. US Nuclear Regulatory Commission; 1981.
- [22] Malhotra M, Trivedi KS. Power-hierarchy of dependability-model types. *IEEE Transactions on Reliability* 1994;43(3):493–502.
- [23] Rauzy A. Mathematical foundations of minimal cutsets. *IEEE Transactions on Reliability* 2001;50(4):389–96.
- [24] Veeraraghavan M, Trivedi KS. An improved algorithm for symbolic reliability analysis. *IEEE Transactions on Reliability* 1991;40(3):347–58.
- [25] Cepin M. Analysis of truncation limit in probabilistic safety assessment. *Reliability Engineering and System Safety* 2005;87:395–403.
- [26] Epstein S, Rauzy A. Can we trust PRA? *Reliability Engineering and System Safety* 2005;88:195–205.
- [27] Xing SAL. *Handbook of performability engineering*. Springer; 2008. p. 595–617 [chapter 38].
- [28] Trivedi KS. *Probability & statistics with reliability, queueing, and computer science applications*. 2nd ed. New York (NY, USA): JohnWiley & Sons; 2002.
- [29] Hermanns H, Mertsiotakis V, Siegle M. TIPPTool: compositional specification and analysis of Markovian performance models. CAV'99, LNCS 1633. Berlin Heidelberg: Springer-Verlag; 1999. p. 487–90.
- [30] Lanus M, Trivedi KS. Hierarchical composition and aggregation of state-based availability and performability models. *IEEE Transactions on Reliability* 2003;52(1):44–52.
- [31] Sahner R, Puliafito A, Trivedi KS. Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package. Kluwer Academic Publishers; 1996.
- [32] Younes HLS, Simmons RG. Solving generalized semi-Markov processes using continuous phase-type distributions. In: *Proceedings of the 19th national conference on artificial intelligence*. San Jose, California; 2004. p. 742–7.
- [33] Kosiuczenko P, Lajos G. Simulation of generalised semi-Markov processes based on graph transformation systems. *Electronic Notes in Theoretical Computer Science* 2007;175:73–86.
- [34] Montani S, et al. Automatically translating dynamic fault trees into dynamic bayesian networks by means of a software tool. In: *Proceedings of the first international conference on availability, reliability and security*; 2006.
- [35] Boudali H, Dugan JB. A discrete-time Bayesian network reliability modeling and analysis framework. *Reliability Engineering and System Safety* 2005;337–49.
- [36] Durga Rao K, et al. Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliability Engineering and System Safety* 2007;94:872–83.
- [37] Merle G, et al. Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. *IEEE Transactions on Reliability* 2010;59(1).
- [38] Wichman BA, Hill ID. Algorithm AS 183: an efficient and portable pseudo-random number generator. *Applied Statistics* 1982;31:188–90.
- [39] Rotz W, et al., A comparison of random number generators used in business. Presented at Joint Statistical Meetings. Atlanta, GA; 2001.
- [40] Ericson CA. What do you do when you run out of computer? In: *Proceedings of the 19th international system safety conference*; 2001.
- [41] Wang W, et al. Reliability importance of components in a complex system In: *Proceedings of the 2004 annual symposium on reliability and maintainability, RAMS*.
- [42] Relx reliability studio reference manual. Relx Software Corporation, Greensburg, PA, USA; 2007.
- [43] Trivedi K. SHARPE interface user's manual version 1.01. Center for Advanced Computing and Communication (CACC), Department of Electrical and Computer Engineering, Duke University; 1999.
- [44] Microsoft Office Excel 2007 <<http://www.microsoft.com/office/excel>>.

MatCarloRe: an integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree

Gabriele Manno*, Ferdinando Chiacchio*

*DMI, Department of Mathematics and Informatics, University of Catania

Abstract

With the aim of a more effective representation of reliability assessment for real industry, in the last years concepts like Dynamic Fault Trees (DFT) have gained the interest of many researchers and engineers (dealing with problems concerning safety management, design and development of new products, decision analysis and project management, maintenance of industrial plant, etc.). With the increased computational power of modern calculators is possible to achieve results with low modeling efforts and calculating time. Supported by the strong mathematical basis of state space models, the DFT technique has increased its popularity. Nevertheless, DFT analysis of real application has been more likely based on a specific case to case resolution procedure that often requires a great effort in terms of modeling by the human operator. Moreover, limitations like the state space explosion for increasing number of components, the constrain of using exponential distribution for all kind of basic events constituting any analyzed system and the ineffectiveness of modularization for DFT which exhibit dynamic gates at top levels without incurring in calculation and methodological errors are faces of these methodologies. In this paper we present a high level modeling framework that exceeds all these limitations, based on Monte Carlo simulation. It makes use of traditional DFT systemic modeling procedure and by replicating the true casual nature of the system can produce relevant results with low effort in term of modeling and computational time. A Simulink library that integrates Monte Carlo and FT methodologies for the calculation of DFT reliability has been developed, revealing new insights about the meaning of spare gates.

Keywords: Reliability Assessment, Dynamic Fault Tree, Monte Carlo Simulation, Continuous Time Markov Chain

1. Introduction

In recent years the importance of risk assessment in the safety context of industrial processes has increased significantly. On the one hand, companies must provide, even more than before, guarantees about the occurrence of significant risks and adopt preventive measures and mitigation of their occurrence, while on the other side, engineers need to predict the reliability of the system from the design phase, especially for critical applications. The systemic reliability representation of the plant process, the quantitative results and sensitivity analysis are straightforwardly obtained using stochastic models such as Reliability Block Diagram (RBD) and Fault Tree Analysis (FTA). These methods have gained wide acceptance for the study of reliability for many kind of plants and systems.

A fault-tree (FT) can be simply described as an analytical technique, whereby an undesirable state of the system is specified; the system is then analysed in the context of its environment and operation in order to find all credible ways in which the undesirable event can occur [1]. Although there are relatively efficient algorithms for solving FTs, the main disadvantage is that dependencies of various kinds, which are in real systems, are not easily captured in the model. Indeed, traditional FT cannot capture the dynamic behaviour of a system such as the sequence of events in time dependence, the replacement of spare parts and priorities of failure events [2, 3, 4, 5].

To overcome these difficulties, dynamic-FT (DFT) was introduced, with the formalization of dynamic gates (PAND, SPARE, SEQ and FDEP). With the help of dynamic gates, the reliability behaviour of systems with time dependencies can be modelled using the DFT technique, keeping the intuitive construction of traditional FT.

DFT resolution cannot rely just on Boolean algebra as well as traditional-FT because dynamic features are not captured with a binary time independent logic. Therefore, beside DFT many other methods have been devised: Dynamic Reliability Block Diagram (DRBD), Continuous Time Markov Chain (CTMC), Input/Output Interactive Markov Chain (I/O IMC), Bayesian Network, Stochastic Petri Network, etc. [6, 7, 8, 9, 10, 11, 12, 13].

The complexity of modern engineering system as well as the need for realistic reliability makes their modelling and analysis a non trivial task. The analytical resolution of systems with dynamic redundancy is very difficult to accomplish. Therefore risk analysts require the use of other techniques in order to construct

a comprehensive but achievable and efficient study of the system. The simulation of DFTs can help to overcome many of the difficulties arising from analytical approaches and can offer a high level modelling interface based on the FT methodology. Scenarios that may be difficult to solve analytically are easily resolved with the approach of Monte Carlo simulation. Due to the intrinsic ability to simulate the actual process and the random behaviour of the system, this approach can eliminate uncertainty of the reliability modelling. It has been used for the availability, reliability and important measure estimation of complex systems [14, 15, 16, 17, 18, 19].

In this paper we present a novel approach to conduct Monte Carlo simulation for the reliability evaluation of DFTs. In our approach Monte Carlo and FT methodologies are integrated in order to allow the user to benefit of an high level *FT-like* interface that exploits the power of Monte Carlo simulation to overcome many of the difficulties arising from the use of analytical resolution methods. A library object, MatCarloRe, based on Simulink was created. Using the graphical interface of the Matlab simulator, it is possible to construct a DFT model of the process by using the library blocks. Each block carries the logic of a DFT gate. The Monte Carlo engine is based on the collection of the outputs of many runs and their agglomerate to construct significant statistics of interest.

The results of a case study based on the probabilistic assessment of the reliability of an alkylation and treatment olefin plant node were evaluated. Three common problems of analytical methods (e.g. CTMC) are discussed: a) the state explosion; b) the impossibility of CTMC to evaluate the reliability of the system when basic events with fixed probability are considered; and c) the impossibility of modularization of the tree in independent sub-trees without incurring in approximation due to a dynamic gate at the TE.

After presenting the validation of the MatCarloRe tool, based on simple cases, it is shown the resolution of the case study in three different configurations: i) static configuration (FT); ii) dynamic configuration (DFT); and iii) dynamic configuration with the introduction of fixed probability basic events. As it is known the first two modelling cases can be compared with analytical values calculated by use of combinatorial and CTMC methodologies, while the last case is solved only through the use of the simulation tool.

Section 2 presents an overview on the main limits of analytical methods; Section 3 makes a comparison between traditional Monte Carlo reliability simulation and our approach; Section 4 introduces in more detail the MatCarloRe tool, its object library, and some validating examples; in Section 5 the results of the case study are presented; and in section 6 conclusions are reported.

2. Analytical reliability evaluation for DFT

A fault tree is a stochastic model for reliability evaluation based on the Boolean algebra. It can synthesize the ways of failures or the undesired events of a system. It is composed of entities known as "gates" which implement the Boolean relationships among the events, leading to the occurrence of higher events up to the tree. The inputs of the gates can be other gates or basic events (BE); the main undesired event is called the top event (TE) of the tree. The methodology is based on three assumptions: (i) the events are binary, (ii) the events are statistically independent, and (iii) the relationships between events are described with the Boolean logic through the gates (AND, OR and Voting).

The main constraints of traditional FTs is that dynamic time dependencies cannot be modelled and complex system behaviour evaluated. DFTs were introduced to overcome these limitations: it is an evolution of traditional FT, in which dynamic gates are added. In particular if a FT includes at least one dynamic gate it becomes a dynamic-FT.

The most likely method used to solve a DFT is through the use of CTMC. They are indeed effective in representing various types of dependencies. The main problem of this methodology is the well known state space explosion which grows exponentially with the increasing of the number of BEs in the model. Moreover, the analysis of large DFT by the mean of CTMC can be costly in computational terms and in terms of modelling efforts.

Modularization is one way to tackle the analysis of large FT. For traditional FT, a module is simply a sub-tree which events do not occur in other parts of the tree. This technique intends to break down large models with a hierarchy of sub-models, characterized by independent failure modes. The sub-models are then analyzed in order to extract the measures of interest which are used as input as a simplified version of the sub-tree at the higher level of the hierarchy. The main point of this technique is the simplification of the equivalent failure rates to use in the simplified parent tree [20, 21, 22]. Dugan et al. [23, 24, 25] have shown that it is possible to identify sub-trees and use several independent Markov chains for each of them. However, this approach has limitations when dynamic gates are present at high levels of the

hierarchy. In fact, the modularization of dynamic sub-trees introduces errors due to the generalized probability distributions that, in the composed model, is still treated as negative exponential distribution [26, 27]. Other ordering methods aim to reduce the complexity of the CTMC by lumping and condensate chain states. However, this requires the construction of the entire chain and the subsequent ranking and aggregation, not solving the tedious problem of modelling the full chain [28, 29, 30].

Another of the limitations resulting from the use of CTMC is the need to use a negative exponential distributions for each basic event considered. This is often simplistic in terms of modelling because events or components are often characterized by other distribution probabilities.

We therefore emphasize that many of the methods to solve the DFT are related to specific problems and can be difficult to generalize for all scenarios.

3. Reliability Monte Carlo simulation for DFT

Simulation programs, especially if well structured, are in general very comprehensible and known for the ease with which modifications and additions can be made. Perhaps their main benefit arises from the fact that it is possible to output information about sub-systems and gain more understanding of the whole system [31]. Monte Carlo simulation is a valuable method which is widely used in solving real problems in many engineering fields. It has been used by [32, 33] for the study of system reliability, based on the well-developed neutron transport ideas. The simulation technique allows the estimation of reliability indices by simulating the actual process and random behaviour of the system through a computer model. This model has to be able the realistic scenario of the system lifetime.

Monte Carlo simulation is implemented by running the model a large number of times, each representing an ensemble of random walks within the discrete phase space of system configurations, in order to generate a large number instances from which all the reliability indices required about the system are retrieved.

As the system is composed of many components (or elements) grouped together to perform a certain function, the system modelling begins with the identification of the components of which the system is composed. Let us denote by s_i the possible states in which the i -th component of the system may be. In the simplest case s_i may have two possible states: one is the "up" or "active and functioning" state; the other is the "down" or "failed" state. The state of the system can be described by a vector S :

$$S = (s_1, s_2, \dots, s_i, \dots, s_n), \quad (1)$$

where n represents the number of components of the system. When a component changes its state, a new point in phase space is reached. Some of this point are of failure for the whole system. While in static-FT certain combination of component failures correspond to system failure, in DFTs the same state can be either a failure state or a good one, depending on the previous state sequence. Hence, in DFT it is not possible to define a system state a good one or a failed one without looking at the system evolution. Vector S changes with respect to time, so each simulation consist of a collection of state vectors representing the movement of the random walk within the phase space. Such history can be written as:

$$H_k = (S_0, t_0; S_1, t_1; \dots; S_m, t_m), \quad (2)$$

where k represents the simulation-run index, t_i the time of transition from state S_{i-1} to state S_i and m represents the index of the ending simulation time. The point (S_i, t_i) represents a phase space point, while the collection of all such points is called the phase space of the system. This space is continuous in time and discrete in the states. The transitions among system states depend on the component transitions from the up state to the down one, according to a stochastic law (the probability density function) that characterizes the component behavior.

The only information needed for the analysis are: a) the probability density function (pdf) of the time to failure of each component and their parameters values; b) the mission time of the system; and c) the system failure mode configuration.

The most common way to conduct the simulation of a FT is to consider the system as a whole; previous literature [34, 35] consider a system failure rate as the sum of all the component failure rate and calculate the transition time to the new state. After that, the component which performs the transition is chosen in a stochastic manner. When the component is chosen its failure rate is set equal to zero and the process is repeated till the occurrence of the TE or the end of the mission. The system operability (i.e. the occurrence of the TE) is evaluated each time a transition occurs. Generally it is convenient to make use of the FT

methodology to check the system operability each time it changes its state. The problem with this approach is that we must check the operability state of the system each time the system makes a transition; moreover it is needed to recalculate the system failure rate by imposing the failed component failure rate equal to zero. When considering DFTs we need to take into account the system evolution history in order to assess the nature of the reached state. Moreover in the case of components with different time distribution the traditional indirect Monte Carlo method [36], in which the transition time for the whole system is firstly sampled from the system transition time distribution and then the kind of transition is sampled, becomes impractical.

A more straightforward method to account for time-dependent failure and repair behaviours of components is the direct Monte Carlo simulation [37]. It samples the time to failure of each BE instead of the overall system transition time.

Following the idea of direct Monte Carlo, our simulation approach makes use simultaneously of the FT and the Monte Carlo methodologies. Instead of simulating the system walk in the phase space we consider BEs as basic entities. The failure time of each BE is calculated and these information are passed to the gates which they are connected with. The gate state is determined and, if failed, its failure time is passed to the higher levels. In this way for each gate is possible to calculate if a failure occurred before the mission time and pass the failure time to next level. These information are important for the dynamic gates. Moreover many simulation data can be stored in a very straightforward way: for each gate we can obtain the failure number of occurrences, the mean time to failure and information about which connected subsystem forced mostly the failure of the gate.

The approach followed is very suitable for dynamic gates since the failing order is tracked. Therefore, in this way it is not necessary to store the previous system states and check the order of occurrence to assess whether a state is of failure or good.

The complexity of the algorithm is carried by each single gate, whose logic is programmed in order to infer its own state. For the same reason, with this approach there is no need to update the overall system failure rate at each transition.

Generated all the transition times of each component, the failure time for each gate can be calculated based on the logic of the gate itself with respect to its inputs. At the highest level information about the system state are available and used to compute the system reliability estimate.

Due to the nature of the approach, it would be straightforward to program or modify the logic of the gates in a modular way, without compromising the environment already set. In this way, the end user can just make full use of the gates created by the programmers and exploit with some small knowledge of Simulink the power of the library.

4. MatCarloRe library

The simulation tool implemented make use of a high level modeling interface: it allows the user to assemble the FT by picking basic events and gates from a library and dropping these elements on a graphical interface furnished by Simulink®. BEs and gates are then linked together to create the system configuration. The tool consists of a Simulink® library called MatCarloRe (Fig. 1), formed of blocks representing the various parts of DFT, such as the dynamic gates PAND, SPARE, SEQ and FDEP as well as the static gates AND, OR and Voting. In addition to the gates it is necessary to insert a block that calculates the failure times of BEs. Each block representing a tree gate can receive n input and distribute m output by simply using the *mux* and *demux* blocks available in the main Simulink library. Inputs and outputs are of two kind: we define y as the binary vector which indicates whether the input and output have occurred (value 1) or not (value 0), and t as the vector containing the failure times. Only for the block representing the SPARE and the FDEP gates is not provided the input vector y .

Once the model is built and the input parameters are defined it is possible to run the simulation without a limit of iterations to achieve the desiderate estimation error. At each iteration the model returns a binary value that indicates whether the system has reached the state of failure or not. In particular, the model returns the value 1 if the fault is reached, 0 vice versa. To avoid large amounts of storage of the entire iterations binary vector the Simulink® block *memory* is used to set the progressive sum of results of each iteration. In this way only the total value of runs that revealed a fault is considered. The estimated unreliability of the system can be finally obtained dividing the number of runs which had shown the system failure by the total number of runs performed.

In the next section it is shown with small examples how to build a simulation model for a DFT introducing in more details the BE block and the dynamic gates. In order to proof the validity of the tool the results performed with the MatCarloRe are compared with analytical results.

Fig.1. MatCarloRe library elements.

4.1 BE block

The BE block is designed to generate the times of failure of basic events. It can generate the times of failure for components with negative exponential failure distribution (e.g. components subject to random failures) as well as fixed probability (e.g. the non-action of an operator). It is worth to mention the possibility to define events that follow distributions of any kind, such as Weibull, Normal distribution etc. simply by making small changes to the code in the BE block.

For components subjected to random failures the unreliability at time t can be expressed by the following relation:

$$F(t) = 1 - e^{-\lambda t}, \quad (3)$$

where λ is the failure rate of the component. The simulated failure time can be calculated by the inverse relationship:

$$t^* = -\frac{\log(1-F^*)}{\lambda}, \quad (4)$$

where F^* is a random number generated in $[0, 1]$. If t^* is smaller than the mission time t_m , the component is considered as failed.

For events supplied with a fixed probability q the following procedure is defined: a random number q^* generated uniformly in $[0, 1]$ is extracted and compare with the value q . This comparison returns a failure time according to the following relation:

$$t^* = \begin{cases} +Inf & \text{if } q^* \geq q \\ t_q^* & \text{if } q^* < q \end{cases} \quad (5)$$

where t_q^* is a random number generated uniformly in $[0, t_m]$.

4.2 PAND Block

The PAND block models the logic that underlies the PAND gate of a DFT [10, 14, 23]. The logic of the gate can be summarized as follows: the occurrence of the gate is obtained if all input components have failed before the mission time but in a fixed order (i.e. from left to right in the graphical representation). The block logic is illustrated in the flowchart in Fig. 2. First, according to the vector y , it is verified if all the input events occurred. If this condition is not satisfied the gate does not trigger. Otherwise the following conditions are checked: $t_i < t_j$ for $\forall i < j$ with $i, j = 1, 2, \dots, n$, where n is the gate number of inputs. If such control over failure times is satisfied the gate triggers with a time to failure equal to the maximum failure time of its inputs.

Fig. 2. Flow Chart of the PAND Block.

To test the validity of the block to perform the requested calculation we show the results of the following example. A PAND gate with two basic events is considered. The basic event failure rates are $\lambda = 0.1$ for the first component from the left and $\lambda = 0.01$ for the second one. The mission time is $t_m = 10$. In Fig. 3 is shown the model built for the simulation with the MatCarloRe tool. It is possible to see the BE block which takes as inputs the failure rates of the two components and the mission time of the system. The output of the block is the time to failure of the two components (i.e. vector t) and the binary vector y which indicates whether the time to failure of the components is smaller than the mission time. These vectors are given as input to the PAND block. The output scalar y_{PAND} of the PAND block is passed to a progressive sum and at the end of all the iterations the result is stored by the block TE into the Matlab® Workspace.

Fig. 3. Simulation model of a PAND gate with two basic event with failure rate $\lambda = 0.1, 0.01$ and mission time $t_m = 10$.

The simulated reliability versus the analytical one calculated by the mean of the associated CTMC are shown in Figure 4. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. It is evident that for a large number of iterations (of five magnitude order) the error is very low and acceptable.

Fig. 4. Simulated Vs analytical reliability of a PAND gate with two basic event with failure rate $\lambda = 0.1, 0.01$ and mission time $t_m = 10$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

4.3 SPARE Block

The SPARE block models the logic that underlies the SPARE gate of a DFT [10, 14]. It can be summarized as follow: given n active components, if one of them fails it can be replaced by one of the ns spare parts. The spare parts can fail under two conditions: either because of a failure when in latent state or because of a failure after becoming active. The spare becomes active when it replaces a failed active component or a failed active spare. The failure of the gate occurs when the number of surviving components is less than the number of required components which depends on the logic of the gate (usually it is assumed equal to the initial active components number). The block can, therefore, model the logic of cold stand-by (i.e. spare parts cannot fail during the latent time) or warm/hot stand-by (i.e. when it is possible for a spare to fail even if not running; usually with a lower failure rate).

The block logic is illustrated in the flowchart in Figure 5. Firstly it is needed to compute the time to failure of the components by the BE block. This is done for active components as well as for spare ones (both when active and in latent state). The SPARE block performs a permutation of the vector t , the time to failure of active components, sorting the vector in ascending order. Then, it is examined among active components whether there are components which have failed before the end of the mission time. If no failure is verified the gate will not trigger.

Vice versa, the algorithm checks if there are spare parts able to replace the active component that have failed. Let us consider the logic of replacement of a generic active component which fails. Its replacement can take place only if:

1. the spare part is still available (namely, it has not been used to replace another failed component) and;
2. the time to failure of the spare (during its latent condition) is greater than the time to failure of the active component to be replaced.

If these two conditions are not satisfied by any spare the gate triggers with a time of occurrence equal to the last failed active component. Otherwise the block updates the time to failure of the active component by adding the time to failure (when active) of the spare component chosen for the replacement. The substituting spare is finally declared as busy. The search for active components applicants for replacement is repeated till there are active components with failure time smaller than the mission time.

It is worth to highlight that the order in which the spares are chosen to replace the failed component follows the graphical order of positioning defined into the spare (e.g. in case of two spares that can replace an active failed component, it is chosen to replace the component with the spare that graphically is placed to the left).

This block offers many advantages in terms of modelling. Many reliability tools (Relex, Galileo [9]) are, in fact, difficult about the construction of DFT whit SPARE gates: if a spare part is shared among more components, the DFT will have as many SPARE gates as the number of active components which share the spare. In this way, the first input of the generic SPARE gate is the active component, while the second input is the shared spare part, common to all the set of SPARE gates drawn. The final logic implemented by the set of SPARE gates is realized linking them together by an OR gate in the upper level (Figure 5).

Fig. 5. Illustration of SPARE gates modelling vantages introduced by the tool; case of common shared spare. Left: Relex model; right: MatCarloRe model.

Another situation is redundancy in the active components (i.e. not all the active components are requested for the system to work). In this case an AND gate is placed to the upper level (Figure 6). If the number of active components is even greater than the presented examples in the previous figures the modelling activity is even more complex involving the use of AND and OR gates in the tree structure. Therefore what the SPARE gate of the MatCarloRe library is able to do is to bypass all these architectural tricks, by the simple use of a single block called SPARE k/N , where k is the number of components requested to work and N is the number of initial active components (Figure 6).

Fig. 6. Illustration of SPARE gates modelling vantages introduced by the tool; case of redundancy in active components. Left: Relex model; right: MatCarloRe model.

The differences in the flow chart between the two SPARE gates considered are located in the first rhombus of the chart in Figure 7 where it is needed to consider the $N-k$ failures allowed for the system to work.

Fig. 7. Flow Chart of the SPARE Block.

To test the validity of the two blocks SPARE and SPARE_k/N the results of two simple example are shown. The system is composed of two active and two spare components with failure rate $\lambda = 0.1$. The latency factor for the spare components is $a = 0.1$ and the mission time is $t_m = 10$. Figures 8 and 10 show the models built with the MatCarloRe tool in the two different cases. The simulation models are equal in both cases except that the number of components needed for the system to work in the second example are defined by $nreq$.

The BE block takes as inputs the failure rates of the two active components, the failure rate of the spare components when in latent state and when active and the mission time of the system. It returns as output the time to failure of the two active components stored in the vector t . The time to failure of spare parts when in latent state and when active are given respectively in the vectors tl and ts . Vectors t , tl and ts and the mission time t_m are the input of the SPARE block. The output y_{SPARE} , taken over repeated iterations, is then used to compute reliability value.

Fig. 8. Simulation model of a SPARE gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$.

In Figure 9 is shown the simulated reliability versus the analytical calculated by the mean of the associated CTMC. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. It is evident that for iteration of order 10^5 the error of prediction is very low and acceptable.

Fig. 9. Simulated Vs analytical reliability of a SPARE gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

Fig. 10. Simulation model of a SPARE_k/N gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$; $nreq = 1$.

The results of the simulation of the k/N model are shown in Figure 11. Again for iteration of order 10^5 the simulated reliability is very close to the CTMC results.

Fig. 11. Simulated Vs analytical reliability of a SPARE k/N gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$; $nreq = 1$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

4.4 SEQ Block

The feature of the SEQ gate is to force the components - inputs of the gate \square to move towards the state of failure in a fixed order [10, 14, 23]. This order is usually expressed graphically by the position of the gate inputs, from left to right. It is generally used to represent different levels of degradation of a component. Therefore, a condition for the gate to trigger is the occurrence of all its inputs. The algorithm used for this task is simple: the SEQ block firstly calculates the sum S of the time to failure of all its inputs. If S is smaller than the mission time the gate triggers with a time to failure equal to S .

4.5 FDEP Block

The FDEP block models the FDEP gate of a DFT [10, 14, 23]. The feature of this gate is to force the input components to reach the failure state if the trigger event has occurred before they fail by themselves. The block checks if the failure time of each input component is smaller than the trigger failure time. If the condition is true the component will fail with its own failure time. Vice versa the component will occur with failure time equal to the trigger failure time.

In the construction of a model with a FDEP, each component subjected to the action of the trigger is firstly connected to the FDEP gate and then the output of the latter is passed to the gate interested by the given component. We do not show the flow chart due to the simplicity of the task performed by the block. Likewise we do not show any example for the FDEP block due its similarity with an OR gate between the trigger event and any of the basic event of the gate.

5. Study Case

In this section we present a case of study of a real complex system, in order to demonstrate the effectiveness of the simulation tool to calculate the reliability of such systems. The case of study represents the FT model of a plant section for the alkylation and treatment of light olefin. Following the top-down procedure of the FT analysis, the tree was designed. The static-tree structure is shown in Figure 12 and Table 1 reports the component failure rates.

Beside that model, the DFT was designed in order to consider a more realistic safety behavior that the plant exposes. The dynamic re-arrangement considered concerns the modeling of the gates IE1, IE8 and of the TE. In fact, in the static modeling they are represented with the traditional AND gates. That results in an approximate evaluation of the logic for the real system, since time dependencies cannot be considered with the static-FT. In the DFT, the gate IE8 was substituted with a SPARE gate as in a classic cold stand-by redundant configuration. The second re-arrangement is done by substituting IE1 with a PAND, in order to consider the priority condition that IE3 has on IE4. The same process is applied at the TE gate.

Fig. 12. FT of the section plant considered.

Table 1. Input data for basic events of the FT of Fig. 10.

Therefore, three cases were studied: (i) simulation of the static-FT, (ii) simulation of the DFT without fixed probabilities by substituting q with the relative failure rate calculated through (3) (i.e. assuming the value of F equal to q and calculating the failure rate trough inverse relationship); (iii) the simulation of the DFT with the original parameters of Table 1.

The analytical resolution of these three cases expose different levels of complexity. In fact, the case (i) is the simplest because no time dependencies arise and traditional techniques, based on the Boolean algebra, can be used. Case (ii) introduces two kind of dynamic gates. One of them is placed at the TE. It makes impossible the use of techniques to relax the complexity of the model (e.g. modularization [20, 21, 22, 23, 24, 25]) without incurring in approximated calculation. The case (iii) can be classified as the most complex since it cannot be solved with the use of the traditional CTMC paradigm due to the presence of fixed probabilities. For this last case no analytical result are presented. The model of the case (iii) implemented in the MatCarloRe tool is shown in Figure 13.

Fig. 13. MatCarloRe model of the DFT of Fig. 10. Case (iii).

We conducted eight simulation for each case. The number of iterations were chosen till the maximum value of 10^8 . For cases (i) and (ii) analytical results were computed through Relex®. The unreliability of the simulation model converges to the analytical result with 10^3 iterations in the case (i) with a very small relative error. In the case (ii) more iterations are needed to obtain valid results because of the more complex nature of the system involving temporal dependencies. About 10^8 iterations to achieve a small estimating error. In the case (iii) we claim that the number of iterations needed to achieve a small error in case (ii) could be used as well. This is supported by the fact that the unreliability seems to stabilize around 10^8 iterations.

Table 2. Unreliability and relative error for the model MatCarloRe of the FT in Fig. 10. Case(i): SFT with fixed probability for BE1 and BE3; Case(ii): DFT with failure rate for BE1 and BE3; Case(iii): DFT with fixed probability for BE1 and BE3.

6. Conclusion

In this paper we summarize an integrating technique of Monte Carlo simulation and FT methodology for reliability assessment of complex systems in presence of time dependencies. We showed that our simulating environment can go beyond the limitations of analytical methodologies with the additional advantage of a high level modelling interface based on the FT method. Some results are presented reporting good performance in terms of modeling and calculation efforts. Moreover important contributions for the SPARE model are introduced, reducing the efforts of the construction of a complex FT. It is shown that for iteration of order $10^7 - 10^8$ it is possible to obtain reliable results for real system in a time frame of 1-10 hours. Reducing computation time technique are well developed (e.g. biasing techniques) and could be easily introduced in the MatCarloRe tool. In the future our effort will be pushed in the development of tool characteristics for the calculation of important measures, availability and system performance indicators.

References

- [1] Roberts, N. H. , Vesely, W. E., Haasl, D.F., & Goldberg, F.F. (1981). *Fault tree handbook*, NUREG-0492. Washington: US NRC.
- [2] Ren, Y., & Dugan, J. B. (1998). Design of reliable systems using static and dynamic fault trees. *IEEE Transactions on Reliability*, 47, 234-244.
- [3] Siu, N. (1994). Risk assessment for dynamic systems: an overview. *Reliability Engineering and System Safety*, 43, 43-73.
- [4] Cepin, M., & Mavko, B. (2002). A dynamic fault tree. *Reliability Engineering and System Safety*, 75, 83-91.
- [5] Amari, S., Dill, G., & Howald, E. (2003). A new approach to solve dynamic fault trees. *Annual Reliability and Maintainability Symposium*, 374-379.
- [6] Distefano, S., & Puliafito, A. (2007). Dynamic reliability block diagrams vs dynamic fault trees. *In Proceedings Annual Reliability and Maintainability Symposium RAMS '07, Jan. 22-25*, 71-76.
- [7] Dugan, J. B., Venkataraman, B., & Gulati, R. (1997). Diftree: a software package for the analysis of dynamic fault tree models. *In Proceedings Annual Reliability and Maintainability Symposium, Jan. 13-16*, 64-70.
- [8] Dugan, J. B., Trivedi, K. S., Smotherman, M. K., & Geist, R. M. (1986). The hybrid automated reliability predictor. *Journal of Guidance, Control, and Dynamics*, 9, 319-331.
- [9] Sullivan, K. J., Dugan, J. B., & Coppit, D. (1999). The galileo fault tree analysis tool. *In Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, June 15-18*, 232-235.
- [10] Boudali, H., Crouzen, P., & Stoelinga, M. (2007). Dynamic fault tree analysis using input/output interactive markov chains. *In Proceedings 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks DSN '07, June 25-28*, 708-717.
- [11] Montani, S., Portinale, L., Bobbio, A., & Codetta-Raiteri, D. (2008). RADYBAN: A tool
- [12] for reliability analysis of dynamic fault trees through conversion into dynamic bayesian networks, *Reliability Engineering and System Safety*, 93, 922-932.
- [13] Bobbio, A., Portinale, L., Minichino, M., & Ciancamerla, E. (2001). Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. *Reliability Engineering and System Safety*, 71, 249-260.
- [14] Volovoi, V. (2004). Modeling of system reliability petri nets with aging tokens. *Reliability Engineering and System Safety*, 84, 149-161.
- [15] Durga Rao, K., Gopika, V., Sanyasi Rao, V. V. S., Kushwaha, H. S., Verma, A. K., & Srividya, A. (2009). Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment. *Reliability Engineering and System Safety*, 94, 872-883.
- [16] Marsaguerra, M., Zio, E., Devooght, J., & Labeau, P. E. (1998). A concept paper on dynamic reliability via Monte Carlo simulation. *Mathematics and Computers in simulation*, 47, 371-382.
- [17] Marquez, A. C., Heguedas, A. S., & Iung, B. (2005). Monte Carlo-based assessment of system availability. A case study for cogeneration plants. *Reliability Engineering and System Safety*, 88, 273-289.
- [18] Zio, E., Marella, M., & Podollini, L. (2007). A Monte Carlo simulation approach to the availability assessment of multi-state systems with operational dependencies. *Reliability Engineering and System Safety*, 92, 871-882.
- [19] Zio, E., Podofillini, L., & Levitin, G. (2004). Estimation of the importance measures of multi-state elements by Monte Carlo simulation. *Reliability Engineering and System Safety*, 86, 191-204.
- [20] Marseguerra, M., & Zio, E. (2004). Monte Carlo estimation of the differential importance measure: application to the protection system of a nuclear reactor. *Reliability Engineering and System Safety*, 86, 11-24.
- [21] Chatterjee, P. (1975). Modularization of fault trees: A method to reduce the cost of analysis. *SIAM Reliability and Fault Tree Analysis*, 101-137.
- [22] Rosenthal, A. (1980). Decomposition methods for fault tree analysis. *IEEE Transactions of Reliability*, R-29, 136-138.
- [23] Khoda, T., Henley, E. J., & Inoue, K. (1989). Finding modules in fault trees. *IEEE Transactions on Reliability*, 38, 165-176.
- [24] Dugan, J. B., Bavuso, S. J., & Boyd, M. A. (1992). Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Transactions on reliability*, 41, 363-377.

- [25] Dugan, J. B., Sullivan, K. J., & Coppit, D. (2000). Developing a low cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transaction on reliability*, 49, 49-59.
- [26] Meshkat, L., Dugan, J. B., & Andrews, J. D. (2002). Dependability Analysis of Systems With On-Demand and Active Failure Modes, Using Dynamic Fault Trees. *IEEE Transactions on reliability*, 51, 240-251.
- [27] Anand, A., & Somani, A. K. (1998). Hierarchical analysis of fault trees with dependencies, using decomposition. *Proceedings Annual on Reliability and Maintainability Symposium*, 69-75.
- [28] Huang, C. Y., & Chang Y. R. (2007). An improved decomposition scheme for assessing the reliability of embedded systems by using dynamic fault trees. *Reliability Engineering System Safety*, 92, 1403-1412.
- [29] Lanus, M., Yin, L., & Trivedi, K. S. (2003). Hierarchical Composition and Aggregation of State-Based Availability and Performability Models. *IEEE Transactions on reliability*, 52, 44-52.
- [30] Feinberg, B. N., & Chiu, S. S. (1987). A Method to Calculate Steady-State Distributions of Large Markov Chains by Aggregating States. *Operations Research*, 35, 282-290.
- [31] Malhotra, M., & Trivedi, K. S. (1993). A methodology for formal specification of hierarchy in model solution. In *Proceedings Fifth International Workshop Petri Nets and Performance Models, (PNPM-1993)*, 258-267.
- [32] Windebank, E. (1983). A Monte Carlo Simulation Method Versus a General Analytical Method for Determining Reliability Measures of Repairable Systems. *Reliability Engineering*, 5, 73-81.
- [33] Goldfeld, A., & Dubi, A. (1987). Monte Carlo Methods in Reliability Engineering. *Quality and Reliability Engineering International*, 3, 83-91.
- [34] Dubi, A. (1989). Monte Carlo Methods in Reliability. *Operation Research and System Engineering Commission of the European Communities Joint Research Centre, Ispra Italy*.
- [35] Lewis, E. E., & Bohm, F. (1984). Monte Carlo Simulation of Markov Unreliability Models. *Nuclear Engineering and Design*, 77, 49-62.
- [36] Dubi, A. (1986). Monte Carlo Calculations for Nuclear Reactors. *CRC Handbook of Nuclear Reactors Calculations, Vol. II*, CRC PRESS.
- [37] Wu, Y. F., & Lewins, J. D. (1992). Monte Carlo Studies of Engineering System Reliability. *Annual nuclear engineering*, 19, 825-859.
- [38] Zio, E. (1995). Biasing the transition probabilities in direct Monte Carlo. *Reliability Engineering and System Safety*, 47, 59-63.

List of figures

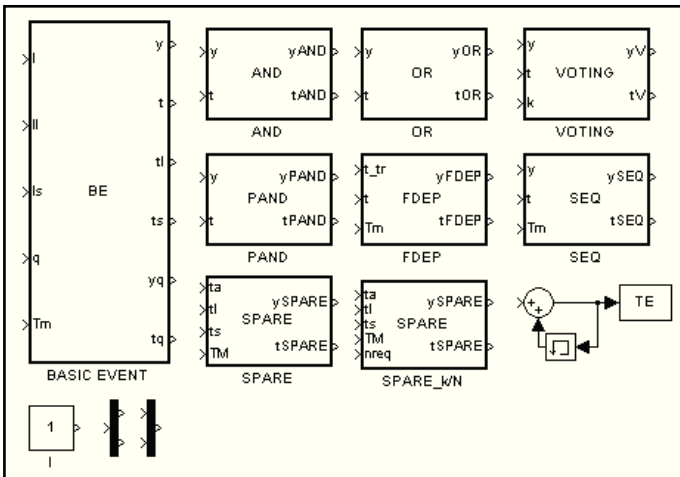


Fig. 1. MatCarloRe library elements.

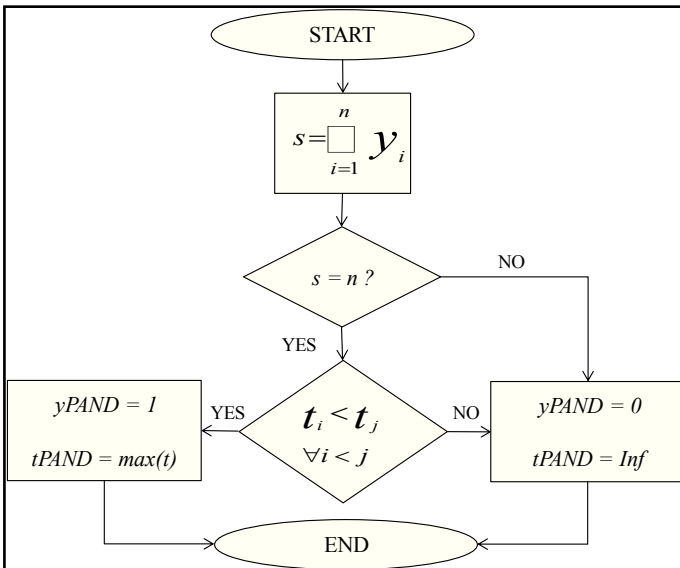


Fig. 2. Flow Chart of the PAND Block.

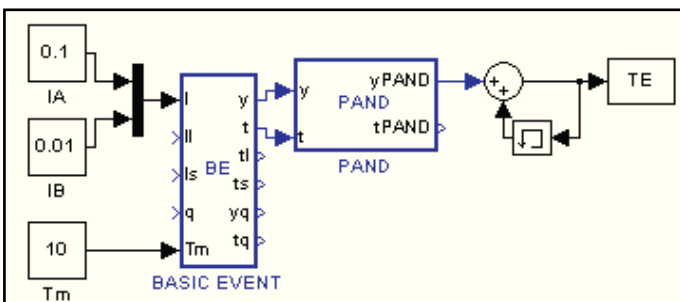


Fig. 3. Simulation model of a PAND gate with two basic event with failure rate $\lambda = 0.1, 0.01$ and mission time $t_m = 10$.

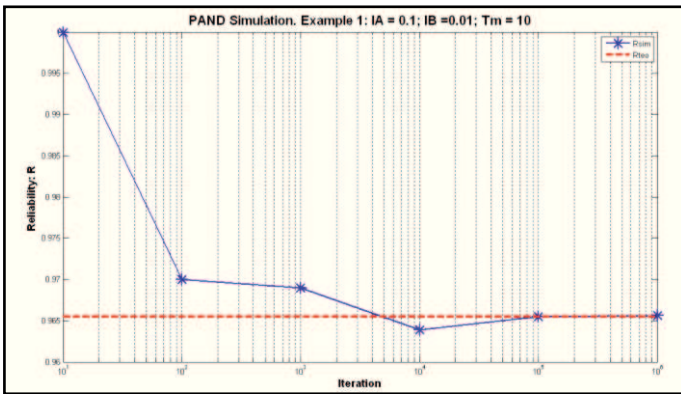


Fig. 4. Simulated Vs analytical reliability of a PAND gate with two basic event with failure rate $\lambda = 0.1, 0.01$ and mission time $t_m = 10$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

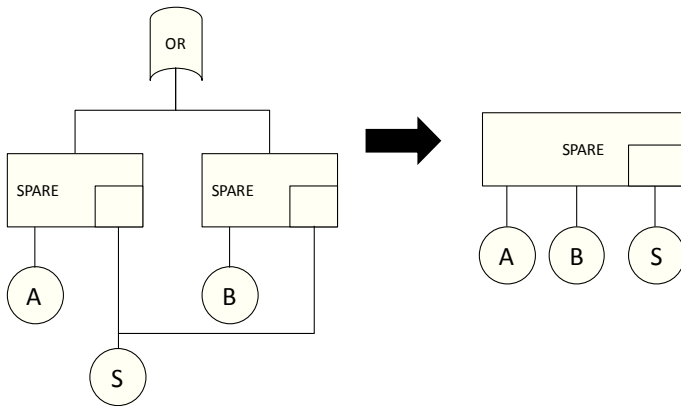


Fig. 5. Illustration of SPARE gates modelling vantages introduced by the tool; case of common shared spare. Left: Relex model; right: MatCarloRe model.

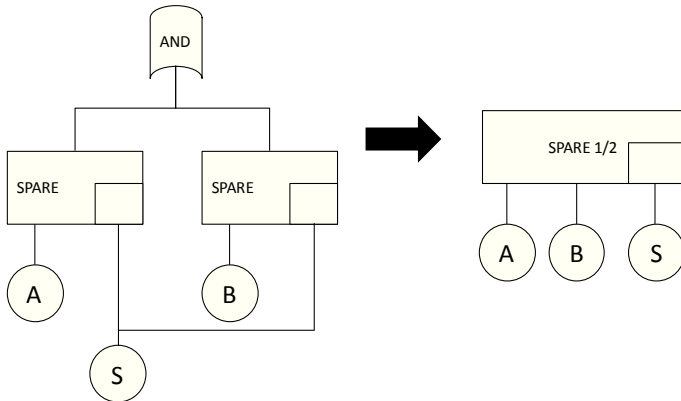


Fig. 6. Illustration of SPARE gates modelling vantages introduced by the tool; case of redundancy in active components. Left: Relex model; right: MatCarloRe model.

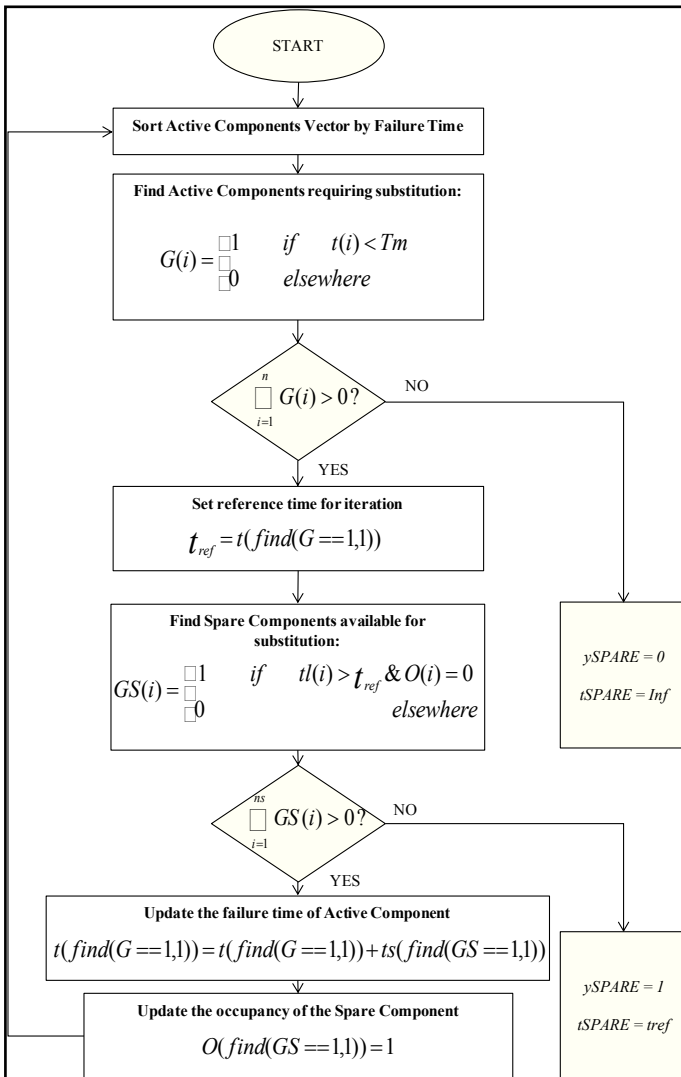


Fig. 7. Flow Chart of the SPARE Block.

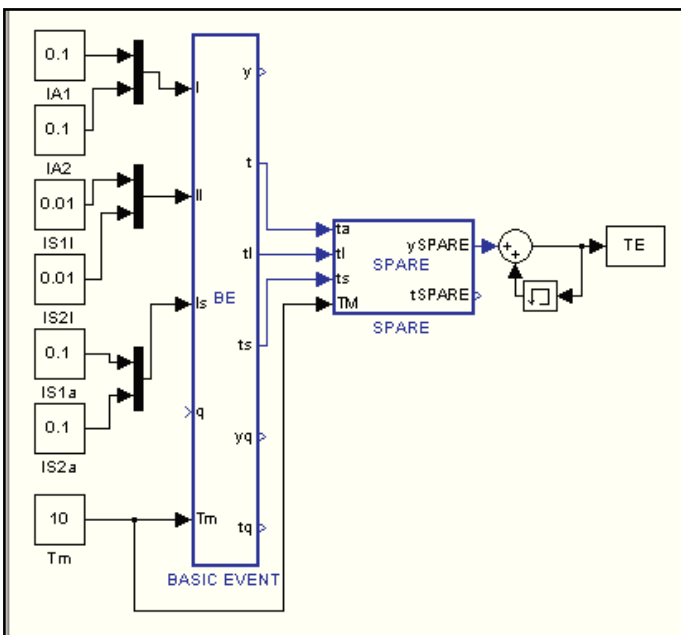


Fig. 8. Simulation model of a SPARE gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$.

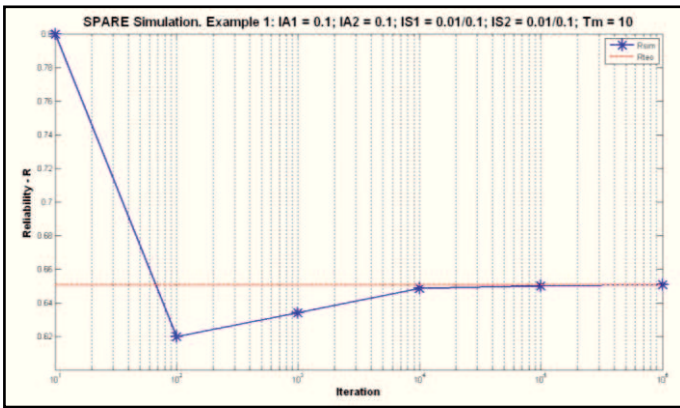


Fig. 9. Simulated Vs analytical reliability of a SPARE gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

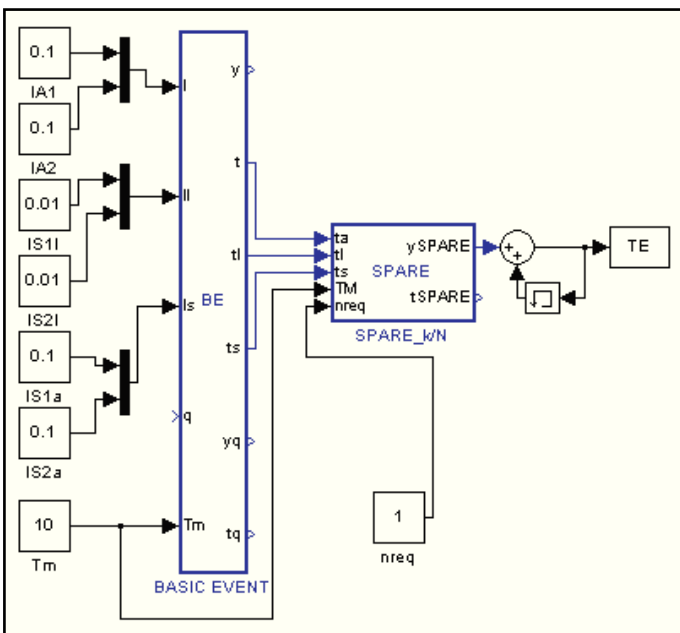


Fig. 10. Simulation model of a SPARE_{k/N} gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$; $nreq = 1$.

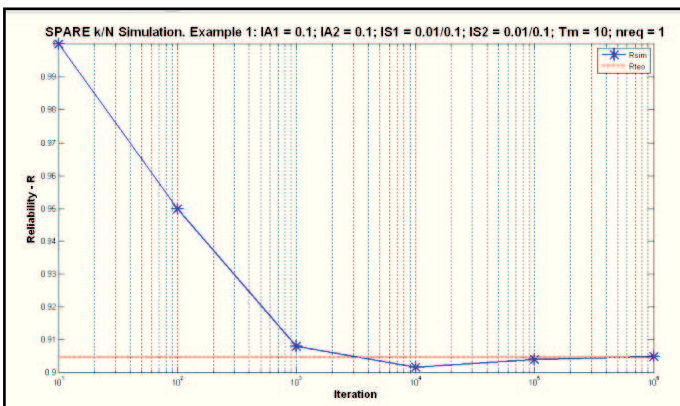


Fig. 11. Simulated Vs analytical reliability of a SPARE_{k/N} gate with two active components with failure rate $\lambda = 0.1$, two spare components with failure rates $\lambda = 0.1$, latency factor $a = 0.1$ and mission time $t_m = 10$; $nreq = 1$. Iteration are chosen equal to 10^i with $i = 1, 2, \dots, 6$. Dotted line: analytical result; marked line: simulated results.

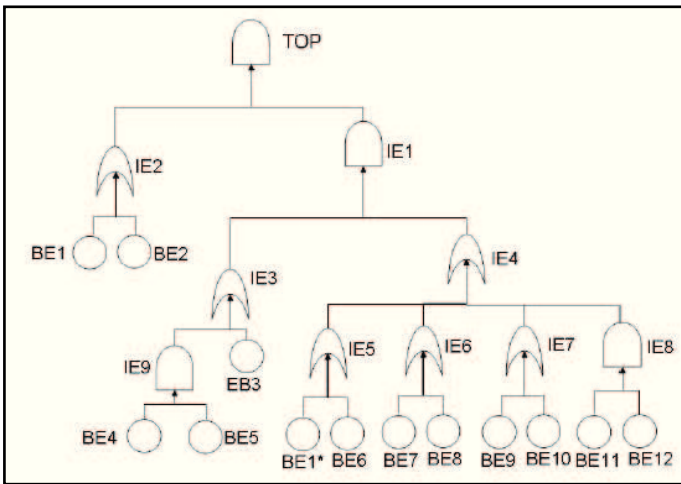


Fig. 12. FT of the section plant considered.

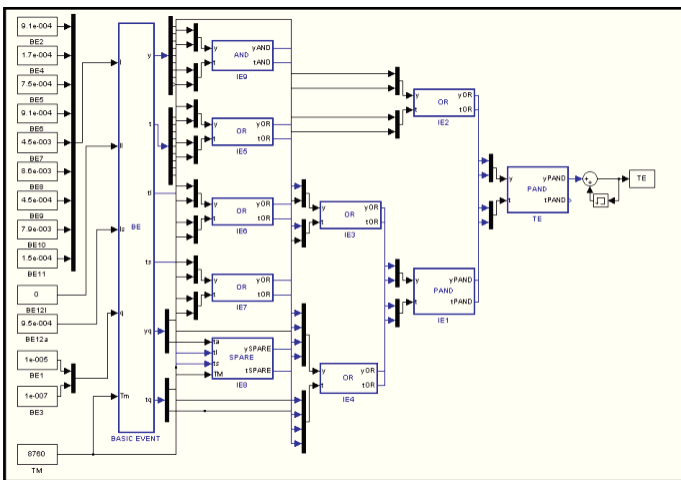


Fig. 13. MatCarloRe model of the DFT of Fig. 12. Case (iii).

List of tables

ID	λ	q
BE1	-	$1.0 \cdot 10^{-5}$
BE2	$9.1 \cdot 10^{-4}$	-
BE3	-	$1.0 \cdot 10^{-7}$
BE4	$1.7 \cdot 10^{-4}$	-
BE5	$7.5 \cdot 10^{-4}$	-
BE6	$9.1 \cdot 10^{-4}$	-
BE7	$4.5 \cdot 10^{-3}$	-
BE8	$8.6 \cdot 10^{-3}$	-
BE9	$4.5 \cdot 10^{-4}$	-
BE10	$7.9 \cdot 10^{-3}$	-
BE11	$1.5 \cdot 10^{-4}$	-
BE12	$9.5 \cdot 10^{-4}$	-

Table 1. Input data for basic events of the FT of Fig. 12

Iter	Case (i)	Err.rel%	Case (ii)	Err.rel%	Case (iii)
10^1	0,7000	9,45%	0	100,00%	0
10^2	0,8000	3,48%	0	100,00%	0
10^3	0,7730	0,01%	0	100,00%	0
10^4	0,7717	0,18%	1,00E-04	88,39%	0
10^5	0,7734	0,04%	6,00E-05	13,03%	3,00E-05
10^6	0,7733	0,02%	6,20E-05	16,80%	4,40E-05
10^7	0,7732	0,02%	5,58E-05	5,12%	5,49E-05
10^8	0,7731	0,00%	5,36E-05	0,98%	5,45E-05
Fteo	0,7731		5,31E-05		

Table 2. Unreliability and relative error for the model MatCarloRe of the FT in Fig. 12. Case(i): SFT with fixed probability for BE1 and BE3; Case(ii): DFT with failure rate for BE1 and BE3; Case(iii): DFT with fixed probability for BE1 and BE3.

The Effect of Correlated Failure Rates on the Reliability of Continuous Time 1-out-of-2 Software

Peter Popov[†], Gabriele Manno[‡]

[†]Centre for Software Reliability, City University,
Northampton Square, London, UK
e-mail: ptp@csr.city.ac.uk

[‡]Department of Mathematics and Informatics, University of Catania
Viale Andrea Doria 6, Catania, Italy
e-mail: gmanno@dmf.unict.it

Abstract. In this paper we study the effects on system reliability of the correlation over partitions of the input space between the rates of failure of two-channel fault-tolerant control software. We use a continuous-time semi-Markov model to describe the behavior of the system switching between different modes of operations (i.e. processing inputs from the different partitions of the input space). We demonstrate via simulation that the variation of the failure rates of the channels over the partitions of the input space can affect the system reliability very significantly. With a plausible range of model parameters we observed that the mean time to system failure may vary by more than an order of magnitude: positive correlation between the channel rates makes the system less reliable while negative correlation between the channel rates implies that the system is more reliable than assuming constant failure rates for the channels. The effects we report are similar to those observed for on-demand software systems. Our observations seem to make a case for more detailed reliability measurements than is typically undertaken in practice. We briefly discuss model parameter estimation applying the theory of competing risks. Finally we compare our model with similar models for a single channel software developed in the past by others and discuss ways forward.

1. Motivation

All systems need to be *sufficiently* reliable. There are two related issues here. In the first place there is the issue of *achieving* the necessary reliability. Secondly, there is the issue of *assessing* the reliability that has actually been achieved, to convince oneself that it is 'good enough'.

In the light of the rather strict limitations to the levels of software reliability that can typically be achieved or claimed from observation of operational behavior of a single version program [1], fault tolerance via *design diversity* has been suggested as a way forward both for achieving higher levels of reliability, and for assisting in its assessment.

Design diversity has been studied thoroughly in the past 30+ years. For a relatively recent study the reader is referred to [2]. The focus, however, has been primarily on

‘on demand’ systems, e.g. a protection system called upon when a failure is detected in the operation of the system controlling a plant.

The focus of this paper is *control software*, i.e. which exercises control of an object of control and in the process executes a series of inputs (trajectories) coming directly from the controlled object, its environment and also the internal state of the software itself. Assessing accurately reliability of control software is important not only for minimizing the losses due to downtime. In some cases, e.g. of critical applications, the control software reliability defines reliability requirements for the protection system designed and deployed to deal with situations of inadequate control. A protection system of given reliability may be adequate in some cases – e.g. when the control system is very reliable – or may be inadequate – e.g. if the control system is of modest reliability. The reliability of the total system (control and protection) depends on both the reliability of the control and of the protection and therefore accurately assessing reliability of both systems is important.

Our focus in this paper is a 2-channel control software for which the input space is divided in partitions, which represent different *modes of operation*. Examples of modes of operation might be an initialization, a normal control loop and terminating the control, e.g. to allow for maintenance. More refined scenario, e.g. as in robotics, might include a robot having to deal with different obstacles, which may require applying different algorithms of adaptive behavior to the current environment, etc. We address on purpose the problem at a sufficiently high level of abstraction – using a continuous time semi-Markov model – which will allow us to state the main result in a concise way.

Continuous-time semi-Markov models are typically used to model the behavior of control software: the modeling assumptions and the model details depend on the specific aspects of interest. For instance, failure clustering is typical for control software [3]. Modeling such behavior is impossible with models assuming that successive inputs are drawn independently from the input space. Instead, models, in which the failure rate changes significantly after the occurrence of first failure proved to be useful [4].

The paper is organized as follows. Section 2 states the problem. Section 3 presents the model and the main result. In section 4 we compare our model with a model developed in the past by Littlewood for a single channel software with modular structure. In section 5 we discuss our findings and some parameter estimation techniques. Section 6 offers a survey of the relevant literature. The conclusions and directions for future research are presented in section 7.

2. The problem

Consider a 2-channel control system as shown in Figure 1. During the operation of the system each of the channels can fail and so can the adjudicator. In this paper we concentrate on the case of an *absolutely reliable adjudicator* and study the reliability of the control system only. Once a failure of a channel is detected by the adjudicator an attempt is undertaken to ‘repair’ the failed channel, which eventually succeeds after time τ , during which time the other channel will either work correctly or will

also fail. Examples of repair envisage here are the typical backward/forward recovery mechanisms used in practice such as retrying the execution with a slightly perturbed data [3] or merely restarting the channel.

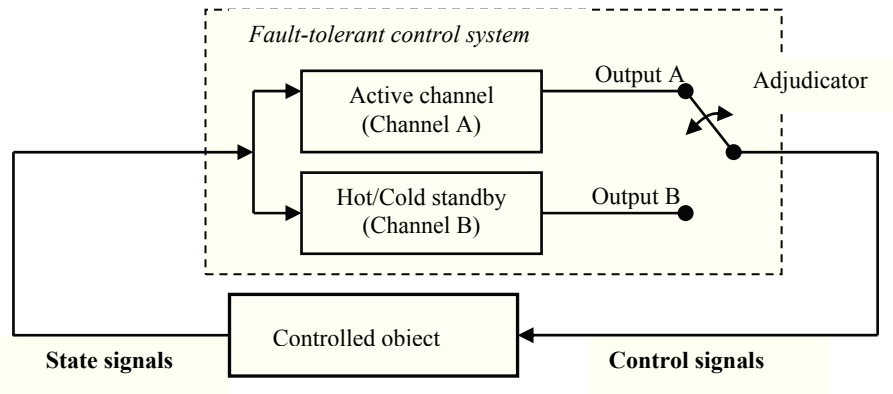


Figure 1. A typical architecture of a 2-channel control system. At any time the actuators of the controlled object (e.g. a nuclear plant) are generated by one of the channels, while the second channel is available as a hot/cold standby. An adjudicator is responsible to detect anomalies of the active channel and switch to the standby channel, if such is available. The failed channel is ‘repaired’ which takes finite time and becomes available to take over control again. The channels are diverse – if design faults are considered – or merely redundant if only hardware related faults are considered.

The channels are assumed to fail independently of each other: the chances of both channels failing simultaneously are, therefore, vanishingly small. The only source of coincident failure is the finite repair time τ of the failed channel during which the second channel may also fail. We later will discuss relaxing the assumption of independent failure and discuss the model of a “common stress” that might cause simultaneous failure of both channels.

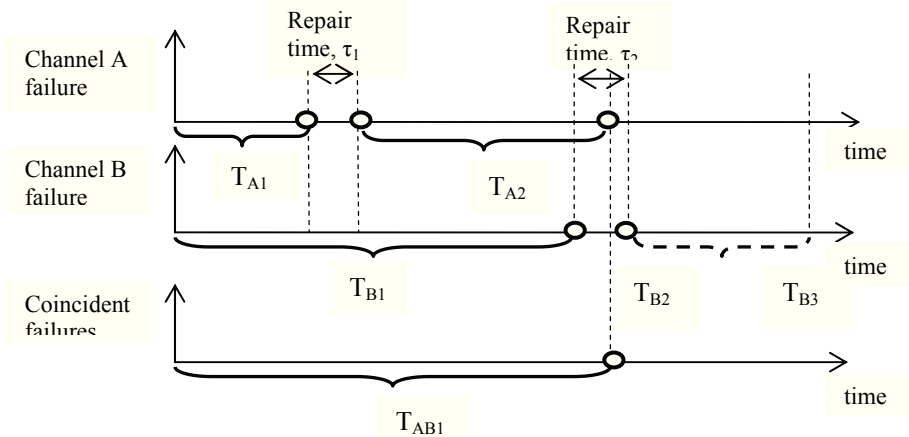


Figure 2. The timing diagram illustrates the events of interest and the times associated with them. The times, T_{Ax} , T_{Bx} and T_{ABx} , respectively, characterize the up times in the stochastic processes associated with the behavior of the individual channels and the control system.

We assume, further that the system’s input space is divided into *partitions*, identical for both channels. Each of these partitions is associated with rates of failure of the channels, respectively. These rates may vary across partitions and it is the nature of this variation – whether the rates of channel failure are positively or negatively correlated or not correlated (e.g. the rate of one of the channels does not vary over the partitions at all) – that the study is focused on.

Figure 2 shows a typical timing diagram which illustrates the failure processes of interest.

In this study we concentrate on the time to system failure (i.e. the times until both channels fail) starting from a state when both channels are operational. Clearly, the time to failure may include multiple cases of a single channel failure and successful repairs of the failed channel.

3. Model of the system

We studied the problem using the formalism of stochastic activity networks (SAN) and the tool support offered by the Möbius tool [5]. The results – the distribution of the time to system failure – are obtained *via simulation*.

3.1. Diagrammatic representation of the model

Now we defined the system model. Consider that the system can be represented as a stochastic activity network, in which there are several partitions as shown in Figure 3.

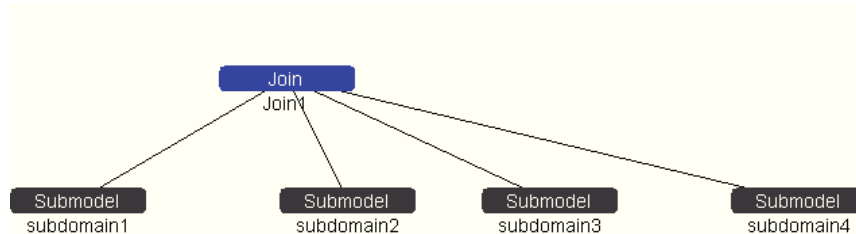


Figure 3. Model of a system operating on 4 partitions of the input space, subdomain1 – subdomain4. The syntax of the graphical representation is Möbius specific. Each of the partitions is a detailed representation of the states that the system (the two channels) might be in while in the respective partition.

Figure 4 shows in detail the system behavior of the system in one of the partitions. The models of the other partitions are identical, but the parameters may differ. The system changes its state from both channels working correctly (OO) to states in which one of the channels has failed (OF or FO), from which it may either recover (i.e. return to OO) or instead the second channel may also fail (i.e. reach the state FF). While in OO state, the system may switch to a different partition: the other partitions are labeled S_2 , S_3 , S_4 , which are really labels for the OO states in the respective partitions (**subdomain₂**, **subdomain₃** and **subdomain₄**). One notices that in our model the

system cannot switch to a different partition unless both channels work correctly. This simplifying assumption seems plausible. In a typical scenario of a fast repair (e.g. a restart) and a relatively infrequent change of modes of operation, the chances that the system will have moved to a different partition are negligible. In some other cases, however, the transitions between the partitions may be fast and the simplification introduced in the model may be problematic. Relaxing this assumption, although possible, is beyond the scope of the paper.

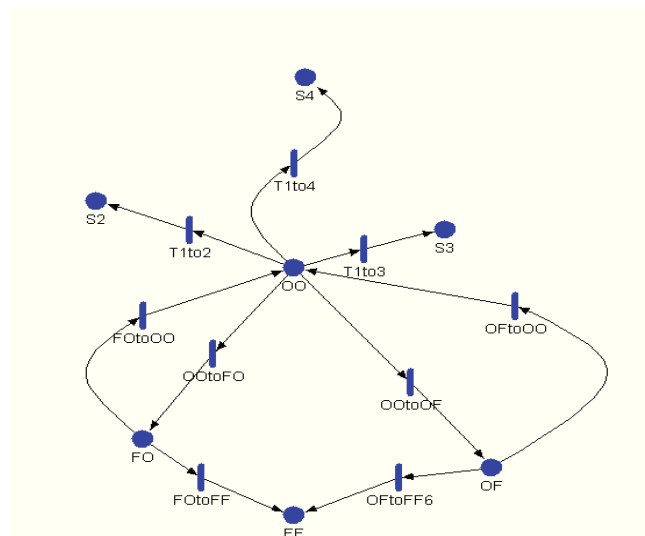


Figure 4. Detailed behavior of the 1-out-of-2 software in partition **subdomain₁**. The model states are shown as places (nodes) OO, OF, FO and FF suitably named to indicate the state of the channels: both channels working correctly is represented by the place OO, ..., both states having failed is represented by the place FF, respectively. The transitions between the places are characterized by a set of ‘stochastic activities’, e.g. a change of the system state from OO to FO is represented by the stochastic activity OOttoFO. A transition in the opposite direction (FO → OO) is represented by the stochastic activity FOttoOO. The place FF is absorbing, i.e. there are no outgoing transitions (activities) from it to some other places.

3.2. Möbius model parameters

The model is parameterized under a set of assumptions:

- Failures of the channels are driven by independent Poisson processes, which are homogeneous conditional on sub-domains, but may be non-homogeneous across partitions.
- Repairs of the channels are perfect, but not instantaneous. Repairs are only undertaken if there is a channel working correctly.

Given these assumptions the model was parameterized as follows.

- The transitions between the partitions (between the respective OO states, that is) are all assumed *exponentially distributed* with a rate of 0.3. The

uniformity of the rates here was chosen for *convenience*: we wanted to keep the channels equally reliable and be able to vary easily their rates of failure in partitions. Any difference, thus, in the system behavior observed between the studied cases could be attributed solely to the correlation between the failure rates in the partitions. This objective is easily achieved if the domains are equally likely, which in turn is achieved by setting the same transition rates between the OO states of the partitions.

- The repair times were assumed of *fixed duration*, 0.01 units.
- The failure rates in the partitions are chosen from the set $\{0.01, 0.02, 0.03\}$ in such a way that the marginal rates of failure of the channels remain unchanged (0.02 given the partitions are equally likely, ≈ 0.25).

The following cases (see Table 1) were studied, which represent different types of correlation between the failure rates of the channels over the partitions.

Table 1. Failure rates of the channels conditional on partitions (S_1 - S_4)¹

		S_1	S_2	S_3	S_4
Experiment 1: ‘High’ Positive correlation between the rates.	Channel 1	0.03	0.01	0.03	0.01
	Channel 2	0.03	0.01	0.03	0.01
Experiment 2: ‘High’ Negative correlation between the rates.	Channel 1	0.03	0.01	0.03	0.01
	Channel 2	0.01	0.03	0.01	0.03
Experiment 3: Constant rates of both channels.	Channel 1	0.02	0.02	0.02	0.02
	Channel 2	0.02	0.02	0.02	0.02
Experiment 4: Constant rate of channel 1.	Channel 1	0.02	0.02	0.02	0.02
	Channel 2	0.01	0.03	0.01	0.03
Experiment 5: ‘Low’ positive correlation between the rates.	Channel 1	0.01	0.02	0.03	0.02
	Channel 2	0.01	0.02	0.03	0.02
Experiment 6: ‘Low’ Negative correlation between the rates.	Channel 1	0.03	0.02	0.01	0.02
	Channel 2	0.01	0.02	0.03	0.02

As one can see, a uniform profile on the set of partitions ($P(S_1) = P(S_2) = P(S_3) = P(S_4) = 0.25$) guarantees that the marginal rates of failure of the channels indeed remains the same – 0.02.

3.3. Measure of interest

The *time to system failure* was measured via simulation and the results are summarized in Table 2.

Clearly, the mean time to system failure differs is significantly affected by the covariance between the failure rates². The greatest MTTF corresponds to Experiment 2 with high negative correlation between the rates of failure of the channels. The other extreme – the shortest MTTF – corresponds to the case with high positive correlation between the conditional rates of failure of the channels. A constant rate of failure of at least one of the channels (Experiment 3 and Experiment 4) forms the ‘case in the middle’, while more modest correlations – either positive or negative – place the

¹ $S_1 - S_4$ are shortcuts for **subdomain**₁ – **subdomain**₄, respectively.

² The MTTF of Experiment 3 and 4 are very close, but for these the covariance of the failure rates is 0, as for at least one of the channels the failure rates are constant over partitions.

respective cases between the ‘case in the middle’ and the respective cases with high correlation of the same sign.

Table 2. Mean time to system failure

	Mean		Runs
	Value	95% Confidence interval	
‘High’ Positive correlation between rates (Experiment 1)	97,569.53	+/- 2,000.9	12,000
‘High’ Negative correlation between rates (Experiment 2)	157,353.8	+/- 4,563.3	5,000
Constant rates of both channels (Experiment 3)	122,060.4	+/- 2,972.8	8,000
Constant rate of channel 1 (Experiment 4)	122,498.6	+/- 2,996.9	8,000
‘Low’ positive correlation between rates (Experiment 5)	107,831.7	+/- 2,426.1	10,000
‘Low’ Negative correlation between rates (Experiment 6)	137,377.9	+/- 3,504.0	7,000

We looked at the distributions associated with the experiments, which are presented in Figure 5. It turned out that the times to system failures are *stochastically ordered*: the ordering being the same as the ordering between the respective MTTFs (Table 2). Analyzing these distributions we established that in all 6 experiments they can be approximated very well using an exponential distribution with parameters equal to the reciprocal of the means defined in Table 2.

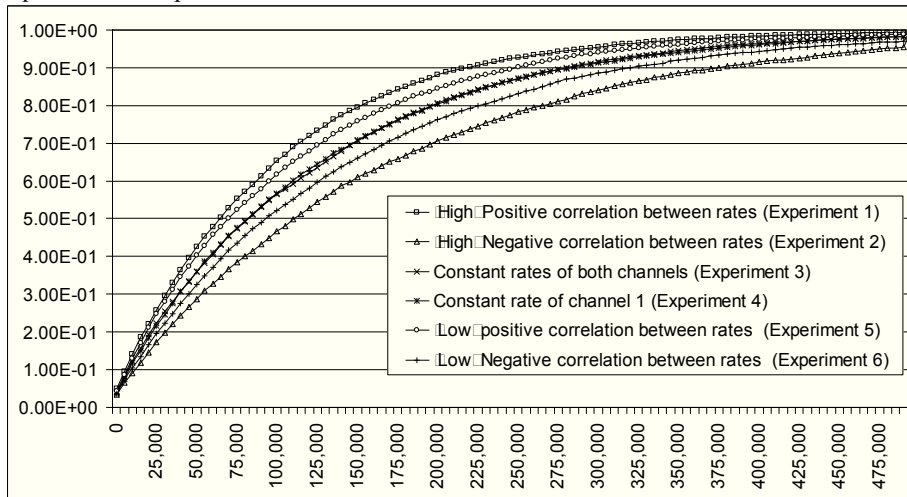


Figure 5. Distribution of the time to system failure, truncated after 500,000 time units of simulation.

We scrutinized further, via simulation, how the distribution of the activities associated with failures of the channels will affect the distribution of the time to

system failure. While the activities modeling the transitions between sub-domains were left exponentially distributed with a rate of 0.3 and fixed repair times of 0.01 were used, as before, we set the activities modeling the time to a channel failure to have Weibull and Gamma distributions with parameters which lead to non-constant hazard rate. The parameters were chosen in such a way that the transitions between the partitions remained significantly more frequent than the channel failures. The distributions of the activities did affect the time to system failure very significantly. The effect that we highlighted above, however, remained in place: negatively correlated rates would lead to longer times to system failure than constant rates, which in turn were longer than if the rates of failure of the channels were positively correlated. We observed that the MTTF may differ up to an *order of magnitude* between positively and negatively correlated rates. The time to system failure in all simulated cases remained exponential despite the significant differences in the rates.

4. Littlewood□ semi-Markov model of software reliability

Littlewood studied [6] systems with modular structure. The structures he considered were defined by the software modules (functions, procedures, etc.) of which the software consists. He assumed that the failures of the software can happen within a module or during the invocation of a module by another module.

Littlewood's reasoning is based on *three essential assumptions*:

- the underlying stochastic process describing the software behavior is semi-Markov. The time that software spends in a module (the sojourn time) can have an arbitrary distribution, but the transition probabilities between the modules are *constant*.
- while occupying a module the program may fail randomly with a *constant failure rate*.
- the transition probabilities between the modules are *significantly greater* than the rates of failure (either within the module or during the module invocation). Otherwise the software would have been, Littlewood argues, very unreliable.

Under these assumptions Littlewood proved analytically that the failure process will be a Poisson process. Its parameter can be computed from the steady-state probabilities of the embedded Markov chain (after eliminating the failure state, which in his description was absorbing), the mean times the program spends in a module and the small failure rates.

How does the model described by Littlewood differ from the one used by us to model the behaviour of a 2-channel control system?

The first assumption by Littlewood is clearly sufficiently general to apply to our model too. Our model is a model of a semi-Markov process, too. Although we describe the model in terms of transitions between the partitions and ignore the internal structure of the software (functions, procedures, etc.) the model is conceptually very similar: there are states represented by the partitions and transitions between these states. In each state our model is strictly a model of competing risks – the shortest activity defines the next system state. However, one can easily transform

the competing risk model with states to a semi-Markov process. Indeed, one can directly express the sojourn time as a function of the distributions chosen for the activities [7]. The marginal probabilities that the random variable representing a particular competing risk will be the shortest one can easily be derived, too (see the Appendix for details). These probabilities will form the transition probabilities for the embedded Markov chain associated with the semi-Markov model. In summary, the first assumption of the Littlewood theorem is satisfied.

The second assumption, however, is not always satisfied. For exponentially distributed activities representing the channel failure, the assumption is satisfied, but for Weibull and Gamma distributed activities – it is not. Thus, our model formally violates the second assumption made by Littlewood, that failures occur randomly.

The third assumption made by Littlewood is also plausible in our case: as evident from the used parameterization, the transition probabilities to all non-absorbing states are significantly more frequent (including the repair) than the transition to the absorbing state of failure of both channels³.

Despite the violation of the second assumption made by Littlewood, his asymptotic result seems to apply: asymptotically the time to system failure is exponentially distributed. It is outside the scope of this paper to offer an analytic explanation why this is the case, a problem worth addressing in the future.

5. Discussion

The effect reported in this paper, that the correlation of failure rates over partitions of the input space matters, is not surprising. Similar effects, that variation of the probability of failure, conditional on partitions, have been studied extensively in the past for on-demand systems [8].

The practical implications of the work presented here seem significant. If one is to measure the marginal rates of failure of the channels and then use these to estimate, e.g. by simulation, how the speed of recovery will impact the reliability of the system one will be implicitly assuming the situation described by our example 3 – no variation of the failure rates of both channels. But such an evaluation may be incorrect

³ There is a subtle difference between a semi-Markov process and the model of competing risks which is rarely discussed in the literature. For the competing risks model the transition probabilities are proportional to the respective hazard rates (of the random variables representing the competing risks), while in the semi-Markov process it is typically assumed that an embedded Markov chain exists with fixed transition probabilities. If all competing risks are exponentially distributed, then the hazard rates are constant and so are the transition probabilities – they remain the same irrespective of the length of the sojourn time. However, if at least some of the competing risks are not exponentially distributed then the hazard rate may vary over time (e.g. with Weibull distribution it may increase or decrease) and thus it becomes dependent on the duration of the sojourn time (see the Appendix for further details). This subtle difference, however, does not seem to matter, at least not for our studies. Despite exploring a wide range of scenarios (with activities assumed to have Gamma and Weibull distributions) the distributions of the times to system failure remained exponentially distributed.

– it may be optimistic or pessimistic depending on the variation of the rates of failure in partitions. Results based on ignoring the variation of the failure rates will only be useful if one can demonstrate that at least one of the channels fails with the same failure rate in all partitions (as in experiment 4 in Table 2). Constant failure rate over the partitions, however, *does not seem realistic*. For various reasons the partitions are likely to be subjected to different scrutiny – some are less critical than others, or are less used by the users and hence problems are less likely to be reported, etc. The point in the end is that, ignoring the effect we report here may lead to overestimation or underestimation of system reliability and it is *impossible to know in advance* even the sign of the estimation error one will make by ignoring the correlation between the rates over the partitions. Overestimating system reliability may be dangerous, while underestimating may lead to waste of resources – e.g. insisting on further V&V to improve system reliability.

The model presented above assumes that the channel failure processes are independent processes. This assumption can be relaxed. An example is the model of ‘common stress’, which causes both channels to fail simultaneously, e.g. due to a specification fault. A useful and widely used example is the Marshall and Olkin model [9]. The joint *pdf* of the channels’ time to failure is defines as follows:

$$f(x, y) = \begin{cases} \theta_1(\theta_2 + \theta_3)e^{-\theta_1 x - (\theta_2 + \theta_3)y}, & \text{if } x < y, \\ \theta_2(\theta_1 + \theta_3)e^{-\theta_2 y - (\theta_1 + \theta_3)x}, & \text{if } y < x, \\ \theta_3 e^{-(\theta_1 + \theta_2 + \theta_3)y}, & \text{if } x = y, \end{cases}$$

where $x > 0$, $y > 0$, $\theta_1 > 0$, $\theta_2 > 0$, and $\theta_3 > 0$. X and Y are the lifetimes of the two channels subjected to three kinds of shocks, assumed to be governed by three independent Poisson processes with parameters θ_1 , θ_2 , and θ_3 , respectively. Shock 1 applies to channel 1 only, shock 2 applies to channel 2 only, while shock 3 applies to both channels (hence ‘common stress’). The model has been used widely in nuclear reactor safety, competing risks reliability, etc [10]. The marginal distributions of X and Y are exponential distributions with parameters $\theta_1 + \theta_3$ and $\theta_2 + \theta_3$, respectively.

Clearly we could easily integrate the Marshall and Olkin model in the model presented in section 3 by adding a third activity, from place OO (Figure 4) to the absorbing place FF, to model the common stress, i.e. shock 3.

Accurately measuring system reliability will require more detailed measurement, which includes the following steps:

- estimating the probabilities of the partitions (the partition profile). Provided sufficient statistical testing is undertaken, one could easily arrive at a very accurate estimate of the probabilities of partitions. With more care one can even measure directly the transition rates between the partitions, which will be used directly in the model;
- estimating the rates of channel failure in partitions. This may require more effort, than measuring the partition profile, especially in case of very reliable channels. Failure rates can be estimated from the log of observed channel failures.
- relaxing the assumption that the channels fail conditionally independently will further complicate the measurements. Now one will need to quantify the strength of dependence between the channel failure processes.

Developing in details techniques for parameters estimation is beyond the scope of this paper. We notice in passing that the theory of competing risks is well developed and has been applied successfully in a wide range of applications. In our model every place (or state of the system) is associated with a set of competing risks – several activities compete to move the system to one of the states reachable by a single activity. For instance, in an OO place (Figure 4) several risks (represented by their respective activities) compete – to move the system to a new partition or to a state in which one of the channels has failed (or both in case the model is extended to include a common stress). The parameters associated with the risks may be unknown with certainty and need to be estimated from the available observations. The Appendix provides details, including the likelihood of any possible observation, sufficient for parameter estimation – either by maximizing the likelihood of the observations or by applying Bayesian inference. The point here is that every time a transition from a state takes place, we collect an observation associated with the realization of the competing risks defined for this place. Given the assumed Markov structure of the model, estimating the model parameters will consist of independent data collection and estimation of the model parameters associated with the individual states.

We note that estimating the parameters of different parts of the model can be done using different techniques. For instance, we can obtain the parameters of the transitions (activities) between the OO states of the partitions directly from the observations (as these will be likely to be frequent and many realizations can be observed in a short period of observation). The parameters (i.e. distributions) of the activities OOtoFO or OOtoOF in turn can be assessed using Bayesian inference (given the typically small number of observations one can collect within a limited statistical testing or operational exposure). Once the estimation of the parameters of OOtoFO and OOtoOF activities is done, one can use these to parameterize the activities FOtoFF and OFtoFF, as in the model we assume them to have the same parameters as OOtoOF and OOtoFO, respectively.

Once the parameters of the model activities are estimated, one could run a simulation experiment to measure directly the time to system failure. Further, as new operational data becomes available, one could revise the model (by re-assessing periodically the model parameters) and then re-run a new simulation to estimate the time to system failure.

6. Relevant Literature

Probabilistic models of on-demand fault-tolerant software have attracted significant attention. The original work by Ekhardt and Lee [11] demonstrated that failure independence is unlikely for even independently developed software versions (channels of a fault-tolerant system). The reason for this is that the individual demands processed by software may differ in their “difficulty”, i.e. they will be problematic to independent developers and the chances of simultaneous failures on these demands of independently developed channels are greater than what would be expected assuming independent failures. This was a very important insight, which affected the research and practical adoption of software fault-tolerance. The model by

Ekhardt and Lee was extended by Littlewood and Miller [12], to the case of forces design diversity (e.g. different teams are forced to use different development methodologies which lead to different difficulties of the demands). This work demonstrated the possibility for achieving system reliability better than assuming failure independence between the channels. Popov and Littlewood [13] extended the earlier models by allowing the channels reliability to grow, e.g. as a result of testing and compared the effect on system reliability of different testing regimes – testing the channels in isolation, testing them together on the same testing suite and back-to-back and ranked these testing regimes according to their impact on system reliability.

These models were models “on average” – they modeled the process of software development of fault-tolerant software as a random selection from populations of versions, which hypothetically can be developed to a given specification. The models, however, do not address the issue of assessing the reliability of a particular fault-tolerant system. This problem was addressed in [8]: the authors developed a model of a fault-tolerant software operating on demand space with partitions and demonstrated that it can be used for *practical assessment* by establishing bounds on the probability of system failure based on estimates of the probabilities of failure of the channels in the partitions only, which are typically estimable.

The models summarized above are applicable to on-demand software only, i.e. in which the individual demands are drawn independently from the demand space. Another line of research addressed the characteristics specific for control software, e.g. the fact that control software is typically executed on trajectories of inputs, which are not independently drawn from the input space. An important implication of trajectory based execution is failure clustering due to the fact that failure regions usually occupy ‘blobs’ of individual inputs, [14] , [3]. Modeling explicitly failure clustering was done in a number of studies, e.g. [15].

7. Conclusion and Future work

We present a model of system reliability of a 2-channel control software operating over partitions of the input space. The failure rates of the channels may vary over the partitions. The model reveals a useful insight – the probability of system failure may be significantly affected by the correlation of the failure rates of the channels over the partitions – we recorded up to an order of magnitude difference in the mean time to systems failure between assessment ignoring the effect of failure rate variation and taking it into account. The result seems important because it suggest the need for more accurate reliability measurement than is currently undertaken.

We further considered a model of reliability for software with modular semi-Markov structure developed by Littlewood in the past and established that our model, although generally very similar, deviates from the mathematical description provided by Littlewood. Despite the deviations, however, similarly to Littlewood, we observed that the time to system failure is exponentially distributed. Providing an analytical proof for the cases when the failures in partitions are not random (i.e. do not occur as Poisson processes) or identifying the cases when the system failure process ceases to be a Poisson process itself, is an open research problem.

We also discuss the issue of model parameter estimation. The theory of competing risks offers a suitable framework for parameter estimation using either maximum likelihood or Bayesian inference. Developing detailed assessment techniques with illustrative examples to help practitioners will be addressed in the future.

References

1. Littlewood, B. and L. Strigini, *Validation of Ultra-High Dependability for Software-based Systems*, Communications of the ACM, 1993. **36**(11): p. 69-80.
2. Littlewood, B., P. Popov, et al. *Design Diversity: an Update from Research on Reliability Modelling*, in *Safety-Critical Systems Symposium 2001*. 2001. Bristol, U.K.: Springer.
3. Ammann, P.E. and J.C. Knight, *Data Diversity: An Approach to Software Fault Tolerance*, IEEE Transactions on Computers, 1988. **C-37**(4): p. 418-425.
4. Bondavalli, A., S. Chiaradonna, et al. *Dependability Models for Iterative Software Considering Correlation among Successive Inputs*, in *IEEE International Symposium on Computer Performance and Dependability (IPDS'95)*. 1995. Erlangen, Germany.
5. PERFORM, *Möbius: Model Based Environment for Validation of System Reliability, Availability, SEcurity and Performance. User's Manual, v. 2.0 Draft*. 2006.
6. Littlewood, B., *A Semi-Markov Model for Software Reliability with Failure Costs*, in *MRI Symposium on Computer Software Engineering*. 1976, Polytechnic Press (Available from Wiley, London): Polytechnic of New York, New York. p. 281-300.
7. David, H.A. and M.L. Moeschberger, *The theory of competing risks*. Griffin's Statistical Monographs & Courses, ed. D.S.E. Prof. Alan Stuart. Vol. 39. 1978. 103.
8. Popov, P., L. Strigini, et al., *Estimating Bounds on the Reliability of Diverse Systems*, IEEE Transactions on Software Engineering, 2003. **29**(4): p. 345-359.
9. Marshall, A.W. and I. Olkin, *A generalised bivariate exponential distribution*, Journal of Applied Probability, 1967. **4**: p. 291-302.
10. Nadarajah, S. and S. Kotz, *Reliability for Some Bivariate Exponential Distributions*, Mathematical Problems in Engineering, 2006. **2006**: p. 1-14.
11. Eckhardt, D.E. and L.D. Lee, *A theoretical basis for the analysis of multiversion software subject to coincident errors*, IEEE Transactions on Software Engineering, 1985. **SE-11**(12): p. 1511-1517.
12. Littlewood, B. and D.R. Miller, *Conceptual Modelling of Coincident Failures in Multi-Version Software*, IEEE Transactions on Software Engineering, 1989. **SE-15**(12): p. 1596-1614.
13. Popov, P. and B. Littlewood. *The Effect of Testing on Reliability of Fault-Tolerant Software*, in *Dependable Systems and Networks (DSN'04)*. 2004. Florence, Italy: IEEE Computer Society Press.
14. Bishop, P.G. and F.D. Pullen. *PODS Revisited - A Study of Software Failure Behaviour*, in *18th International Symposium on Fault-Tolerant Computing*. 1988. Tokyo, Japan: IEEE Computer Society Press.
15. Bondavalli, A., S. Chiaradonna, et al., *Modelling the effects of input correlation in iterative software*, Reliability Engineering and System Safety, 1997. **57**(3): p. 189-202.

Appendix

The material and the notation used here are based on [7].

Let C_l ($l = 1, \dots, k$) denote the k competing risks or causes of failure. Let the random variable Y_i denote the individual length of life if Y_i were the only risk present with cdf $P_i(x) = \Pr\{Y_i \leq x\}$ and pdf $p_i(x)$. When all risks are present, we can only observe the random variable, Z , defined as follows: $Z = \min(Y_1, \dots, Y_k)$.

Clearly, if Z exceeds x , then every Y_i exceeds x , too, i.e.:

$$\Pr\{Z > x\} = \Pr\{Y_1 > x, \dots, Y_k > x\}, \text{ which we denote as } \overline{F_Z}(x) = 1 - F_Z(x).$$

An important characteristic is the *hazard* rate defined as: $r_Z(x) = \frac{f_Z(x)}{F_Z(x)}$. The hazard

rates of the individual competing risks are defined similarly: $r_i(x) = \frac{p_i(x)}{P_i(x)}$

For the case of independent risks, the total hazard rate is equal to the sum of the hazard rates of the competing risks: $r_Z(x) = \sum_{i=1}^k r_i(x)$.

Let π_i be the probability that a failure is caused by risk C_i .

A related measure is the conditional probability $\Pr\{Y_l = \min(Y_1, \dots, Y_k) \mid Z = x\}$, which is defined by the ratio $\frac{r_i(x)}{r_Z(x)}$. If this ratio is a constant (so called proportional hazard rates) the probability does not depend on the value of x and is equal to π_i . But this is not always the case and in the general case $\frac{r_i(x)}{r_Z(x)} \neq \pi_i$.

If N_i individuals fail from cause C_i , and X_{ij} denotes the lifetime of the j -th individual failing from cause C_i ($j=1, \dots, n_i$; $i = 1, \dots, k$), then the joint pfd of the X_{ij} is:

$$f(x_{11}, \dots, x_{1n_1}, \dots, x_{k1}, \dots, x_{kn_k}) = \prod_{i=1}^k \frac{1}{\pi_i^{n_i}} \prod_{j=1}^{n_i} p_i(x_{ij}) \prod_{\substack{l=1 \\ l \neq i}}^k \overline{P}_l(x_{ij}).$$

This is conditional on the random variables, $N_i = n_i$ ($i = 1, \dots, k$), which have multinomial distribution: $f(n_1, \dots, n_k) = \frac{n!}{\prod_{i=1}^k n_i!} \prod_{i=1}^k \pi_i^{n_i}$, where $n = \sum_{i=1}^k n_i$. Hence, the

likelihood function of interest is: $L = \frac{n!}{\prod_{i=1}^k n_i!} \prod_{i=1}^k \prod_{j=1}^{n_i} p_i(x_{ij}) \prod_{\substack{l=1 \\ l \neq i}}^k \overline{P}_l(x_{ij})$.

This expression is sufficient for one to apply either maximum likelihood for parameter estimation associated with the individual risks, Y_i , or Bayesian inference directly to the distributions of Y_i .

An open-source application to model and solve dynamic fault tree of real industrial systems

Ferdinando Chiacchio¹, Lucio Compagno², Diego D'Urso², Gabriele Manno¹ and Natalia Trapani²

¹ Dipartimento di Matematica e Informatica, Università degli Studi di Catania (ITALY)

² Dipartimento di Ingegneria Industriale e Meccanica, Università degli Studi di Catania (ITALY)

In recent years, a new generation of modeling tools for the risk assessment have been developed. The concept of "dynamic" was exported also in the field of reliability and techniques like dynamic fault tree, dynamic reliability block diagrams, boolean logic driven Markov processes, etc., have become of use. But, despite the promises of researchers and the efforts of end-users, the dynamic paradox hangs: risk assessment procedures are not as straight as they were with the traditional static methods and, what is worse, it is difficult to assess the reliability of these results.

Far from deny the importance of the scientific achievement, we have tested and cursed some of these dynamic tools realizing that none of them was appropriate to solve a real case. In this context, we decided to develop a new DFT reliability solver, based on the Monte Carlo simulative approach. The tool is greatly powerful because it is written with Matlab® code, hence is open-source and can be extended. In this first version, we have implemented the most used dynamic gates (PAND, SEQ, FDEP and SPARE), the existence of repeated events and the possibility to simulate different cumulative distribution function of failure (Weibull, negative exponential CDF and constant). The tool is provided with a snappy graphic user interface written in Java®, which allows an easy but efficient modeling of any fault tree schema. The tool has been tested with many literature cases of study and results encourage other developments.

Index Terms Reliability, Dynamic Fault Tree, Monte Carlo simulation, Parallel Computing, Continuous time Markov chains.

I. INTRODUCTION

In recent years the importance of risk assessment in the industrial field has significantly increased and the most used tools of RAMS — the well known combinatorial techniques such as Reliability Block Diagram (RBD) and Static Fault Tree (SFT) — have been object of a reasonable criticism. In fact, these techniques are not able to model the time-dependent behaviors of a system (or a process), like the replacement of spare components, a chain of events and so on.

State-space models have been used in order to overcome the previous limits and, on the wave of their success, other formalisms were proposed, such as:

- DFT (Dynamic Fault Tree) [1];
- DRBD (Dynamic Reliability Block Diagrams) [2];
- BDMP (Boolean logic Driven Markov Process) [3].

The declared aim was to combine the intuitive symbolic representation of the combinatorial methods with the powerful modeling of the state space models: these new techniques were named "dynamic" and, conversely, the combinatorial ones were referred as "static". Among these new formalisms, the dynamic fault tree analysis (DFT) has been one of the most promising [4] because, with the introduction of the dynamic gates, the power of the modeling increases but the effort of the designing is kept low (like in static fault tree).

The reason of our interest in this technique is that most of the risk assessments in the industrial field are still based on the static fault tree, hence a re-examination of such risk reports with the aid of the DFT could be a valuable work. The attempt to apply the DFT technique to some existing reliability schema typical of the industrial field is discussed in [5], where the inadequacy of the traditional analytical methods is shown for complex systems, the limits of the most known automated software of reliability are highlighted and the lines for the development of a simulative algorithm in a spreadsheet environment are traced (ie. Excel®).

Starting from the results [5], in this paper we present a software tool for the resolution of complex DFT model, based on the direct Monte Carlo simulative approach [6].

The paper is organized as follow: in the second section we introduce the DFT technique, briefly mentioning the resolution procedures and the limits of the most used tools; in the third paragraph we describe the simulative approach for the fault tree resolution and in the fourth we introduce our Monte Carlo engine (*MatCarloRE*), implemented within the Matlab® framework; in the fifth paragraph, the Java® graphic user interface (GUI) (*jDFTDes*, which stands for java Dynamic Fault Tree Designer) that embeds the *MatCarloRE* library is presented and in the sixth section some results are discussed and compared with the enhancing of the parallel computing approach. In the end conclusion are drawn.

II. THE DYNAMIC FAULT TREE TECHNIQUE

A Dynamic Fault Tree (DFT) is a stochastic model for the reliability evaluation that synthesizes the ways how an undesired and time dependent event can occur. As a Static Fault Tree (SFT), a DFT is composed by a top gate which represents the most undesired event (TE, top event) and a certain number of lower level gates and basic events (BEs) that, combined according with the logic of the fault scenario, cause the occurrence of the TE. The main hypotheses for the use of the DFT are that (i) events are binary and (ii), according to many authors [7-9], components are not repairable. Thus, the main difference with the SFT is that DFT were not conceived to compute the availability but are appropriate to evaluate the reliability of model characterized by complex stochastic dependencies (Fig. 1).

The possibility to model complex interactions with the graphical symbolism of the SFT has encouraged the development of dynamic models [10] but, in point of the fact, DFT has shown many issues for what concern their resolution.

The reason of this anomaly has to be traced in the lack of a rigorous semantic language [11] that has caused the proliferation of several and variegated techniques of resolution that resort to an equivalent stochastic model [1, 11-13]. At the state of the art, an analytical solution exists only if another hypothesis is added to the previous ones [9]: BEs have to be described by the exponential distribution. In this way, it is possible to convert a DFT into a state-space model and solve it within the domain of the Markov processes. Unfortunately the mentioned hypotheses can result too restrictive, especially for real industrial applications characterized not only by exponentially distributed time to fail but also by Weibull, gaussian or lognormal probability distribution. Therefore, the reliability evaluation of systems that present generalized functions of probability is not possible with the analytical Markov processes and, at the state of the art, the more effective solution is the simulation [14, 15].

Name	Graphical Representation	Description (N input)
SPARE		It triggers only after the primary if all the N spares occur. Spares can be shared with other spare gate.
PAND		It behaves like an AND gate but it triggers only if the input events occur in the order from the left to the right.
SEQ		It forces the input events to occur from the left to the right order. It can model the gradual degradation of a system.
FDEP		This gate models the failure of the dependent input events if the primary occurs. The output is a dummy.

Fig. 1: the most frequently used dynamic gates

In general, all the previous techniques of resolution (analytical and simulative) have been implemented in a lot of software applications for reliability analysis [16, 17] but, despite that, the real effectiveness of these tools is still questionable because none of them can be used to design and solve "complex DFT" in a straightforward manner. In Table 1, we synthesize the main features of some automated tools that we have tested; for more details about their characteristics we remind to [5].

Table 1: main features of the reliability automated tools for DFT

TOOL	MAIN RESULTS	LIMITS
RELEX	Reliability	Results are questionable
	Availability	NO nested dynamic gates
	Importance Measures	Only exponential CDF
GALILEO	Reliability	NO repeated events
	Sensitivity	
BDMP	Reliability	Not intuitive
	Availability	(introducing other formalism)
	Importance Measures and Sensitivity	

In our scope, a "complex DFT" is a model that presents a combination of the following characteristics: repeated events, events characterized by generalized distributions of probability and nested dynamic gates. All of these elements are necessary to obtain an accurate modeling of the failure scenarios in real industrial systems.

III. MONTE CARLO SIMULATION IN RELIABILITY ASSESSMENT

Monte Carlo simulation is a statistical method used to solve real problems in many engineering field, in particular when analytical approaches are not feasible. This method is based on the generation of a large number of realizations of the simulated process, which represent the generic random walk inside a discrete "phase space" of the system configurations. In this method, a certain number of stochastic sampling (called iterations, runs or batches of the simulation) of the independent variables are performed in order to implement the simulated process; for this reason, a Monte Carlo simulation cannot produce an exact evaluation as it is computed as a weighted average among the results of the generated phase space.

Nowadays, this class of methods is gaining a lot of interest due to the power of modern computers that permit to speed up the simulative process and collect a large number of runs, ensuring higher accuracy. For reliability evaluations [18, 19], the main advantage of the simulative approach over the analytical ones is that it is possible to remove the hypothesis of the exponential distribution and, more in general, to implement any kind of failure and safety logic. The other important difference concerns the resolution process: with the simulative paradigm the computation of a TE needs the information from the entire parts of the fault tree (ie. single components and sub-models), whereas the analytical solutions mix these dynamics in a set of ordinary differential equations. Hence, at the end of a simulation it is possible to retrieve also information about the other parts of the fault tree (BEs and intermediate events), whereas with the analytical methods this would need an ad-hoc computation for the sub-system of interest.

In our approach, we make use of the direct Monte Carlo simulation [6] that considers BEs as single entities, sampling the time of failure of any BE once, at the beginning of any iteration. These information are used to evaluate the logic of the gates which the BEs are connected with, in order to retrieve the state and the trigger time of each gate. In turn, these information are passed to the gates of the higher levels, ascending the fault tree up to the TE gate. From the implementation point of view, the main advantage of this approach is the modularity, as the algorithms behind the logic of the gates (static and dynamic) can be implemented separately.

In order to build up a Monte Carlo simulation, let us consider the following notation:

- 1) FT is the generic fault tree composed by a number n of BEs; at the beginning of any iteration, the method will sample n stochastic time of failure t_i^f , one for each BE _{i} ;
 - 2) $F(t)$ is the unreliability of the system at the time t , computed as the probability that the system is down at the time t , $F(t) = P(\text{system is down})$;
 - 3) $S = (s_1, s_2, \dots, s_n)$ is the generic vector of the states where s_i represents the state of the i -th component that can be working (=0) or failed (=1) and represents a point of the phase space of the system;
 - 4) Φ is the set of all the vectors of the states S in which the system fails and depends on the structure of the fault tree;
 - 5) $H_k = \{S(t_0); S(t_1); \dots; S(T_m)\}$ is the evolution of the k -th iteration of the Monte Carlo simulation; in general, the evolution H_k of an iteration can be characterized by a number of different vectors of the states S ; in fact, these vectors are determined with respect to the stochastic times of failure generated at the beginning of the Monte Carlo sampling.
- For instance, let consider the simple SFT of Fig. 2 and assume a Monte Carlo simulation of K iterations and a mission time $T_m = 10\text{h}$. In this case $\Phi = \{(1_A, 1_B, 1_C)\}$, no matter the sequences of failures of the components.

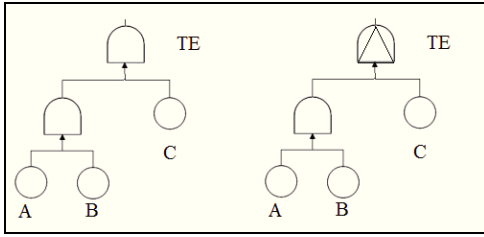


Fig. 2: a simple example of SFT and DFT

Now, let us consider the generic i -th iteration ($i < K$) and assume the following random sampling:

$t_A^f = 5\text{h}$; $t_B^f = 9\text{h}$; $t_C^f = 8\text{h}$. In this case we can consider the following vectors of the states:

$S(t_0) = (0_A, 0_B, 0_C)$; $S(t_A^f) = (1_A, 0_B, 0_C)$; $S(t_B^f) = (1_A, 1_B, 0_C)$; $S(t_C^f) = (1_A, 1_B, 1_C)$; $S(T_m) = (1_A, 1_B, 1_C)$; therefore $H_1 = \{S(t_0); S(t_A^f); S(t_C^f); S(t_B^f); S(T_m)\}$. In this case, there is a vector of the states S in H_1 which belongs to Φ , $S(t_B^f)$, therefore the TE occurs.

Let us assume that the j -th ($j < K$) iteration is characterized by the following sampling:

$t_A^f = 11\text{h}$; $t_B^f = 9\text{h}$; $t_C^f = 15\text{h}$. In this case we have: $S(t_0) = (0_A, 0_B, 0_C)$; $S(t_B^f) = (0_A, 1_B, 0_C)$; $S(T_m) = (0_A, 1_B, 0_C)$. What happens after T_m is out of the scope of the Monte Carlo simulation therefore, for this sampling, $H_2 = \{S(t_0); S(t_B^f); S(T_m)\}$ and none of the $S(t_i)$ is contained in Φ . For this iteration, the TE does not occur.

In order to speed up the simulative algorithm, it is possible to trim the iterations by checking at any transition whether the state $S(t_i)$ belongs or not to Φ : if this condition is verified the batch ends and the TE occurs, otherwise it continues until T_m is reached.

At the end of the iterations, the unreliability of the fault tree is computed with the following:

$$F = \frac{\sum_{i=1}^{iter} F_i}{iter}$$

where $iter$ is the number of iterations of the Monte Carlo

simulation and F_i is the occurrence of the TE during the i -th batch. F_i will assume the value 1 if the TE occurs, 0 otherwise.

This algorithm can be used effectively for the resolution of a DFT because the evolution H is able to keep track of the sequence of failures simulated during an iteration of the Monte Carlo simulation; in this way the time dependencies of a DFT can be taken into account for the correct evaluation of the dynamic logics. For instance, let us consider the DFT of Fig. 2; also for this DFT the $\Phi = \{(1_A, 1_B, 1_C)\}$, but the way how this configuration is reached matters. In fact, in this case, the evolution H generated for the previous example ($H = \{S(t_0); S(t_A^f); S(t_C^f); S(t_B^f); S(T_m)\}$) would not reach to a failure state, due to the presence of the PAND gate.

The other important features of this simulative approach is that with the use of the H , many simulation data can be stored in a very straightforward way: for each BE and gate we can count the number of triggering, the mean time to failure and easily infer other measures.

IV. THE SIMULATIVE ENGINE OF MATCARLORE

MatCarLoRE is the library of functions that constitute the Monte Carlo simulative engine for the resolution of DFTs reliability. It has been implemented in the Matlab® environment following the direct Monte Carlo paradigm [6]. The library is composed by a set of .m files such that, in order to make the library as flexible as possible, the logic of any gate is implemented in different file. In this way it is possible to modify these files separately without compromising the other logics. Any of these files represents a Matlab® function that is invoked inside the code of the fault tree model.

In this current release of the library, it is possible to compute the unreliability (F) of the system at a fixed instant of time or in a discrete time interval and use all the measures retrieved inside the Matlab® environment for further studies.

In the next sections we will discuss the most important functions, focusing in particular on the logic of the dynamic gates [11, 15].

A. The «BE function»

The BE function is the core of the Monte Carlo engine because it samples the times of failure of the BEs, retrieving if the BE has failed before the time of mission (Fig. 3).

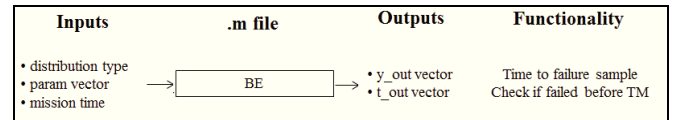


Fig. 3: specifications of the BE function

So far, the «BE function» can model basic events with a negative exponential distribution, with a fixed probability or with a Weibull distribution. The code can be easily customized to define any type of probability distribution. In fact, the characteristic time of failure t^{Fi} of the generic BE _{i} is sampled for any batch of the simulation through the inverse function: $t^{Fi} = CDF^{-1}(param)$, where CDF^{-1} is the inverse of the cumulative distribution function and $param$ is the vector which contains the parameters of the CDF.

For instance, let us consider a component characterized by an exponential distributed time to fail, $F(t) = 1 - e^{-\lambda t}$, where λ is the failure rate of the component. The sampled time of failure can be calculated by the inverse relationship:

$$t^{Fi} = -\frac{\log(F^*)}{\lambda}$$

where F^* is a random number in $[0, 1]$ generated with a uniform distribution of probability. If t^{Fi} is smaller than the mission time T_m , the component is assumed failed at the time t^{Fi} . These method has to be invoked for any BE of the fault tree.

B. The «PAND function»

The «PAND function» models the Priority-And gate: the specifications of the function are shown in Fig. 4, while the logic is reported in the flow chart of Fig. 5.

At first, the algorithm verifies if all the BEs of the gates (contained in the input vector \underline{y}_{in}) have occurred (the sum has to equal the number n of the inputs of the gate). If this condition is not satisfied the gate does not trigger. Otherwise, the following conditions are checked: if $t_i < t_j$ for each $i < j$, with $i, j=1,2,\dots,n$, the gate triggers with a time of failure equals to the maximum failure time of the inputs.

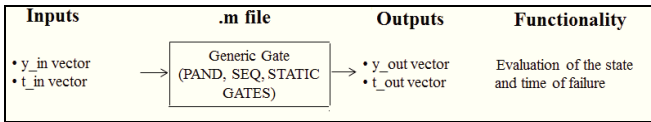


Fig. 4: specifications of the PAND function

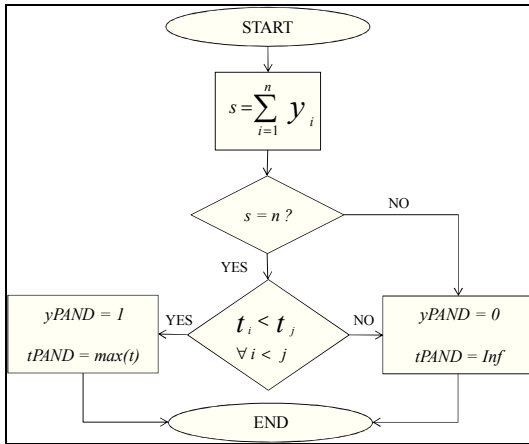


Fig. 5: flow chart of the PAND function.

C. The «SPARE function» and «ALL_SPARE function»

The «SPARE function» is the most complicated logic due to the variety of configurations allowed. The MatCarloRE library can handle cold, warm and hot stand-by (C/W/H Stand-By) with any number of active and spare parts. The algorithm is shown in the flow chart of Fig. 6, while the specifications of the function are shown in Fig. 7.

In this first release it is not possible to model extended DFT [20] therefore the inputs of a spare gate can be only BEs (and not other gates). The assumption made for the spare gate is that its failure occurs when the number of surviving components is less than the minimum number of components required for functioning (usually it is assumed equal to the number of the initial active components). Therefore, the first

operation performed is to sort in the ascending order the times of failure of the active components, in order to list which have failed before the end of the mission time. If no failure is verified the gate will not trigger, otherwise the algorithm checks if there are spare parts able to replace the failed active components following the order previously established. The replacement of an active component can take place only if:

1. the spare part is still available (namely, it has not been used to replace another failed component);
2. the time of failure of the spare part (during its latent condition) is greater than the time to failure of the active component to be replaced.

If these conditions are satisfied, the function updates the time of failure of the active component by adding the time of failure of the spare component that is finally declared as busy. Otherwise the spare gate triggers, setting a time of failure equal to the last failed active component.

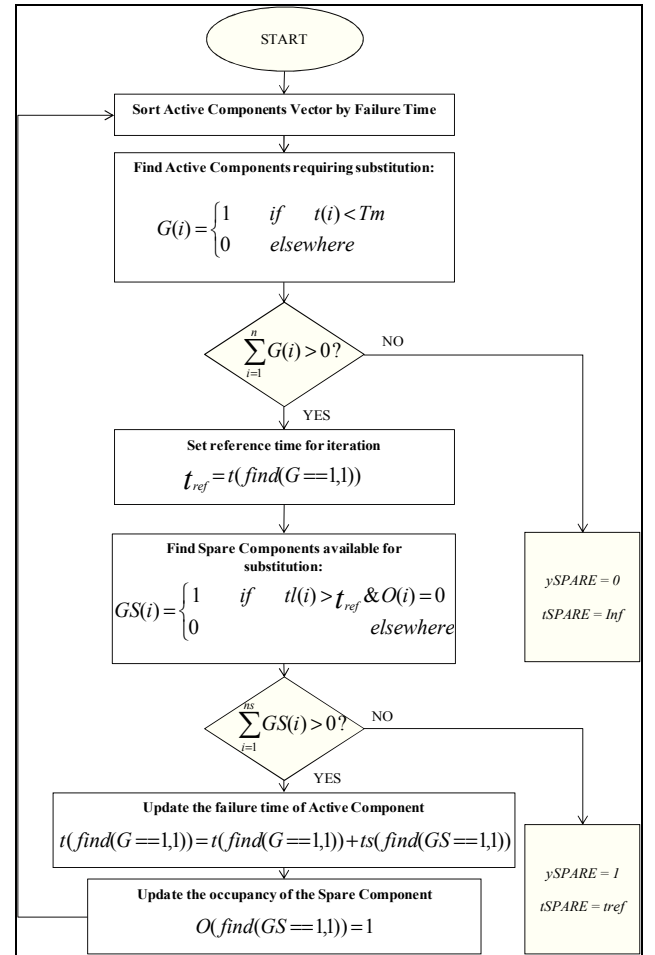


Fig. 6: flow chart of the SPARE function

The MatCarloRE makes use of another function called «ALL_SPARE» that takes all the spare gates as input (see Fig. 8). The output of the «ALL_SPARE function» is a vector that contains the state and the time of trigger of the spare gates passed. This function is important because it handles the allocation of the shared resources among the spare gates which have the same spare components.

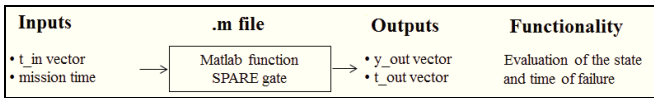


Fig. 7: specifications of the SPARE function

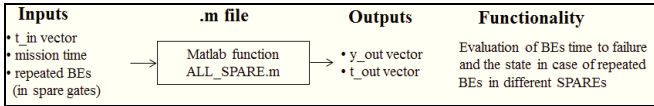


Fig. 8: specifications of the ALL_SPARE function

D. The «SEQ function»

The seq gate forces the components to fail in a fixed order [11, 15], from the left to right position. It is generally used to represent different levels of degradation of a component. The algorithm used for this task is simple: the «SEQ function» firstly calculates the sum Q of the times of failure of all the inputs. If Q is smaller than the mission time the gate triggers with a time of failure equal to Q .

E. The «FDEP function» and «OUT_FDEP function»

The fdep gate vehicles the effects of the primary input to its dependent components [11, 15] in a way that if the primary component fails all the dependent ones fail too, with the same time of failure.

MatCarloRE handle this gate with two function: the «FDEP» and the «OUT_FDEP». The former is invoked to compute the time of failure of the secondary inputs. This logic can be easily implemented because a fdep gate with k secondary inputs is equal to k or gates with two inputs: the primary and the i -th secondary component of the fdep gate. But, if a component is a secondary input of more than one fdep gate, the «OUT_FDEP» function (Fig. 9) must be invoked. In fact, in this case the real time of failure has to be chosen as the minimum among the time of failure of the correspondent fdep gates.

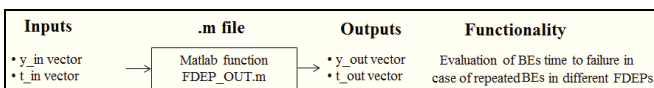


Fig. 9: specifications of the FDEP_OUT function

V. JDFTDES: A JAVA GUI FOR MATCARLORE

The creation of a DFT model with the MatCarloRE syntax can be error prone and tedious. In Fig. 10 it is shown the flow chart that describes how to prepare the Matlab® script. At first, the initialization of the main variables (F , number of iterations) is needed. Then, the simulation runs inside a loop that contains the commands that refer to the DFT model; the functions of the library have to be invoked in the order shown in Fig. 10, according with the structure of the DFT. The «BE functions» need to be invoked for first, in order to sample the correspondent times of failure of each BE; the «FDEP functions» and «OUT_FDEP functions» are called if the DFT contains fdep gates, in order to refresh the time of failure of the BEs which are connected with such kind of gate. Hence, if there are spare gates which share spare components, they have to be assembled invoking the «ALL_SPARE function». Once this first part of the code is written, it is possible to call all the other functions (which represent the

gates) following a bottom-up approach since the information of the lower level of the fault tree are requested to the upper gates. In the end, the unreliability is computed as the ratio between the number of TE occurrence over the number of iterations.

In order to simplify the effort of typing the code of a model with the MatCarloRE syntax, a graphic user interface (GUI), the jDFTDes, was developed. The jDFTDes is a code-processor that translates a graphic model of DFT in a program for the MatCarloRE engine. The jDFTDes stands for «Java® DFT Designer», a java package that can be invoked directly under the Matlab® shell. The choice of using Java was natural, since Matlab® runs under a Java Virtual Machine (JVM) and this permits to use the Java interpreter and run programs written in Java. In our application, we created a Java Archive (JAR), a Java file that includes all the classes of the jDFTDes. The jDFTDes can be invoked through the «javaaddpath(°)» command, specifying the path where the library is located and creating a dummy variable that contains an instance of the java main frame of the jDFTDes.

jDFTDes is greatly simple and the construction of a DFT model is straight, easy and fast. This was accomplished implementing a «drag and drop» interface that permits a quick interactions with the element of the DFT. Infact, by clicking the right button it is possible to change the property of a component (rename, modify the type of component and change the order of the dependency) while by clicking the left button is possible to add an input (if the component clicked is a gate) or specify the type of CDF if the component is a BE. The text field T_m must contain the value of the time mission and the text field «ITERATION» the number of batches requested for the simulation. Once the DFT is assembled and the input are correctly set, by clicking «COMPUTE» the jDFTDes will process the graphical model generating the code for the MatCarloRE and finally dumping it in the Matlab® shell.

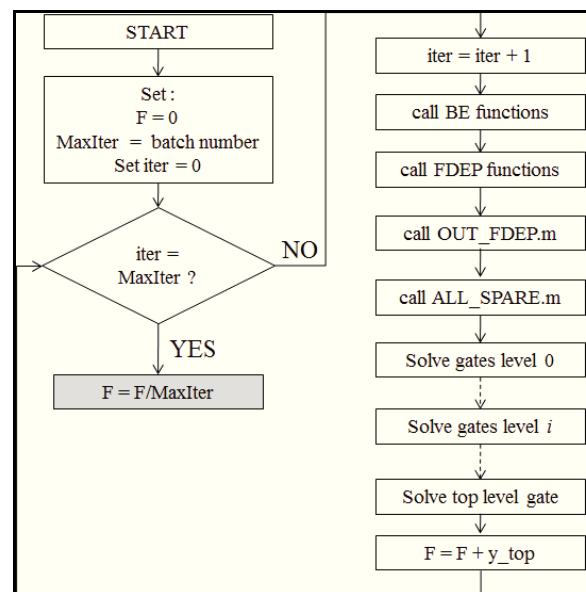


Fig. 10: flow chart to build a Matlab® script of a DFT model with the MatCarloRE syntax

VI. CASES OF STUDY: CONFIGURATION OF THE MULTI-PROCESSOR SYSTEM

In this section we present the implementation of a set of cases of study that refer to previous literature [5, 20].

The MatCarloRE library and Galileo® were tested using a standard 64bit laptop Intel® i7 CPU, Q740 @ 1.7 GHz with 6GB of Ram. As far as concerns the DFTSim (the simulative tool which is the closest benchmark for the MatCarloRE library), so far, we could not obtain any version of the tool, therefore we did consider the results taken from [20] which were obtained with a different system configuration (a Pentium 4 @ 3.2 GHz with 2GB of Ram). For this reason, a detailed comparison between the simulating tools was not possible.

Looking forward to get a version of the DFTSim, we configured our machine in order to make possible a comparison in term of results accuracy and time of execution between the MatCarloRE and DFTSim [20] (see also note 1 and Table 5 in the next section), starting from the following assumptions:

- the simulative engines of both MatCarloRE and DFTSim behave similarly, as they sample the time of failure for each BE and propagate the BEs failure times through the DFT;
- iterative computations (like Monte Carlo simulations) are mostly affected by the CPU clock and by the dimension of the second-level cache, whereas the extension of the Ram memory does not bring improvement if it can store entirely the amount of data processed (and this is the case we are dealing with).

Under the previous conditions, a system mounting two CPUs of 1.7 GHz, working in parallel and sharing the computation effort, can be compared with one only CPU of 3.2 GHz [20], as they can process approximately they same number of operations per unit of time. In Matlab® it is possible to enable the use of the multi-processors platform through the command *matlabpool*; clearly, the multi-processors is effective only if the source code of the script is compliant with the rules of the parallel programming. MatCarloRE can be easily customized to run in a parallel environment, just replacing the traditional *for* instruction with the *parfor*, standing for \square parallel for \square [21].

A. The Cascaded PAND System (CPS)

This is a hypothetical case of study of a cascade of dynamic PAND gates, which results of interest for the comparisons between the simulative approach and the analytic one. In fact, due to the state space explosion Relex® got stuck, while Galileo® took ten times the time needed with the simulative approach to solve it. MatCarloRE converges within few seconds. All the BEs have a failure rate of 1 [h⁻¹].

B. The Multi-processor Distributed Computing System (MDCS)

The main feature of the model of the MDCS (Fig. 11) is the shared W/stand-by spare memory (M3) between the subsystems Mem1 and Mem2. Table 2 shows the parameters of the components of the MDCS.

Table 2: failure rates of the BE of the MDCS in [h⁻¹]

Component	N	P1,P2	D11,D12,D21,D22	M1,M2,M3
Rate	0.00002	0.005	0.8	0.0003

C. The Cardiac Assist System (CAS)

The CAS system is a model more complex than the previous ones, as it is composed by many dynamic gates. Fig. 12 shows the screenshot of the jDFTDes application: the red circles denote the same sub-systems (the Pumps) and the jDFTDes needs to draw the repeated spare PS twice (in orange).

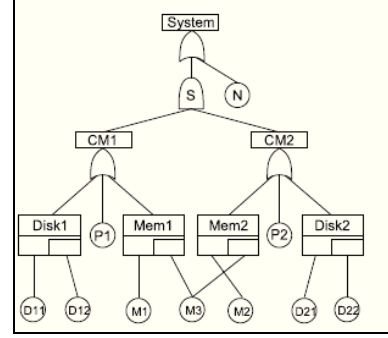


Fig. 11: DFT of the MDCS [20]

Table 3 shows the input parameters of the BEs; moreover, B is in a warm stand-by with a dormancy factor of 0.5, while PS and MB are in cold stand-by.

D. The Section of an Alkylation Plant (SAP)

This case of study [5] represents the dynamic version of a real SFT of an alkylation plant (Fig. 13). For this example, analytical modeling is not appropriated due the presence of the fixed probability (BE1 and BE3), while DFTSim could not be tested due to the impossibility to obtain a version of the tool.

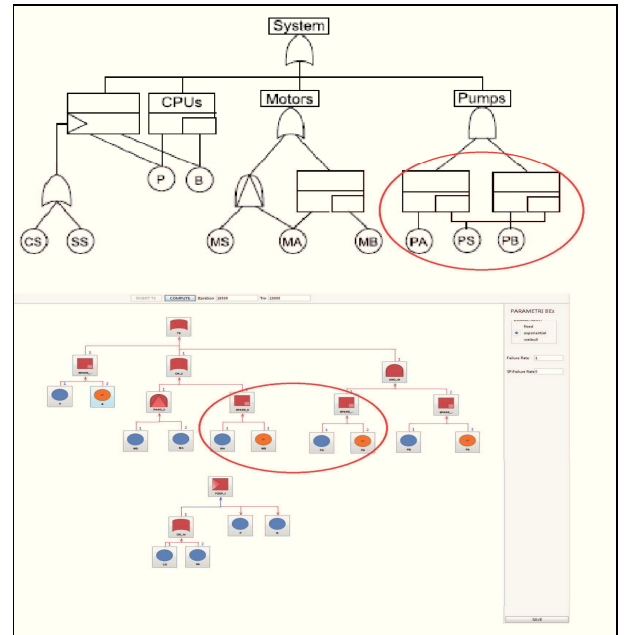


Fig. 12: DFT of the CAS [20] developed with jDFTDes

Table 3: failure rates of the BE of the CAS in [h⁻¹]

Component	CS	SS	P	B	MA	MB	MS	PA	PB	PS
Rate	0.2	0.2	0.5	0.5	1	1	0.01	1	1	1

VII. FINAL RESULTS AND CONCLUSION

Table 5 shows the comparison among the results for the proposed cases of study, for each tools used. They definitely prove that:

1) simulation is the most valuable approach for those complex dependent models that analytical tools are not able to solve and

2) simulation can also be used to compare and confirm the evaluation provided by the analytical approaches.

The binomial of MatCarloRE engine and jDFTDes GUI suits perfectly these two requirements; in fact, on one hand the risk assessment evaluations of the MatCarloRE library are satisfactory both in terms of accuracy and time of computation, warranting the resolution of complex models; on the other hand, the intuitiveness of the graphic user interface implemented with the jDFTDes favours the modeling design of large DFT, encouraging the use of the tool.

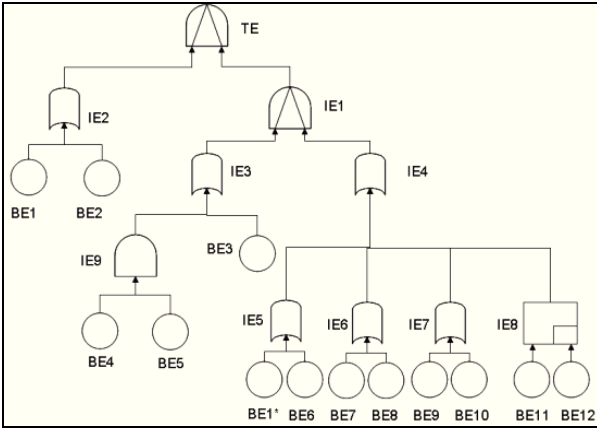


Fig. 13: DFT of the SAP [5]

As far as concerns the comparison of performance between the simulating tools DFTSim and MatCarloRE, we have to rely on the assumptions made in the previous section. Under those conditions we can assess that, on equal terms of machine configuration, DFTSim seems to offer greater performance than MatCarloRE, as proven by the Execution Time (Table 5). This happens in particular for DFT models in which more than one SPARE gates share a spare component (see example of Fig. 11 and Fig. 12). In our opinion, the reason is that the stream of instructions used to code the algorithms of the SPARE gates logic is more efficient in the DFTSim; this offers new clues for the improvement of MatCarloRE. In future research, we aim to be able to test both the tools with the same machine configuration in order to refine these evaluations.

At the state of the art, a point in favour of the MatCarloRE library is the integration with the Matlab® framework that allow a simple implementation of the parallel paradigma. In Table 6, the results of the distributed computing are shown:

when four processors share in parallel the computation effort, the performance of MatCarloRE are greater than the DFTSim.

Table 4: failure rates of the BE of the SAP

Component	Value	Type
BE1	1x10 ⁻³	Fixed
BE2	9.1x10 ⁻⁴ [h ⁻¹]	Exponential
BE3	1x10 ⁻³	Fixed
BE4	1.71x10 ⁻⁴ [h ⁻¹]	Exponential
BE5	7.51x10 ⁻⁴ [h ⁻¹]	Exponential
BE6	9.11x10 ⁻⁴ [h ⁻¹]	Exponential
BE7	4.51x10 ⁻³ [h ⁻¹]	Exponential
BE8	8.61x10 ⁻⁴ [h ⁻¹]	Exponential
BE9	4.51x10 ⁻⁴ [h ⁻¹]	Exponential
BE10	7.91x10 ⁻³ [h ⁻¹]	Exponential
BE11	1.51x10 ⁻⁴ [h ⁻¹]	Exponential
BE12	9.51x10 ⁻⁴ [h ⁻¹]	Exponential

Another considerable plus value of working inside the Matlab® environment is the plenty of mathematical instruments offered which can be used for further evaluations, like importance measures, sensitivity analysis and optimization.

In future work we aim to improve the library in order to:

- 1) remove the hypothesis of non reparable components, switching in the domain of the availability,
- 2) implement a set of instruments for the evaluation of the maintenance strategies,
- 3) deal with components that can be described by multiple states (overcoming the constraint of the working or failed state) and
- 4) adapt the parallel source code of MatCarloRE for the cloud computing (the Sicilia Grid infrastructure of Cometa).

Table 5: comparisons among the reliability tools

Case study	Tool	Iterations	Unreliability	Execution Time (sec)
CPS (T _m =1h)	Relex®	-	X	-
	Galileo®	-	0.00135	380
	DFTSIM ¹	10 ⁵	0.00142	40
	MatCarloRE	10 ⁵	0.00140	17 (2 Cpu)
CAS (T _m =1h)	Relex®	-	X	-
	Galileo®	-	0.65790	1
	DFTSIM ¹	10 ⁵	0.65651	43
	MatCarloRE	10 ⁵	0.65770	64 (2 Cpu)
MDCS (T _m =1h)	Relex®	-	X	-
	Galileo®	-	0.06664	1
	DFTSIM ¹	10 ⁵	0.06737	39
	MatCarloRE	10 ⁵	0.06680	52 (2 Cpu)
SAP (T _m =8760h)	Relex®	-	X	-
	Galileo®	-	X	-
	DFTSIM	Not tested	Not tested	Not tested
	MatCarloRE	10 ⁶	0.000185	278(2 Cpu)

¹ Results taken from the paper [20] performed with a different system configuration

Table 6: distributed computing results of MatCarloRE

Case study	N.of CPU	Unreliability	Execution Time (sec)
CPS	1	0.00130	26
	2	0.00140	17
	4	0.00140	9.8
CAS	1	0.65570	87
	2	0.65770	64
	4	0.65960	39
MDCS	1	0.06480	77
	2	0.06680	52
	4	0.06591	32
SAP	1	0.000192	546
	2	0.000185	278
	4	0.000197	162

For any request about the use of the MatCarloRE and JDFTDs, you can email the scientific coordinator, Prof. Lucio Compagno, at his email address: lco@diim.unict.it.

REFERENCES

- [1] M. Cepen, B. A Mavko, "A dynamic fault tree", *Reliability Engineering and System Safety*, 75: 83-91, 2002.
- [2] S. Distefano, A. Puliafito, "Dynamic reliability block diagrams vs dynamic fault trees" *In Proceedings Annual Reliability and Maintainability Symposium*, RAMS '07: 71-76, 2007.
- [3] M. Bouissou, 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*, 11, 149-163, 11 2003.
- [4] R. Gulati, J. B. Dugan, "A modular approach for analyzing static and dynamic fault trees" *in Proceedings Annual Reliability and Maintainability Symposium*: 57-63, 1997.
- [5] F. Chiacchio et al., "Dynamic fault tree resolution: a conscious trade-off between analytical and simulative approaches" *Reliability Engineering and System Safety*, 2011, DOI 10.1016/j.res.2011.06.014.
- [6] E. Zio, "Biasing the transportation probabilities in direct Monte Carlo" *Reliability Engineering & System Safety*, 47, 59-63, 1995.
- [7] W.E. Vesely, "Fault Tree Handbook" *NUREG-0492*, 1981.
- [8] G. Merle et al., "Dynamic Fault Tree Analysis based on the Structure Function" *in Proceedings Annual Reliability and Maintainability Symposium*: 1-6, 2011.
- [9] H. Boudali, P. Crouzen, M. Stoelinga, "A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains" *K.S. Namjoshi et al. (Eds.): ATVA 2007, LNCS 4762*, pp. 441-456, 2007.
- [10] J.B. Dugan, J.S. Bavuso, M.A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems" *IEEE Transactions on Reliability*, 41 (3):363-377, 1992
- [11] H. Boudali, P. Crouzen, M. Stoelinga, "Dynamic fault tree analysis using input/output interactive Markov chains" *Proceedings 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks DSN '07*: 708-717.
- [12] A. Bobbio, L. Portinale, M. Minichino, Ciancamerla, E., "Improving the analysis of dependable systems by mapping fault trees into Bayesian networks" *Reliability Engineering and System Safety*, 71, 249-260, 2001.
- [13] A. Anand, A. K. Somani, "Hierarchical analysis of fault trees with dependencies, using decomposition" *Proceedings Annual on Reliability and Maintainability Symposium*, 69-75, 1998.
- [14] E. Windebank, "A Monte Carlo Simulation Method Versus a General Analytical Method for Determining Reliability Measures of Repairable Systems" *Reliability Engineering*, 5, 73-81, 1983.
- [15] K. Durga Rao et al., "Dynamic fault tree analysis using Monte Carlo simulation in probabilistic safety assessment" *Reliability Engineering and System Safety*, 94, 872-883, 2009.
- [16] K.J. Sullivan, J.B Dugan, D. Coppit, "The Galileo fault tree analysis tool" *in Proc. Digest of Papers Fault-Tolerant Computing Twenty-Ninth Annual International Symposium*: 232-235, 1999.
- [17] S. Amari, G. Dill, E. Howald, "A new approach to solve Dynamic Fault Trees" *in Proceedings Annual Reliability and Maintainability Symposium*, 374-379, 2003.
- [18] E. E. Lewis, F. Bohm, "Monte Carlo Simulation of Markov Unreliability Models" *Nuclear Engineering and Design*, 77, 49-62, 1984.
- [19] A. Dubi, "Monte Carlo Calculations for Nuclear Reactors" *CRC Handbook of Nuclear Reactors Calculations, Vol. II*, CRC PRESS (1986)
- [20] H. Boudali, A.P. Nijmeijer, M.I.A. Stoelinga, "DFTSim: A Simulation Tool for Extended Dynamic Fault Trees" *in Proceedings of SpringSim*, 2009, article n.31, 1-8.
- [21] <http://www.mathworks.com/help/toolbox/distcomp/creatematlabpooljob.html>

MATCARLOAV, AN EXTENSIBLE MATLAB® LIBRARY FOR THE SIMULATIVE EVALUATION OF DYNAMIC FAULT TREES

Abstract

In recent years, a new generation of modeling tools for the risk assessment have been developed. The concept of “dynamic” was exported also in the field of reliability. At first, the state-space technique of modeling were successfully used to implement dynamic dependencies in a fault schema, thus overcoming the limits of the traditional combinatorial techniques. Afterwards, more descriptive techniques (like DFT, DRBD, BDMP, etc.) have been proposed in order to enrich the intuitiveness of combinatorial methods with the capability to model dynamic dependencies. But, despite the promises of researchers and the efforts of end-users, the dynamic paradox raised: risk assessment procedures were not as straight as earlier and, what is worse, it was difficult to understand the effects of such dynamism. In this paper, we focus on the DFT technique and present a tool for the reliability and availability computation of a quite generic class of system. Starting from the state of the art, a set of standardized rules that clarify the real behaviours of the dynamic gates – in particular for what concerns DFT with repairable components – is drawn. Afterwards, a comparisons with earlier works (commercial and non commercial applications) will prove the advantages of this novel simulative approach, for which a Matlab® library is under development. The aim is to provide a basic library for the resolution of extended DFT. The tool may result of great interest because it is written with Matlab® code, hence is open-source and can be also extended. It has been tested with many literature cases of study such that results encourage other developments.