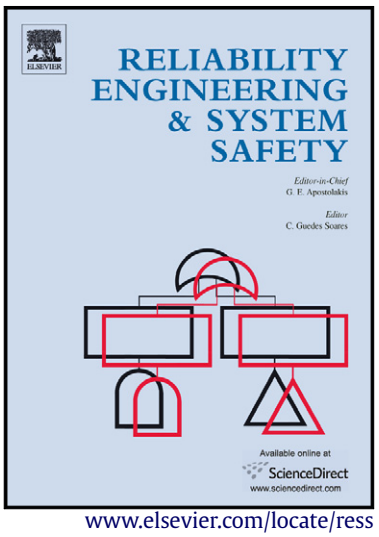


www.elsevier.com/locate/ress

Conception of Repairable Dynamic Fault Trees and Resolution by the use of RAATSS, a Matlab® Toolbox based on the ATS formalism

G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, N. Trapani

PII: S0951-8320(13)00259-7
DOI: http://dx.doi.org/10.1016/j.ress.2013.09.002
Reference: RESS4938

To appear in: *Reliability Engineering and System Safety*

Received date: 24 November 2012
Revised date: 24 August 2013
Accepted date: 5 September 2013

Cite this article as: G. Manno, F. Chiacchio, L. Compagno, D. D'Urso, N. Trapani, Conception of Repairable Dynamic Fault Trees and Resolution by the use of RAATSS, a Matlab® Toolbox based on the ATS formalism, *Reliability Engineering and System Safety,* http://dx.doi.org/10.1016/j.ress.2013.09.002

Conception of Repairable Dynamic Fault Trees and Resolution by the use of RAATSS, a Matlab® Toolbox based on the ATS formalism

G. Manno

Det Norske Veritas, Research & Innovation, Høvik, Norway

F. Chiacchio

University of Catania, Department of Mathematics and Informatics, Catania, Italy

L. Compagno, D. D'Urso & N. Trapani

University of Catania, Department of Industrial & Mechanical Engineering, Catania, Italy

ABSTRACT: Dynamic Fault Tree (DFT) is a well-known stochastic technique for conducting reliability studies of complex systems. At the state of the art, existing tools (both academic and commercial) do not fully support DFT with repairable components and repeated events, lowering the penetration of this powerful technique in real industrial applications (e.g., industrial processes and plants, computer, electronic and network applications). One of the main reasons limiting the attractiveness of DFT is that, originally, DFTs were conceived without repairable components; only recently few related works have started to deal with a formal semantic, which would avoid undefined behavior and misinterpretation of DFT. Other researches have tackled the problem by introducing extensions of the original Fault Trees (FTs) technique like Boolean Driven Markov Processes (BDMPs) and Generalized Fault Trees (GFTs). However, despite they consider repairable systems and repeated events, we have found that the introduction of a different formalism with more complex features has again limited the penetration of these powerful methods in real applications. The target of this work is the original DFT technique. Starting from the state of the art, a set of standardized rules that frame the behaviors of dynamic gates are designed and a well-defined semantic for repairable-DFT is drawn through the application of a novel formalism, the Adaptive Transitions System (ATS). The proposed theoretical framework is afterward used to code a software tool, RAATSS, for the resolution of extended, repairable-DFT. Moreover, this work introduces some novel concepts regarding the modeling of a system by a DFT and provides a basic hint of the ATS capabilities to describe interdependencies in complex system.

1 INTRODUCTION

Although risk assessment evaluation of complex dependable systems can be performed through the use of dynamic stochastic modeling, in the real industrial world the well-known combinatorial techniques, such as Reliability Block Diagram (RBD) and Static Fault Tree (SFT), are still the most widely used [1, 2]. The main reasons that have limited the adoption of dynamic stochastic modeling in industrial applications can be related to the following issues: (*i*) enterprises do not bother to update the risk assessment of existing processes (or plants) evaluated with the static techniques; (*ii*) static techniques offer exact and simple solution algorithms as they do not consider time and cross dependencies among the parts of a system (e.g., spares replacement, load sharing, chain of events); and (*iii*) dynamic modeling is still too enigmatic and there is not a straight solution algorithm to solve any type of complex model [3].

The first issue is reasonable because updating a risk model is not the main activity of an enterprise and it costs time and money. Moreover, the enhancement of existing static models with dynamic features can result tedious mainly because commercial (and academic) solutions are not well developed or easy to use. In fact, at the state of the art, all the available solutions present several limits [3]. Continuous time Markov chains (CTMCs) are the progenitor of stochastic dynamic modeling [4]; they are very flexible and can model several types of systemic dependencies. However, although they obey to a rigorous mathematical foundation, their use is limited to applications that can only be described by exponential distributions. Moreover, CTMCs have at least two other important issues: (*i*) large models can easily turn into the state space explosion, which makes the analytical resolution unfeasible; (*ii*) the system oriented representation, i.e., components, mechanisms and their interaction, is lost due the flat representation of the state space.

Regardless of the mentioned merits and limits, CTMCs have been the launching pad for other formalisms, like Dynamic Fault Trees (DFTs), Dynamic Reliability Block Diagrams (DRBD), Boolean Driven Markov Process (BDMP), Generalized Fault Tree (GFT) [5-9]. Introducing new formalisms, researchers have tried to overcome the limits of the static techniques by combining a symbolic high level representation based on modeling building blocks with the powerful flexibility of state space models.

Among these new formalisms, DFT covers an important role as it represents the evolution of Static Fault Trees (SFTs). In DFTs the use of dynamic gates makes it possible to consider time and cross dependencies by the aid of high level building blocks, which reduce the modeling effort typical of flat representations (like state-space techniques). However, despite the easiness of designing a DFT, its resolution could be very tedious [3]. As a matter of fact, researchers have proposed several solutions that handle only reliability evaluations, excluding the possibility of restoration [5]. In the authors' opinion, this is another major reason that has further confined the diffusion of DFT. In fact, the DFT semantics was conceived taking into account only non-repairable systems [10]; thus, their use and diffusion within the real industrial world has been limited only to reliability studies.

The aim of this work is to address and overcome the major issues for the introduction of repairable-DFT. Starting from the state of the art [3, 5, 7-29], a set of standardized rules that frame the behavior of dynamic gates are designed and a well-defined semantics for Repairable-DFT (RDFT) is drawn through the application of a novel formalism, the Adaptive Transitions System (ATS), able to implement a powerful and flexible discrete event simulation engine [13]. The proposed theoretical framework has been used to code a software tool, the 'Reliability and Availability ATS Simulator' (RAATS), for the resolution of extended-RDFTs [20]. The main innovative features introduced in RAATSS are the following:

- The computation of the availability and reliability of semi and fully repairable systems [14], i.e., reliability with repairable components, through the introduction of failure gates;
- The adoption of well-defined rules for repairable components.

The paper is organized as follows: in Section 2 a short overview of RDFT is presented, highlighting the limits of the state of the art and presenting the enhancements brought with the introduction of repairable components. This represents an important part of the work as it offers the foundation for the comprehension of the fault logics arising when using repairable components and how to avoid unexpected behaviors. In Section 3, the ATS paradigm is shortly introduced. In Section 4 the RAATSS tool is presented, showing how it is possible to describe and convert a RDFT into an ATS model. Finally, in Section 5 conclusions and future works are drawn. The appendix section gives more insights on the mathematical foundation of ATS.

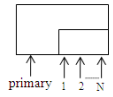
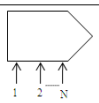
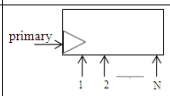
Name	Graphical Representation	Description (N input)
SPARE		It triggers only after the primary if all the N spares occur. Spares can be shared with other spare gate.
PAND		It behaves like an AND gate but it triggers only if the input events occur in the order from the left to the right.
SEQ		It forces 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.

Figure 1. Most frequently used Dynamic gates.

2 REPAIRABLE DYNAMIC FAUL TREES

DFT analysis has been one of the most promising RAMS techniques for the reliability evaluation of complex dependable systems. The interest around this technique is due to special features introduced by dynamic gates, able to model time evolution and cross dependencies (Figure 1).

Like in the original Fault Tree technique, also denoted as static-FT (SFT), the construction of a DFT is a TOP-DOWN exercise where a Top Event (TE) is related to its basic causes (i.e., Basic Events, BEs) by logical gates. BEs are the leaves of the fault tree and represent the elementary events, generally linked to the failure of components, not requiring further investigations. BEs can also be repeated (i.e., they appear two or more times in the model of the fault schema as inputs of two or more different gates). However, their entity within the real system is unique. For instance a single component as a battery can be a power supply for several units of the same system; therefore, its failure will affect all these units that are generally modeled in different subsystems (i.e., the sets of elements below a gate).

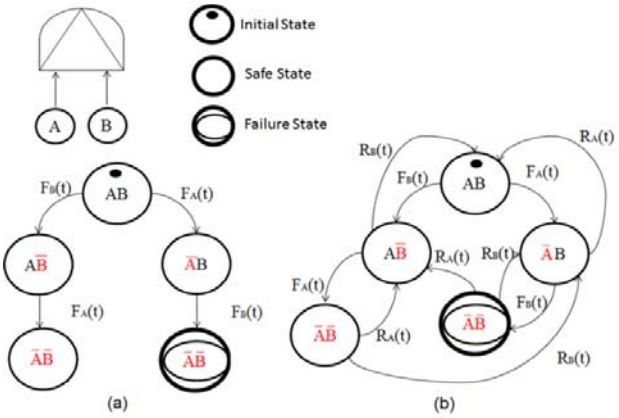
The resolution of a DFT is much different than the one of SFT because temporal and cross dependencies cannot be modeled and solved through Boolean algebra. At the state of the art several techniques have been proposed to solve DFTs, but none of these is able to provide a generalized process of resolution for all the classes of DFT models, which contain repeated events, dynamic gates at the top of the fault tree, repeated events and basic events whose time to occur is non-exponentially distributed [3]. Techniques for solving a DFT are generally based on mapping the DFT into a different model which can be solved analytically or via simulation. Among them it is possible to distinguish state-space models (e.g., Markov chains, Generalized Semi-Markov Processes, Bayesian network, etc.) or other graphical models using a high level language of description like, e.g., Stochastic Petri Nets (SPNs) and BDMP[42, 43].

State space models can potentially describe unlimited type of scenarios, as the representation of the process evolution is described through the use of states and connections whose meaning is decided by the modeler. Drawbacks of state-space models are:

- They are difficult to design, the more the number of states, the higher the probability to make mistakes or to forget some logic status of the process;
- They are hard to read and maintain because of the flat description of the process (i.e., components, mechanisms, their interactions and status are coded inside the states of the model).

Research efforts have been looking for automated tools [11,12,17,20,23,25-29] able to convert DFT into state-space models and able to apply techniques to improve computational performance and avoid the state-space explosion, like Stochastic Process Algebra, lumping or aggregation [30-33] or using local explorations of the state graph via sequences [34-36]. In [16, 25] DFT are solved via the conversion into a Bayesian Network (BN). This method mitigates the state space explosion but, as direct acyclic graph, BN cannot model cyclic dependencies; hence restoration is not allowed. Other approaches have used SPN as the target formalism for solving DFTs [18]. The main advantage of these techniques over the state space models is the use of a high-level description language and the possibility to solve the model via simulation. This feature is very important because a simulative solution is always feasible. A main drawback of this approach is that the resulting SPN model are cumbersome in the sense that the entire converted DFT have to be implemented in one atomic SPN model to be solved; limiting the readability and the maintainability of the model and the possibility of making changes to the model, at the SPN level, where a separation of concerns approach is usually more suitable for modeling complex relations.

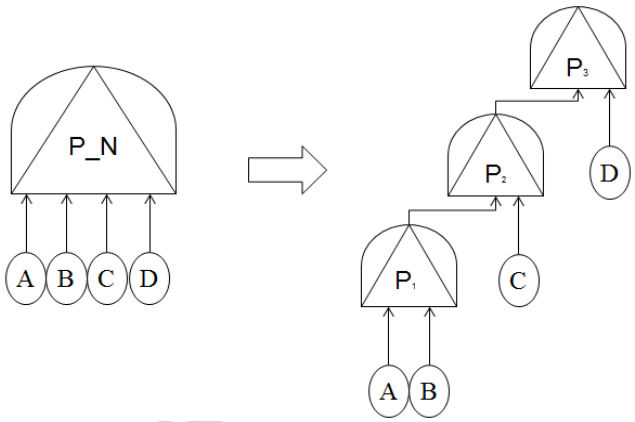
The discussion of other methods which do not use DFT as the main modeling formalism is out of the scope of this paper. However, the reader should be aware of other alternatives like Dynamic Reliability Block Diagrams and BDMP. They are very powerful formalisms. However, in the authors' opinion there are several limitations that confine their use in real applications. For instance, in BDMP only one type of cross dependency among components is allowed, by the addition of a trigger event. Unfortunately, this feature is too limitative to approach the modeling of complex DFT as, for instance, it cannot manage systems with two or more



spare gates which share several spare components.

Finally, conversely from SFTs, in the original conception of DFTs [5] repairable components were not included: this hypothesis has, so far, limited the use of this technique in real industrial applications. The conception of RDFT has to be framed within this context, dictated by the wish of the reliability modelers to exploit a user-friendly graphical language able to describe complex dependencies, while keeping the readability of the fault schema. The following sub-sections introduce the semantics of RDFT comparing it with earlier works [5,8,9,20, 21,23].

2.1 The Priority-AND Gate



The failure of a Priority-AND (PAND) gate occurs only if its inputs fail in the order from left to right in the gate graphical notation. As observed in [8], if repairable components are inputs of a PAND, a component can repeatedly fail and be repaired such that the failure order could also be respected, causing the gate to trigger. In [8], it was assumed that only the first failure of each component is the one affecting the final outcome of the gate; but, as explained later on, it is important to consider always the last failure of each component (unless restorations are not meaningful). In fact, following the approach of [8], the outcome of the gate would be known and freeze immediately after a sequence of "first failure" not verifying the "working order" of the gate occurs (see Semantics of [8]). This logic is not worth if repairable components are used because in this way restorations would become meaningless. Conversely, in this work, the authors adopt a different logic considering the last failure occurred in the sequence of failure order. The state space representation of a *2*-input PAND in Figure 2 can help understanding the difference of behavior between a PAND without and with repairable components and explain the logic above adopted.

As clarified in Figure 2.b, it can happen that component B can fail first than its preceding A; if this happens, a possible sequence could be that B can be repaired; in this condition it means that A is broken while B is not: this verify the failure order that drives the PAND gate to trigger. If, as stated in [8], only the first failure determines the outcome of the gate it means that possible restorations would not play any role for the PAND gate and the outcome of the gate is already known as soon component B fails.

Further considerations are needed as for the restoration of the gate (i.e., after the PAND has triggered). In [9] several options are listed; among the solutions suggested, in this paper it has been assumed that the PAND behaves like an AND gate, hence it steps down if one of its input is restored.

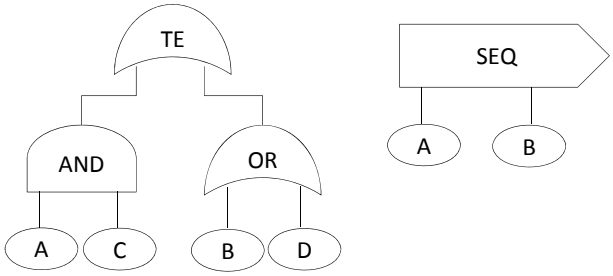
The above defined behavior for a 2-input PAND Gate can be generalized for a *n*-input PAND Gate, with $n > 2$. In fact, as shown also in [9], a PAND with *n* inputs can be implemented with a cascade of $(n - 1)$ 2-inputs PAND gates, see Figure 3.

Figure 2. State space representation of a 2-input PAND with (a) non and (b) repairable components. F(t) represents the cumulative distribution function of failure while R(t) the restoration one.

Figure 3. Equivalent 2-inputs representation of a 4-inputs PAND.

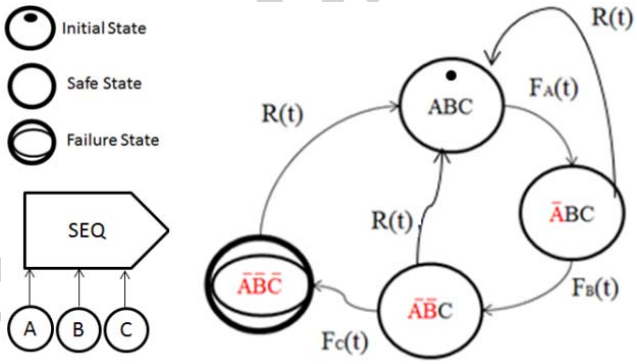
2.2 The Sequence Enforcing Gate

The Sequence Enforcing (SEQ) gate forces its input events to occur in a fixed order: from left to right in the gate graphical notation. The SEQ gate is often associated with repeated events. In this case, the inputs of a SEQ can be more appropriately referred



as "dependent events". The use of the SEQ gate without repairable components was discussed in [22] where it was noticed that its behavior can be expressed in terms of cold SPARE gate.

The influence of a SEQ gate on its dependent events has been described in earlier works [8, 9] and it is straightforward if its inputs are non-repairable [9]. Figure 4 shows a hypothetical simple fault tree for the purpose of clarifying the behavior of a SEQ Gate: in this example, the event B under the OR gate cannot fail until event A, under the AND gate, has failed. This is due to the ordering of the dependent events under the SEQ Gate.



What happens when components can be repaired? According to the original meaning of the SEQ Gate, the gate rules only the order of failure of components; therefore their restorations must be independent. If component A of Figure 4 fails and gets repaired, what should be the behavior of B? Does this sequence of events allow component B to fail? In [8] the approach used allows it, because it is claimed that once A has failed B can fail, no matters if A has been restored before the failure of B. In other words, the failure order imposed by the SEQ Gate is valid only the first time. In this paper, the logic used is more rigid, therefore component B will be allowed to fail only if A is in its failed state. More specifically, in this work inputs of a SEQ are considered as being the same component in different degradation states; therefore, a repair will influence the whole SEQ system, restoring the system as good as new. In the authors' opinion the difference between the above described behaviors should be translated into two different kinds of SEQ gates, which could be named as SEQ-DEG, standing for degradable, and SEQ-DEP, standing for dependent. As said, in this work only the SEQ-DEG will be described. A very good reference to the second type of SEQ behavior can be found in [11].

Further considerations are needed for the output of the SEQ Gate. In general, the output can be of two types: the first one is an intermediate event that feeds another gate of the fault tree, while the second one is a dummy [11]. In the authors' opinion, the use of one or the other output can be well ruled and depends on the practical application. In fact, when the SEQ gate is exclusively used to impose the sequence of failure of its dependent components (i.e. the repeated events inputs of other gates), it is more likely to use the dummy output because in this case the SEQ gate does not represent a real subsystems whose output corresponds with an intermediate event. In the other cases, the SEQ gate can be used as a real subsystem and its output can correspond with the intermediate event modeling its fault. Among these applications, the SEQ gate can be used to model degradable systems [23]. In this case, the cold SPARE gate representation cannot describe exhaustively the logic of the degradable system because, as shown in Figure 5, restoration can be intended as a global restoration, bringing back the overall subsystem and its component as new. This type of choice is an alternative choice to the previous one and it has been made by the authors in order to cope with the idea of degradable system, guaranteeing both the failure logic imposed by the SEQ Gate and a consistent behavior as for the restoration of the system.

Figure 4. Simple example containing a 2-input SEQ.

Figure 5. A 3-inputs SEQ gate modeling degradable system: restoration is allowed from any state bringing back the system as new.

2.3 The Functional Dependency Gate

The Functional Dependency (FDEP) gate forces an instantaneous unavailability of its secondary inputs (known as "dependent inputs") when the primary input (known as the "trigger") occurs. In this paper, like in [24], the trigger can be modeled also with more complex structures and not just with a basic event.

The output of this gate is a dummy; therefore the FDEP gate is not input of any other gate. This gate can be used to model a functional dependency existing among dependent components and the trigger. Remarkable literature cases can be found in [5] where a structure of parallel processors interact with a telecommunication network or in the Cardiac Assist System [10], modeling the behavior of a set of spare components that can be activated only if the switch is not broken.

The effect of a functional dependency induced by a *n*-inputs FDEP gate (with *n* - 1 secondary components) can be obtained with an equivalent representation which makes use of a set of (*n* - 1) equivalent OR gates [19, 24]. In particular, all the dependent BEs of the DFT which appear as the *i*-th secondary input of the FDEP gate can be replaced with a 2-inputs OR gate, where the first input is the trigger of the FDEP gate and the second one is the *i*-th BE input of the FDEP. This description helps understanding the behavior of a FDEP in case of repairable components. Basically, the logic used is to consider the trigger as a switch input of the equivalent OR gate. Therefore, if the trigger is failed the OR gate exhibits a failure, but the dependent components can autonomously fail and get restored.

The ATS model of the FDEP Gate is introduced in the next section; it is important to highlight that the ATS representation will not make use of the equivalent OR gate representation above described, but the resulting logic will be equivalent.

2.4 The SPARE gate

According to [5], a SPARE gate can have only one primary input and an undefined number of spare components which could be shared with other SPARE gates. Components of SPARE gates have the following characteristics:

- They are ordered (from left to right in the graphical notation);
- They can be active or stand-by, where "active" means operating in a SPARE gate and "stand-by" means not operating in any SPARE gate;
- They can be available for replacement (i.e., ready to start working to substitute a failed component) if it is in the working state and it is not active; otherwise it is said unavailable.

If all the inputs of a SPARE gate are unavailable, the gate triggers. In case of repairable components the following logics are used:

- If a preceding component (following the graphical ordering) of an active component becomes available, the former is set to "active" while the latter is set to "stand-by" (*priority rule*);
- The time to failure of a component is subjected only to changes between stand-by and active conditions, i.e., no re-sampling of the time to failure when switching between different active conditions (i.e., for spare components shared by more gates);
- If an available spare component is requested simultaneously by two or more SPARE gates an opportune activation policy must be chosen, e.g., first-in-first-out (FIFO), non-deterministic, specific priority policies, etc. According to this current framework, a non-deterministic policy of replacement is used, i.e., the SPARE gate is chosen randomly.

2.5 Failure Gates

Failure gates are not a new type of gate but a property that can characterize all the existing gates. In fact, either a static or a dynamic gate is said a failure gate if it cannot be restored after it triggers the first time. This type of behavior can be used to solve the problem of the first occurrence (i.e., the reliability of a system with repairable components) just by modeling the top event with a failure gate, thus overcoming the traditional solution that used to resort to the equivalent CTMC model.

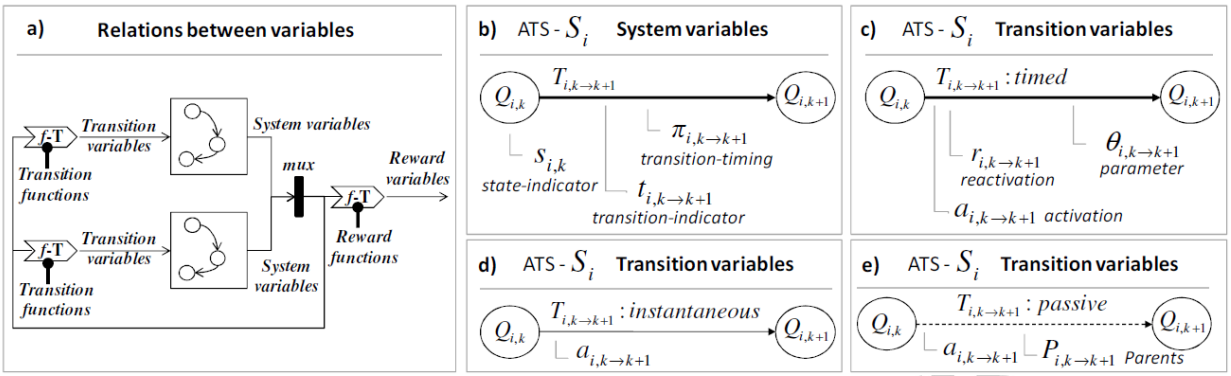


Figure 6. a) relation between variables; b) system variables; c,d,e) model of transitions.

Furthermore, ATS allows to extend the concept of failure gate also for all the other gates of the fault tree (the intermediate events), making it possible to consider hybrid systems, made up of non-repairable and repairable logics. Two types of failure gates have been introduced:

- Type I, the basic events cannot be restored even though they are repeated events for other gates;
- Type II, unlike the previous case, the restoration of the basic event is not affected by the status of the failure gate.

Failure gate of type I are used to create a strong dependence between an intermediate event and its input components, in order to model situations in which the failure of the gate turns in an irreversible failure of the components, though it feeds other intermediate events. Instead, Failure gate of type II can be used to decouple the status of the failure gate from its inputs: in this way it is possible to model intermediate events that cannot be restored, but still allowing the input components to undergo repairs.

3 THE ATS FORMALISM

Adaptive Transition System (ATS) [13, 20] is a modeling formalism for parallel interdependent systems. In ATS a system is represented by a set of interactive transition systems that are said adaptive since the communication mechanism is based on the adaption, i.e., the change of the internal parameters of each ATS with respect to the evolution of the others. Some comparison between ATS and well known modeling formalisms like Performance Evaluation Process Algebra (PEPA), Interactive Markov Chains (IMC) [37], Stochastic Activity Networks (SAN) [38,39], Probabilistic Symbolic Model Checker (PRISM) [40], etc. [30, 37-39] can be found in [10].

ATS are made up of states and transitions. In ATS, central, is the model of transitions (see Figure 6.b-e). It is possible to distinguish among timed, instantaneous and passive transitions. Different classes of transitions are characterized by different attributes which can be expressed as a function of opportune variables, namely system variables, which define the state of the system. Thus, expressing the attributes of transitions as a function of the system variables, different transition systems can adapt to each other, i.e., state changes affect the behavior of different models.

Attributes of a timed transition are:

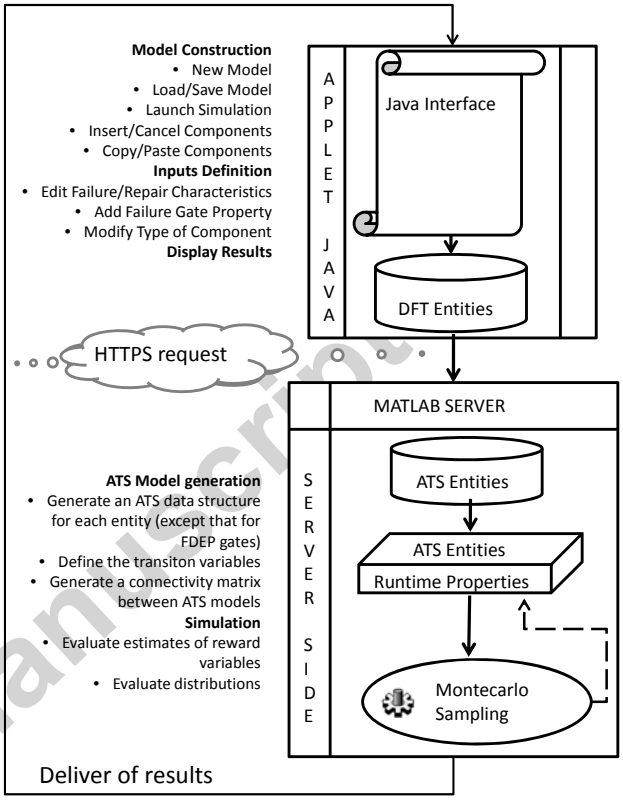
- Type, it is the mathematical relationship that defines the time to complete of the transition (e.g., exponential, Weibull, fixed, etc.);

Activation, $\boldsymbol{a} \in \{0,1\}$, it is a necessary condition that determines if a transition may complete;

Reactivation, $\boldsymbol{r} \in \{0,1\}$, it is the condition which defines whether the transition must be "re-started" (i.e., sampled again in simulation terminology);

Parameter, $\boldsymbol{\theta} \in \boldsymbol{R}$, it is a real variable(s) that specifies the parameter(s) of the supported distribution function defined by the type of the transition.

Differently from timed transitions an instantaneous transition completes as soon as enabled. The only attribute of an instantaneous transition is the activation of the transition.



For a passive transition the activation attribute is defined, too. However, differently from instantaneous transitions, passive transitions are not allowed to complete alone but *simultaneously* when one parent transition completes.

System variables allow reorganizing the information about the system evolution by taking into account: (*i*) the state of each transition system, $\boldsymbol{s} \in \{0,1\}$ (indicator function of the state; defined for each state); (*ii*) the last occurred event across the overall model, $\boldsymbol{t} \in \{0,1\}$, i.e., the last completed transition; and (*iii*) the time at which every event has occurred in the model, $\boldsymbol{\pi} \in \boldsymbol{R}$, i.e., the most recent time of completion of transitions.

An ATS is said to depend on another ATS, if the transition attributes of the former are expressed as a function of the system variables of the latter. Finally, system variables are used as inputs of reward functions, a set of functions used to retrieve measures of interest about the system (Figure 6.a) [39].

The execution logic of ATS is similar to the one defined for Generalized Semi Markov Processes (GSMP) [41]. A vector of the time to complete transition is kept for each of them. The minimum among these times determines the transition that will complete. The process of completion of a transition includes the update of the system variables and subsequently of the reward and transition variables, i.e., the values of the attributes of transitions. At this point, changes in the enabling or in the reactivation conditions will determine a new value of the time to complete of the transition. In practice, a simulation batch of ATS can be described by the following steps:

- (*i*) initialization of the state variables; (*ii*) of the time to complete of transitions (set by default to + ∞); and (*iii*) of the progressive simulation time (set by default to 0);

- then, until the ending simulation time (e.g., the mission time of the system) is reached, the following updating steps are performed in succession and cyclically: (*i*) update of reward variables; (*ii*) update of transition variables; (*iii*) selection of the next transition; and (*iv*) update of state variables.

Several attributes of ATS have been borrowed from the SAN and PEPA formalisms. In particular attributes of transitions such as the activation and the reactivation are common in SAN, while the concept of passive transitions have been borrowed from PEPA. Moreover, ATS offers a more general state representation, as three kinds of variables are used (i.e., state-indicator, transition indicator and transition timing). The enhanced state representation allows a more concise model design. For instance, the reactivation of a transition can be related to the occurrence of a specific event. An advantage that in SAN can be obtained by programming the SAN model with a high level programming language like C++ or, in case a graphical representation is needed, by making the model richer of support places included in the model only for the sake of obtaining the wanted behavior. Finally, the inclusion of passive transitions allows controlling unexpected behaviors given by the non-determinism induced by concurrently enabled instantaneous transitions.

See Appendix A for a more detailed description of ATS.

Figure 7. Architecture of RAATSS in an online environment.

4 RAATSS: THE ATS TOOL FOR RDFT RESOLUTION

RAATSS is a tool developed under the Matlab® framework. It embeds: (*i*) a graphical interface for the construction of a RDFT; (*ii*) an automatic conversion engine able to convert the RDFT into a set of ATS models; and (*iii*) a discrete event simulator for the solution of the ATS models. The graphic interface unit (GUI) has been coded in Java while the conversion engine and the discrete event simulator have been developed in Matlab® code. The main reason for the use of Matlab® is that an ATS model can be represented with a set of *n*-dimensional matrices; hence, Matlab® is a very suitable tool as it provides a wide number of high level functions for the creation and manipulation of mathematical structures (e.g., *n*-dimensional matrices, structures). Moreover, it offers a powerful environment for the programming of computer algorithms. Figure 6 shows the interaction between the GUI and the Matlab® Server. The figure shows how RAATSS would work through a web-service application: the construction of the RDFT is handled through a Java Applet that runs the GUI into the client of the user, while the computational effort of converting the RDFT into an ATS model and the simulation of the ATS are performed by a dedicated server.

Matlab® receives the information generated by the Java applet in the form of data structures regarding the elements and the parameters of the RDFT. This information is used by the RDFT-to-ATS converter to generate the data structures of the ATS models which are passed to the discrete event simulation engine that, in turn, returns the values of rate and impulse rewards [39] associated with each state and transition of the model. Figure 7 shows the architecture of RAATSS.

4.1 RDFT-to-ATS Conversion Rules

A set of graph conversion (e.g., [18]) rules are applied in order to map a RDFT into an ATS model. In particular RDFT objects are mapped into ATS objects. For this reason, at first, the *RDFT objects have to be classified* in order to define a suitable ATS model. During the conversion procedure, RAATSS assigns ATS models to the RDFT elements on the basis of the class they belong to.

Any new extension at the RDFT level (e.g., new type of gates) can be realized specifying the correct mapping in terms of ATS model. In the following sections the conventional rules to convert traditional RDFT into ATS are presented.

4.2 ATS Models of BEs

BEs are classified in four groups depending on their position in the tree (inputs of SEQ gates are treated separately in Section 4.6). In particular four classes of BEs are defined: C1, C2, C3 and C4.

Table 1.a illustrates how to assign the correct class to a generic BE: basically it depends whether or not a BE is a secondary input of a SPARE or/and of a FDEP gate (the element not in Table 1.a means that the BE is not a secondary input of any

of these gates). BEs belonging to the class C1 are independent from other BEs of the fault tree since they are not influenced from other elements of the RDFT. BEs belonging to the class C3 are influenced only by trigger elements (i.e., they are secondary inputs of FDEP gates), while BEs belonging to C2 are affected only by spare policies (i.e., they are secondary inputs of SPARE gates). Finally, C4 is the class of BEs subjected to the influence of trigger elements and to changes of the distribution of the time to failure (i.e., they are secondary inputs of FDEP gates and spare components of SPARES gates).

Three kinds of ATS are defined and their combinations are used to define the structure of the model for each class of BEs defined above (see Table 1.b):

- H-ATS (or healthy-ATS), it is a transition system made up of two states, representing the fact that a BE might be working or failed, connected by two timed transitions (i.e., failure and repair processes). It is used for BEs belonging to any class. A H-ATS is said H^d-ATS in case of further dependencies affecting the failure distribution of the BE.
- O-ATS (or operative-ATS), it is a transition system made up of two states, representing the fact that the BE might be stand-by or active. It is used for BEs belonging to the classes C3 and C4.
- F-ATS (or functional-ATS): it is a transition system made up of two states representing the fact that the BE might be on or off. It is used for BEs belonging to the classes C2 and C4.

4.3 ATS Model of AND, OR and PAND

For each static gate (e.g., AND, OR) and the dynamic PAND gate a transition system is generated, namely G-ATS (or gate-ATS), that is made up of only two states that represent the working and failed conditions of the gate. The conversion procedure simply codifies the logic expressed by these gates in the form of activation functions of the instantaneous transitions of

the model.

As an example, let us consider Figure 8.a where a 2-inputs PAND is shown. The resulting ATS model of the system consists of three ATS, one for each element of the tree. Bold arrows indicate that the G-ATS model is dependent on the two H-ATSs, i.e., the transition variables of the two instantaneous transitions of the G-ATS are expressed as a function of the system variables of the two H-ATSs.

In particular, Table 1.c says that only the activation predicates of these transitions are relevant attributes for this ATS. Table 1.c shows also that a G-ATS can depend on any other kinds of ATSs except O-ATSs. In fact, gates belonging to this group can take as inputs any type of RDFT object.

For the sake of clarity, a brief description of the way the activation predicate of the transition from the working to the failed state of the G-ATS is defined is given below. Let f_1 and f_2 denote the *state indicator variables* of the two states representing the failed condition of BE1 and BE2, respectively. Moreover, let τ_1 and τ_2 denote the *transition timing variables* of the two transitions representing the failure process of BE1 and BE2, respectively. Then, the activation predicate, a_{G1}, of the transition from the working to the failed condition of the PAND gate is:

$$a_{G1} = f_1 \cdot f_2 \cdot (\tau_1 \leq \tau_2). \qquad (1)$$

4.4 ATS Model of SPARE Gates

The ATS model of a SPARE gate with *N* inputs is made up of *N*+1 states and is fully connected. *N* of the *N*+1 states give information about the component that is currently active, while the remaining one represents the failed condition of the gate. This kind of ATS is called SPARE-ATS.

Figure 8.b shows the representation of a spare subsystem in ATS. The SPARE gates SG is represented by a SPARE-ATS with three states; BE1 (of class C1) is represented by a H-ATS; BE2 (of class C2) by an H^d-ATS and an O-ATS.

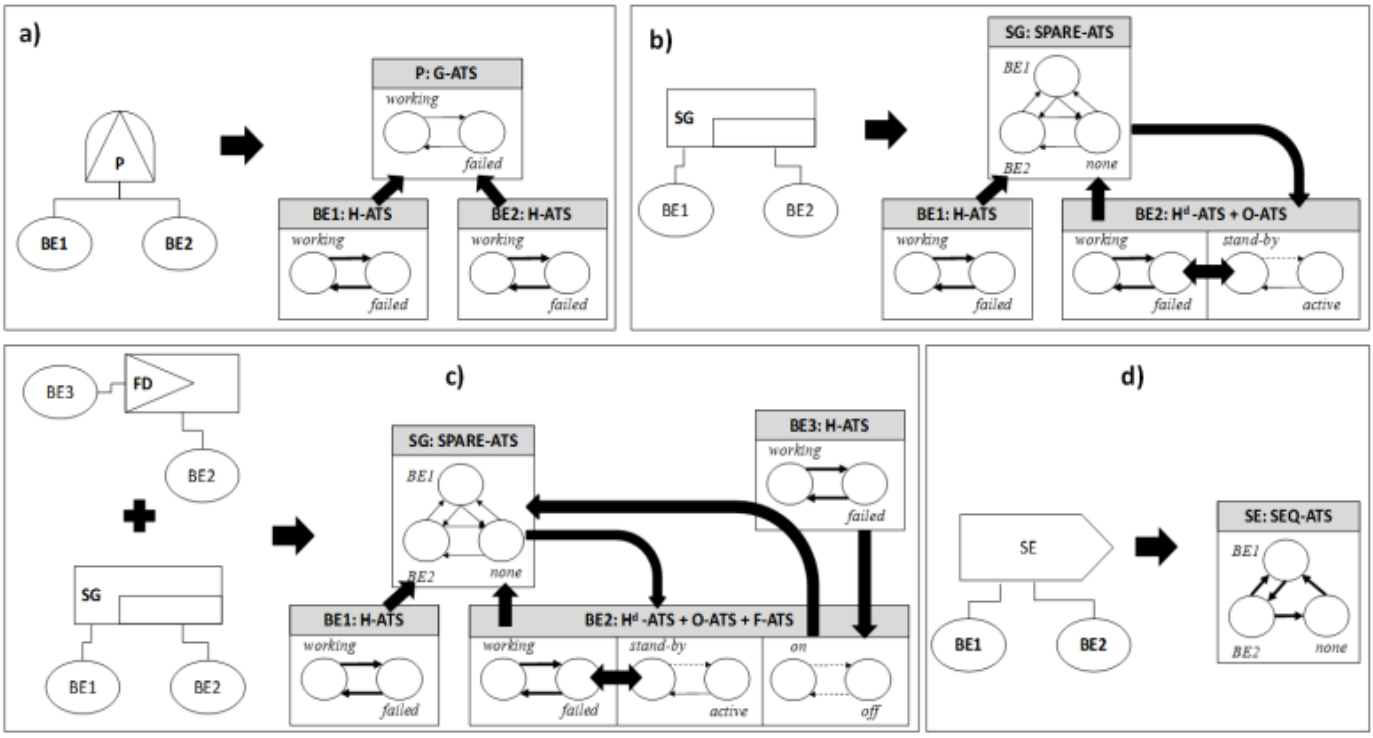


Figure 8. Example of RDFT-to-ATS conversion. a) PAND; b) SPARE; c) FDEP; d) SEQ

Here, the SPARE-ATS depends on the H-ATSs of the two BEs. A change of state, in one of the two H-ATSs may lead to a change of state in the SPARE-ATS. This, in turn, leads to a change of state in the O-ATS with consequent effects also on the H^d-ATS. Table 1.c shows that a SPARE-ATS can depend almost on any kind of ATS objects. This because SPARE gates can have as inputs also other kind of gates (as in extended DFT [17]). When modeling a SPARE-ATS it is necessary to define the activation variables of all the instantaneous transitions of the gate in such a way that the conditions defined in Section 2.2 about the replacement logic implemented by the gate will be respected.

4.5 ATS Model of FDEP Gates

FDEP gates are not converted into a specific ATS because these gates just entail a dependency among the trigger component and the triggered ones. Therefore, the logic implemented with a FDEP gate is modeled through the relations existing between the ATS models of the BEs inputs of the gate.

In Figure 8.c the FDEP gate FD is added to the model of Figure 8.b. In this model BE2 is not anymore of class C2, but C4. Thus, its ATS model is made up also of the F-ATS that is dependent on the H-ATS of BE3, i.e., its trigger component.

Table 1.c shows that a F-ATS can depend on any other ATS object except O-ATS. In fact, the trigger elements can be represented by any kind of RDFT object, as in the setting of extended DFT [17].

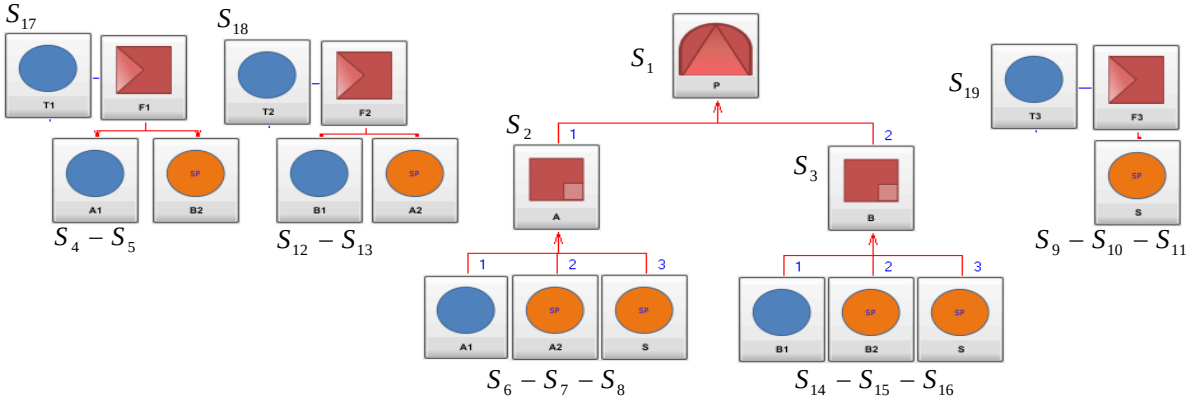
4.6 ATS Model of SEQ Gates

A SEQ gate with *N* inputs is converted in an ATS with *N*+1 states. *N* of them represent the failure of an input of the gate, while the remaining one the failed condition of the gate. Restorations at intermediate levels are allowed; restoring the inputs of the SEQ "as new" (see Figure 8.d). In the current release of RAATSS SEQ gates can accept only BEs as inputs.

5 CASE STUDY

The results of the analysis executed with RAATSS on a case study found in literature [16] are reported in this section. The DFT of the case study (Figure 9) results of particular interest for the presence of multiple SPARE and FDEP gates. The tree is composed by few elements that are tightly interconnected by functional dependencies and spare management policies. The two spare systems A and B have 3 inputs each (A1, A2 and S for A and B1, B2 and S for B), with the third being shared. All components are subjected to the effect of three triggers T1, T2 and T3. This case study presents another interesting peculiarity: the existence of two cases of non-determinism: (*i*) when T1 and T2 fail in succession; and (*ii*) when S is restored while A and B are in the failed state. In these cases there is no knowledge a priori of which spare subsystem will activate S.

Following the conversion rules introduced in the previous section, RAATSS generates an ATS model for each element of the RDFT. First the structures of the ATS models are generated and then the transition functions of each ATS are defined on the basis of the dependencies existing between ATSs.



DFT OBJECT		DFT OBJECT *label*	Healthy-ATS	Operative-ATS	Functional-ATS	SPARE-ATS	GATE-ATS
PAND		P					S1 2 states (UP/DN) *inst* trans fully conn.
SPARE		A, B				S2, S3 2 states (UP/DN) *inst* trans fully conn.	
BE	**C1**	T1. T2. T3	S17, S18, S19 2 states (W/F) *timed* trans fully conn.				
	C3	A1, B1	S4, S12 2 states (W/F) *timed* trans fully conn.		S5, S13 2 states (ON/OFF) *inst* trans fully conn.		
	C4	A2, B2, S	S6, S14, S9 2 states (W/F) *timed* trans fully conn.	S7, S15, S10 2 states (Act/S-by) *pass* trans fully conn.	S8, S16, S11 2 states (ON/OFF) *1 pass & 1 inst* trans fully conn.		

5.1 ATS Model

The conversion engine RDFT-to-ATS of RAATSS generates a group of ATS models for each element of the tree. In particular there will be one ATS model for each gate (except for FDEP gates) and a set of ATS models (from 1 to 3) for each BE depending on their position in the tree. In Figure 9 are shown also the labels of the generated ATS for the various objects of the DFT. The structure of the ATS models is reported in Table 2 where is shown, for each DFT object, the ATS models that define the behavior. In the next subsections we describe the characteristics of some of the ATS models described above.

5.1.1 ATS Model of the PAND gate P

The ATS model of the PAND gate P (S_1) is shown in Figure 10. In the figure are shown the state transition diagram (Figure 10.a) and the functional-Tree (*f*-T) of the activation variables, $\boldsymbol{a}_{1,1\to 2}$ and $\boldsymbol{a}_{1,2\to 1}$ of the transition $T_{1,1\to 2}$ and $T_{1,2\to 1}$, respectively (Figure 10.b and 10.c). Functional-Trees are a graphical representation of the transition variables (i.e., activation, $\boldsymbol{a}$, reactivation, $\boldsymbol{r}$, and parameter, $\boldsymbol{\theta}$).

S_1 can switch from $Q_{1,1}$ to $Q_{1,2}$ only if the conditions that trigger a PAND gate occur. These conditions are expressed in the activation function of the instantaneous transition $T_{1,1\to 2}$. In Figure 10.b is shown that the activation variable $\boldsymbol{a}_{1,1\to 2}$ takes on value 1 if: (*i*) both S_2 and S_3 are in their fourth state ($Q_{i,4}$ $i = 2,3$; the states representing the fact that no component is *active* in any of the two SPARE gates A and B); and (*ii*) S_2 has entered the *failed* state before S_3.

The first conditions is true if the state indicator variables $\boldsymbol{s}_{2,4}$ and $\boldsymbol{s}_{3,4}$ take on value 1, while the second condition is verified if the maximum of the transition-timing variables ($\boldsymbol{\pi}_{2,1\to4}, \boldsymbol{\pi}_{2,2\to4}, \boldsymbol{\pi}_{2,3\to4}$) of the transitions entering $Q_{2,4}$ is less than the maximum of the transition-timing variables ($\boldsymbol{\pi}_{3,1\to4}, \boldsymbol{\pi}_{3,2\to4}, \boldsymbol{\pi}_{3,3\to4}$) of the transitions entering $Q_{3,4}$.

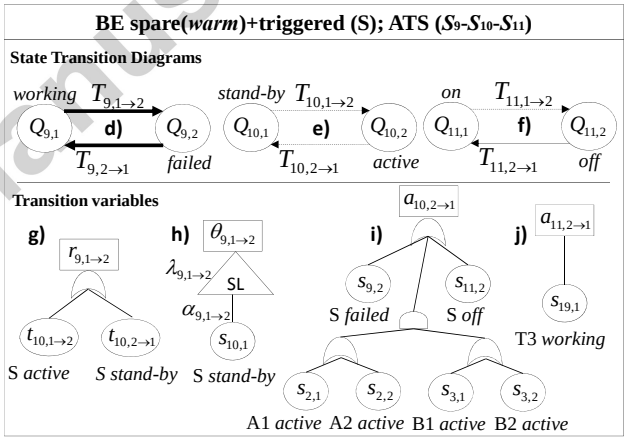
On the other hand, as shown in Figure 2.c, the transition from the *failed* state of S_1 to the *working* state is active ($\boldsymbol{a}_{1,2\to1}=1$) if one of the two transition systems S_2 or S_3 is not in the failed state.

Figure 10. ATS model of the PAND gate P.

5.1.2 ATS Model of the spare component S

Component S is represented in terms of ATS by three ATS models that we call *healthy* (S_9), *occupancy* (S_{10}) and *powered* (S_{11}) as described in the previous section. The model is shown in Figure 11.

The powered-ATS (Figure 11.f) is dependent on the ATS model of the trigger T3 that consists of a single healthy-ATS model (S_{19}). RAATSS allows also considering more complex situations where the trigger is a triggered element, too, or, as in the setting of extended-DFT, is a more complex structure.



In this case, since S is subjected only to the effect of one trigger, the passive transition $T_{11,1\to2}$ has only one parent transition, that is: the "failure" transition of the trigger T3 ($T_{19,1\to2}$).

On the other hands, the activation variable $\boldsymbol{a}_{11,2\to1}$ is defined for the instantaneous transition $T_{11,2\to1}$ (Figure 11.j). It takes on value 1 only if the trigger has been restored ($\boldsymbol{s}_{11,1}=1$). The choice to use an instantaneous transition (instead of using a passive) is dictated by the fact the component could be subjected to the action of more triggers. This, in turn, requires that all the triggers will be in the working state for the activation variable to take on value 1.

The healthy-ATS (Figure 11.d) is made up of two timed transitions and is dependent on the state of the occupancy-ATS. Moreover, the kind of dependency is dictated by the fact that component is a warm or a cold stand-by. While the "repair" transition has fixed transition variables ($\boldsymbol{r}_{9,2\to1}=0, \boldsymbol{\theta}_{9,2\to1}=\mu$ and $\boldsymbol{a}_{9,2\to1}=1$ if repairable and $\boldsymbol{a}_{9,2\to1}=0$ if not) for the "failure" transition only the activation variable ($\boldsymbol{a}_{9,1\to2}=1$) is fixed (warm st-by).

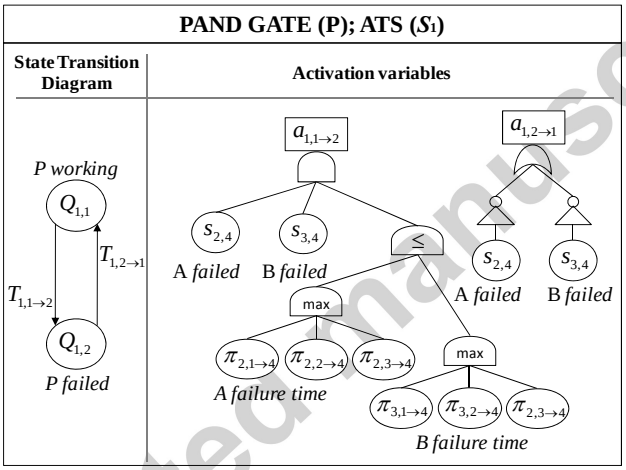
The parameter variable is equal to the nominal parameter ($\lambda_{9,1\to2}$) of the transition scaled by a dormancy factor ($\alpha_{9,1\to2}$) by the relation $\boldsymbol{\theta}_{9,1\to2}=\lambda_{9,1\to2}\,C\alpha_{9,1\to2}^{\boldsymbol{s}_{10,1}}$ (implemented by the SL gate in the *f*-T of Figure 11.h). Thus, the parameter of the transition

is scaled if the component is in the *stand-by* state and it is not if the component is in the *active* state. The logic implemented makes use of the notion of dormancy factor defined by DFT. However, more complex relation where the component takes on a different parameter depending on the SPARE subsystem where it is active, could also had been considered. A straightforward further development, therefore, is to extend the logic to consider load sharing subsystems.

Finally, the reactivation variable (Figure 11.g) takes on value 1 every time a change in the occupancy-ATS occurs $\left(\boldsymbol{r}_{9,1\to 2} = OR(\boldsymbol{t}_{11,1\to 2}, \boldsymbol{t}_{11,2\to 1})\right)$.

The parents of the instantaneous transition ($T_{10,1\to 2}$) of the occupancy-ATS S_{10} (Figure 11.e) are all the transitions ($T_{i,k\to 3}\, i = 2,3; k = 1,2,4$) in the ATS models of the SPARE gates (S_2 and S_3) where the component is input that lead to the state of these SPARE-ATSs where the component is regarded as *active* ($Q_{2,3}$ and $Q_{3,3}$).

On the other hand the instantaneous transition $T_{10,2\to 1}$ is active when either S becomes *unavailable* or because it is not needed anymore by any SPARE gate (a preceding component has been restored). In Figure 11.i is shown that RAATSS uses an AND gate to model the last part of this logic. Inputs of this AND gate are two OR gates with inputs all the state-indicator variables of the states of the SPARE-ATSs that identify the *active* condition of the preceding components in each SPARE gate where S is input. This logic is



used to avoid the re-sampling of the time to failure of the component when switching between *active* states of different SPARE gates. To explain this fact let us consider the case where S is active in the SPARE gate A while the SPARE gate B is in the failed state. If A1 becomes available, in order to respect the priority rule, A will switch to the state A1 active. However, since now S is available for B, we require that S remains in its *active* state.

Figure 11. ATS model of the spare component S.

5.1.3 ATS Model of the SPARE Gate A

The ATS model of the SPARE gate A (S_2) is shown in Figure 12. In the figure are shown the state transition diagram (Figure 12.a) and all the *f*-Ts of the activation variables of S_2 (Figure 12.b to 12.f). S_2 is made up of four states that indicate which component is active at a given time. In particular, $Q_{2,1}$ represents the fact that A1 is active, $Q_{2,2}$ that A2 is active, $Q_{2,3}$ that S is active and $Q_{2,4}$ that there are not active components, i.e., A is in the failed state.

The activation variables $\boldsymbol{a}_{2,2\to 1}$, $\boldsymbol{a}_{2,3\to 1}$ and $\boldsymbol{a}_{2,4\to 1}$ take on value 1 if A1 is *available*. This condition is true when the state-indicator variables $\boldsymbol{s}_{4,1}$ and $\boldsymbol{s}_{5,1}$ representing the *working* the *on* conditions of A1, respectively (Figure 12.b) are both true.

The activation variables $\boldsymbol{a}_{2,1\to2}$, $\boldsymbol{a}_{2,3\to2}$ and $\boldsymbol{a}_{2,4\to2}$ take on value 1 if A1 is *unavailable* and A2 is *available*. These conditions are represented by the *f*-T in Figure 12.c. In particular, A1 is *unavailable* if at least one of the state-indicator variables $\boldsymbol{s}_{4,2}$ (representing the *failed* condition) and $\boldsymbol{s}_{5,2}$ (representing the *off* condition) take on value 1. On the other hands, A2 is *available* if both the state indicator variables $\boldsymbol{s}_{6,1}$ (representing the *working* condition) and $\boldsymbol{s}_{8,1}$ (representing the *on* condition) take on value 1. It is worth noticing that the logic preserves the priority rule of ordering of the inputs of SPARE gates, e.g., if S_2 is in state $Q_{2,3}$ and A2 becomes available, S_2 switches to the state $Q_{2,2}$ regardless of the state of S.

The activation variables $\boldsymbol{a}_{2,1\to3}$, $\boldsymbol{a}_{2,2\to3}$ and $\boldsymbol{a}_{2,4\to3}$ take on value 1 if A1 is *unavailable*, A2 is *unavailable* and S is *available*. These conditions are represented by the *f*-T in Figure 12.d. A2 is *unavailable* if at least one of the state-indicator variables $\boldsymbol{s}_{6,2}$ (representing the *failed* condition) and $\boldsymbol{s}_{8,2}$ (representing the *off* condition) take on value 1. Differently from the case "A2 *available*", for what concerns S we need to consider also the fact that the component must be in the *stand-by* state in order to be *available*. This is due to the fact that S is shared between the two SPARE gates A and B and in order to be *available* for A it cannot be *active* in B. Thus, S is *available* if the state indicator variables $\boldsymbol{s}_{9,1}$ (representing the *working* condition), $\boldsymbol{s}_{10,1}$ (representing the *stand-by* condition) and $\boldsymbol{s}_{11,1}$ (representing the *on* condition) take on value 1.

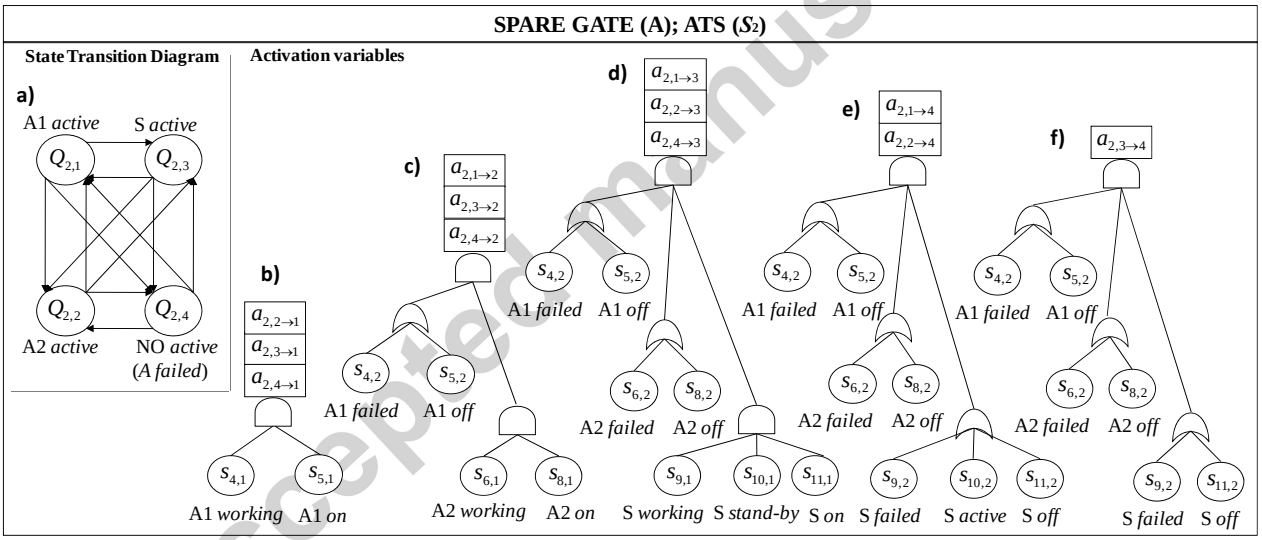


Figure 11.ATS model of the SPARE gate A.

Finally, the activation variables $\boldsymbol{a}_{2,1\to4}$, $\boldsymbol{a}_{2,2\to4}$ and $\boldsymbol{a}_{2,3\to4}$ take on value 1 if all the components, A1, A2 and S are *unavailable*. In this case, however, we need to distinguish between two different situations due to the fact that S is shared between A and B (see Figure 12.e and 12.f) and, thus the condition S active must take into account in which SPARE it is active, i.e., if S_2 is in state $Q_{2,3}$, the component is certainly *active* and, in particular, it is *active* in A.

With the proposed logics RAATSS is provided by a powerful mean to manage SPARE gates with shared inputs in presence of repairable components.

As a last remark, the authors would like to highlight the presence of non-determinism that arises when more SPARE gates require a component that has just been repaired. For instance, let consider the case where all the components of the system are *failed*. In this situations the SPARE-ATS models of A and B are in the *failed* states $Q_{2,4}$ and $Q_{3,4}$, respectively. Suppose now that S is restored. In this case, both the transitions $T_{2,4\to3}$ and $T_{3,4\to3}$ will be enabled simultaneously and a non-deterministic choice will arise. We have decided to solve any non-determinism that arises when more transitions are enabled (or complete) simultaneously

with a probabilistic setting, i.e., selecting randomly the transition that will complete first. This clarification is important also because the completion of a transition may disable another previously simultaneous enabled transition.

Finally, we conclude this section highlighting that the non-deterministic setting in SPARE gates can be substituted by other kind of logics like FIFO and LIFO disciplines or other kind of specific priority rules. The same consideration applies also for what concerns the priority rule stated in Section 2.1 about the ordering of inputs of SPARE gates. For instance, the concept of primary component could be removed leading to a system where a component is kept in the *active* state as long as it can perform its functions.

5.2 Results and Validation

Figure 13 shows the results obtained running the simulation engine of RAATSS in the case of an AND gate at the top of the tree (left-hand side) and in the case of a PAND at the top (right-hand side).

In figure 13 the values of the unreliability, unreliability with repair (i.e., the top event is a failure gate) and unavailability as a function of time are shown. Failure rates are shown in Table 3. Moreover, all the spare components are in cold stand-by.

As expected, the probability of top event is decreasing with respect to the level of reparability of the system. Moreover, the system with the PAND gate at the top of the tree is more reliable due to the additional constraint of the gate.

Table 3: parameters of the case study (tu: time units)

Event	Failure Rate [tu^{-1}]	Repair Rate [tu^{-1}]
A1	0.001	0.025
A2	0.005	0.025
B1	0.002	0.025
B2	0.0035	0.025
S	0.005	0.025
T1,T2,T3	0.003	0.025

The reliability estimation in the case of AND gate at the top of the tree is in accordance with the value reported in [14]. In the case of a PAND at the top of the tree results fit with the ones given by MatCarloRe [11,12]. Unfortunately at the state of the art there are not available tools, both commercial and academic, that make use of DFT as the main language of system description which are able to handle repairable spare components and triggers. In order to overcome this issue, simulations were also conducted using Mobius [33] after the conversion of the ATS model into SAN [38]. Although it does not validate the logic implemented by RAATSS in case of repairable components, these results confirm the reliability of tool itself.

The time of computation for 10^7 iterations was of 4824 seconds, running in a laptop with the following characteristics: CPU, Intel Core 2 Duo 1.83 GHz; RAM, 1.99 GB.

6 CONCLUSION

In this paper the RAATSS tool, a Matlab® toolbox based on ATS for the evaluation of repairable DFT (RDFT) was presented. In order to define an ATS model of a RDFT the issues of the dynamic gates when repairable components are involved was analyzed, in particular for spare management and functional dependencies. The formalization of well defined behavioral rules has been used for the definition of a standard approach to convert RDFT into ATS models which was incorporated in an automated translator engine within the RAATSS environment. Exploiting the ATS formalism the objective to compute the availability of complex RDFTs was reached; moreover, with the introduction of the failure gates, the tool gives the ability to model and evaluate the first occurrence of a top event when both repairable and non-repairable subsystems are considered. In this context, RAATSS can be considered as a first step towards the definition of an extended DFT methodology; in fact, exploiting the modularity and flexibility of ATS, the implementations of several features would be possible: share loading, finite maintenance resources, imperfect coverage of control systems, the possibility to add new kind of gates are only few of the possible enrichments that can be provided.

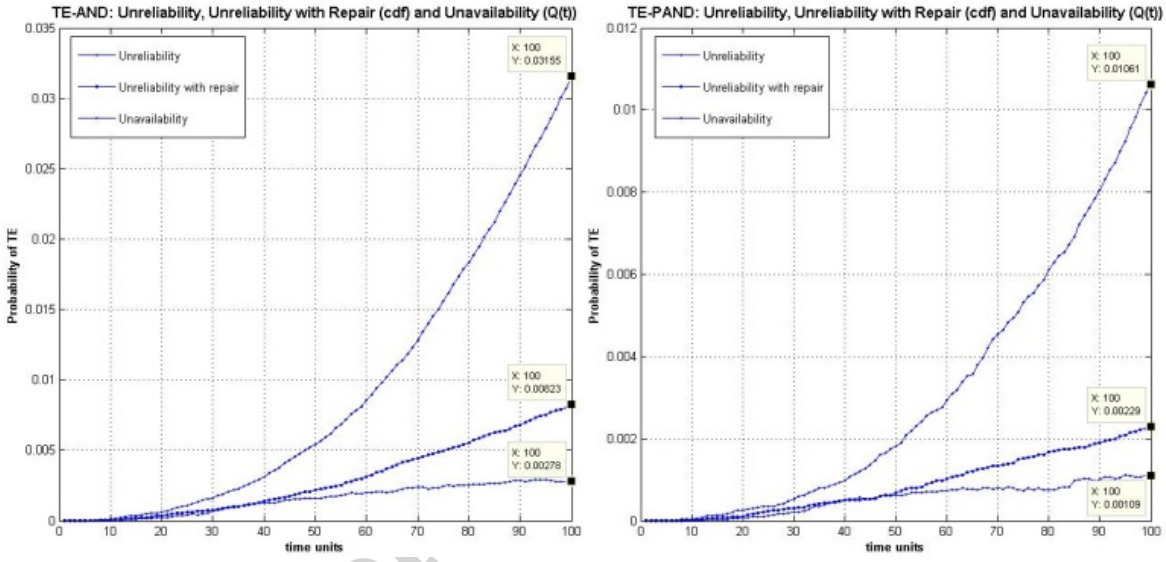


Figure 12.Unreliability, Unreliability with repairs, and Unavailability. Left: AND top; Right: PAND top

So far, the results obtained have been validated through the comparison with existing cases but, thanks to the nature of the ATS, an ATS model can be analyzed through a state space exploration making possible the use of the traditional analytical approach based on the Markov chains. This will be one of the objectives of next developments.

For these reasons, RAATSS represents an important addition to the well know formalisms (and their tool implementations) for stochastic modeling of complex system because it offers a standardized way to build model of complex systems. Thanks to its intuitiveness, the tool will make much more likely - for domain experts - to engage more closely the scrutinizing of the models and, thus, improve their quality and, what is more worth, mark the transition from static to dynamic stochastic modeling also in the industrial world.

For more information about the RAATSS tool, please visit: http://www.ii17.diim.unict.it/projects/ or send an email to Dr. Ferdinando Chiacchio or Dr. Gabriele Manno.

REFERENCES

[1] Stamatelatos M., Vesely W, Dugan J B., Fragola J., Minarick J. and Railsback J. Fault tree handbook with aerospace applications, *NASA Office of Safety and Mission Assurance*, Washington. DC, 2002.

[2] Xing L. and Amari S V. Fault Tree Analysis. Handbook of Performability Engineering, 595-620. Springer-Verlag London Limited, 2008.

[3] Chiacchio F., Compagno L., D'Urso D., Manno G. and Trapani N., Dynamic fault tree resolution: a conscious trade-off between analytical and simulative approaches, *Reliability Engineering and System Safety*, 96(11), 1515-1526, 2011.

[4] Amari S V., Myers A F., Rauzy A. and Trivedi K S. Imperfect Coverage Models: Status and Trends. Handbook of Performability Engineering, 321-348. Springer-Verlag London Limited, 2008.

[5] Dugan J B., Bavuso S J. and Boyd M A. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability, 41(3), 363-377, 1992.

[6] Distefano S. and Puliafito A. Dynamic reliability block diagrams vs dynamic fault trees. In Proceedings Annual Reliability and Maintainability Symposium RAMS '07, 71-76, 2007.

[7] Bouissou M. and Bon J L. 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, 2003.

[8] Codetta-Raiteri D., Franceschinis G., Iacono M. and Vittorini V., Repairable Fault Tree for the automatic evaluation of repair policies*, Int. Conf. on Dependable Systems and Networks*, 2004

[9] Bouissou, M., A Generalization of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP). *in Proceedings of the16th European Safety and Reliability Conference (ESREL'07)*, 2007

[10] Boudali H., Crouzen P. and Stoelinga M. 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, 2007.

[11] Manno G., Chiacchio F, Compagno L., D'Urso D. and Trapani N., MatCarloRe: an integrated FT and Monte Carlo Simulink tool for the reliability assessment of dynamic fault tree, *Expert Systems with Applications*, Volume 39, Issue 12, 15 September 2012, Pages 10334–10342.

[12] Chiacchio F., Compagno L., D'Urso D., Manno G. and Trapani N., An open-source application to model and solve dynamic fault tree of real industrial systems, *in proceedings of SKIMA*, Benevento 8-11 September 2011, ISBN: 9781467302487.

[13] Manno G., Reliability Modelling of complex systems: an adaptive transition system approach to match accuracy and efficiency*. PhD Thesis, University of Catania*, 2012.

[14] Vaurio J K. Fault Tree analysis of phased mission systems with repairable and non-repairable components, *Reliability Engineering and System Safety*, 74, 169-180, 2001.

[15] Gulati R. and Dugan J B. A modular approach for analyzing static and dynamic fault trees. *Proc. Ann. Reliability and Maintainability Symposium*, 57- 63, 1997.

[16] Boudali H. and Dugan J B. A New Bayesian Network Approach to Solve Dynamic Fault Trees, in *Proceedings Annual Reliability and Maintainability Symposium*, 451-456, 2005.

[17] Boudali H., Nijmeijer A.P. and Stoelinga M.I.A., DFTsim: a simulation tool for extended dynamic fault trees, in *proceeding of SpringSim '09. 2009*.

[18] Codetta Raiteri D., The conversion of dynamic fault trees to stochastic petri nets, a case of graph transformation. *Electronic notes in theoretical computer science*, 162(2), 45-60, 2004.

[19] Chiacchio F, Cacioppo M., Manno G., D'Urso D., Trapani N and Compagno L., A Weibull-based modular approach for the reliability of Dynamic Fault Trees, *Reliability Engineering and System Safety*, Volume 109, January 2013, Pages 45–52.

[20] Manno G., Chiacchio F. *, D'Urso D., Trapani N. and Compagno L., RAATSS, an extensible Matlab® toolbox for the evaluation of repairable dynamic fault trees, *11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference 2012 (PSAM11 ESREL 2012),* pp. 1055-1064, ISBN: 9781622764365, Curran Associates, Inc., June 2012, Helsinki, Finland.

[21] Codetta-Raiteri D., Integrating several formalisms in order to increase Fault Trees'modeling power, *Reliability Engineering and System Safety*, Volume 96 2011, pp. 534-544.

[22] Boudali H., Crouzen P. and Stoelinga M., A Compositional Semantics for Dynamic Fault Trees in Terms of Interactive Markov Chains, *Proc. Fifth Int. Symp. Automated Technology for Verification and Analysis*, pp.441 -456, 2007

[23] Relex® Reliability Studio Reference Manual, 2008, Relex Software Corporation, Greensburg, PA 15601, info@relex.com, www.relex.com

[24] Merle G., Roussel J. M. and Lesage J. J., Improving the Efficiency of Dynamic Fault Tree Analysis by Considering FDEP Gates as Static, *19th European Safety & Reliability Conf. (ESREL'10), Rhodes (Greece),* pp. 845-851, September 2010

[25] Montani S., Portinale, L.; Bobbio, A. and Codetta-Raiteri, D., 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.

[26] Dugan J.B., Venkataraman B. And Gulati R., DIFTree: a software package for the analysis of dynamic fault tree models. *in Proceedings Annual Reliability and Maintainability Symposium* : 64-70, 1997.

[27] Dugan J.B. And Sullivan K.J., Developing a Low-Cost High-Quality Software Tool for Dynamic Fault-Tree Analysis, 2000, *IEEE Transactions on Reliability*, 49 (1):49-58.

[28] Sullivan K.J., Dugan J.B. And Coppit D., The Galileo fault tree analysis tool, *in Proc. Digest of Papers Fault-Tolerant Computing Twenty-Ninth Annual International Symposium*: 232-235, 1999.

[29] Trivedi K.S., Sahner R., Reliability modeling using Sharpe. *IEEE Transactions on Reliability*, R-36:186-192, 1987.

[30] Hillston J. A Compositional Approach to Performance Modelling. Cambridge University Press, 1996.

[31] Derisavi S., Hermanns H. and Sanders W.H., Optimal State-Space Lumping in Markov Chains, *Information Processing Letters*, vol. 87, no. 6, September 30, 2003, pp. 309-315.

[32] Lanus M. and Trivedi K.S., 2003, Hierarchical composition and aggregation of state-based availability and performability models, *IEEE Transactions on Reliability*, 52 (1): 44-52.

[33] PERFORM. Möbius: Model Based Environment for Validation of System Reliability, Availability, Security and Performance. User's Manual, v. 2.0 Draft, 2006.

[34] Bon, J. and Collet, J.: An algorithm in order to implement reliability exponential approximations, *Reliability Engineering & System Safety* 43 (1994), Nr. 3, S. 263-268

[35] Collet, J. and Renault, I.: Path probability evaluation with repeated rates, *Reliability and Maintainability Symposium*, 1997 Proceedings, Annual., 1997, S. 184-187

[36] Bouissou, M. and Lefebvre, Y.: A path-based algorithm to evaluate asymptotic unavailability for large Markov models, in *Proceedings of the 48th Reliability and Maintainability Annual Symposium* (RAMS'02)., 2002, S. 32-39

[37] Hermanns H., Interactive Markov Chains, *Lecture Notes in Computer Science*, 2428. Springer, 2002.

[38] Sanders W H. and Meyer J F. Stochastic activity networks: Formal definitions and concepts, *Lectures on Formal Methods and Performance Analysis*. Springer Verlag, 2002.

[39] Sanders W H., Construction and Solution of performability models based on stochastic activity networks. *PhD Thesis*, University of Michigan, 1988.

[40] Kwiatkowska M., Norman G. and Parker D., PRISM: probabilistic model checking for performance and reliability analysis, *ACM Sigmetrics Performance evaluation review*, 36(4), 40-45, 2009.

[41] Glynn P W., On the role of generalised semi-Markov processes in simulation output analysis, *Proceedings of the 1983 Winter Simulation Conference*, 38-42, 1983.

[42] Bouissou, M., Automated Dependability Analysis of Complex Systems with the KB3 Workbench: the Experience of EDF R&D, in *Proceedings of the International Conference on Energy and Environment (CIEM'05).*, 2005.

[43] Bouissou M., Bon J.L., A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes, *Reliability Engineering & System Safety* Volume 82, Issue 2, November 2003, Pages 149−163

APPENDIX

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

$S = \{S_1, \ldots, S_N\}$ 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 $Q_i \in Q$ $(i = 1, 2, \ldots, N)$ and a finite set of NT_i transitions $T_i \in T$ $(i = 1, 2, \ldots, N)$ 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 $Q_{i,j}$ $(j = 1, 2, \ldots, NQ_i)$ and elements of T_i as $T_{i,j \to j+1}$ $(j = 1, 2, \ldots, NT_i - 1)$ or as $T_{i,k}$ $(k = 1, 2, \ldots, NT_i)$.

$f_S : S \to V_S$, 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; $f_S = \{s, t, \pi\}$.

$f_R : V_S \to \mathcal{R}$, is a set of reward functions defined over the set of system variables $\boldsymbol{v}_S \in V_S$;

$f_T : V_S \to V_T$ is a set of transition functions defined over $\boldsymbol{v}_S \in V_S$; $f_T = \{a, r, \theta\}$.

$\boldsymbol{q}(0) \in Q$, is the initial state. $\boldsymbol{q}_i(t) \in Q_i$ is the *state variable* of S_i at time *t*. As the system moves at point in times, we are interested in $\boldsymbol{q}_i(n)$, the values of these variables as a state-transition occurs. With $\boldsymbol{q}_i(n) = Q_{i,j}$ we say that during the interval $I_n = [\tau_n, \tau_{n+1}[$, S_i is in state $Q_{i,j}$.

Let $\boldsymbol{Z} = (\boldsymbol{q}_i(\boldsymbol{n}), \boldsymbol{\tau}_n)$ be the underlying stochastic process of an ATS where: $\boldsymbol{q}_i(\boldsymbol{n})$ is the random variable representing the state of $\boldsymbol{S}_i$ after the n^{th} event; and $\boldsymbol{\tau}_n \in \boldsymbol{\mathcal{R}}^+$ is the random variable registering the instant of occurrence of the n^{th} event.

System variables

System variables reorganize the information about the state of the system. We define three classes of system variables:

the *state-indicator* variable of $\boldsymbol{Q}_{i,j}$ is defined by

$$\boldsymbol{s}_{ij}(\boldsymbol{n}) = \begin{cases} 1, & if \quad \boldsymbol{q}_i(n) = \boldsymbol{Q}_{i,j} \\ 0, & otherwise \end{cases} ; \qquad (1)$$

- the *transition-indicator* variable of $\boldsymbol{T}_{i,j \to j+1}$ is defined by

$$\boldsymbol{t}_{i,j\to j+1}(\boldsymbol{n})=\begin{cases}1, & if \quad \boldsymbol{q}_i(\boldsymbol{n})=\boldsymbol{Q}_{i,j+1}\wedge\boldsymbol{q}_i(\boldsymbol{n}-1)=\boldsymbol{Q}_{i,j}\\ 0, & otherwise\end{cases} \quad \text{; and} \qquad (2)$$

- the *transition-timing* variable of $\boldsymbol{T_{i,j\to j+1}}$ is defined by

$$\boldsymbol{\pi}_{i,j\to j+1}(\boldsymbol{n})=\begin{cases}\boldsymbol{\tau}_{\boldsymbol{n}}, & if \quad \boldsymbol{t}_{i,j\to j+1}(\boldsymbol{n})=1\\ \boldsymbol{\pi}_{i,j\to j+1}(\boldsymbol{n}-1), & otherwise\end{cases} \qquad (3)$$

Transition variables and attributes

- *type*, specifies the kind of transition and, if timed, the distribution of the time to complete;

activation variable $\boldsymbol{a}_{i,k\to k+1}$, applies to timed, instantaneous and passive transitions;

reactivation variable $\boldsymbol{r}_{i,k\to k+1}$, applies only to timed transitions;

parameter $\boldsymbol{\theta}_{i,k\to k+1}$, applies only to timed transitions;

parent set, $\boldsymbol{P}_{i,k\to k+1}$ applies only to passive transitions.

Enabling condition and time to complete of transitions

A transition $\boldsymbol{T}_{i,j\to j+1}$ is said *enabled* only if the activation variables ($\boldsymbol{a}_{i,j\to j+1}$) take on value 1 and if the state from where transition depart ($\boldsymbol{Q}_{i,j}$) is occupied by the system, i.e., $\boldsymbol{q}_i=\boldsymbol{Q}_{i,j}$.

$$\boldsymbol{en}_{i,j\to j+1}=\boldsymbol{a}_{i,j\to j+1}\,\mathrm{C}\boldsymbol{s}_{i,j}, \qquad (4)$$

A transition can complete only if *enabled*. The *time to complete* of a transition is defined as

$$\boldsymbol{ttc}_{i,j\to j+1}(n)=\begin{cases}+\infty, & if \quad \boldsymbol{en}_{i,j\to j+1}(n)=0\\ type_{i,j\to j+1}\left(\boldsymbol{\theta}_{i,j\to j+1}(n)\right)+\tau_n, & if \quad \boldsymbol{en}_{i,j\to j+1}(n)=1\wedge\\ & \wedge\left(\boldsymbol{r}^{m}_{i,j\to j+1}(n)=1\vee\boldsymbol{ttc}_{i,j\to j+1}(n-1)=+\infty\right)\\ \boldsymbol{ttc}_{i,j\to j+1}(n-1), & if \quad \boldsymbol{en}_{i,j\to j+1}(n)=1\wedge\\ & \wedge\boldsymbol{r}_{i,j\to j+1}(n)=0\wedge\boldsymbol{ttc}_{i,j\to j+1}(n-1)<+\infty\end{cases} \qquad (5)$$

Choice of the next transition and update of the state variables

Given the time to complete $\boldsymbol{ttc}_x(n)$ $(x=1,2,\ldots,NT)$ 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 $T'(n)=\left\{T_x'\in T \mid \forall\ T_x\in T,\ \boldsymbol{ttc}_x'(n)\le\boldsymbol{ttc}_x(n)\right\}$ denote the set of transitions with minimal time to complete. We non-deterministically select T_x^* from the set $T'(n)$. It follows that that $\tau_{n+1}=\boldsymbol{ttc}_x'(n)$.

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

$$\boldsymbol{c}_{i,j\to j+1}(n) = \begin{cases} 1, & if \quad X(n) = X_{i,j\to j+1}, \quad \forall\ i, j,: \ T_{i,j\to j+1} \in T. \\ 0, & otherwise \end{cases} \qquad (6)$$

Given $\boldsymbol{c}_{i,j\to j+1}(n) = 1$ the state variable of the *i*-th ATS at time step τ_{n+1} is updated following:

$$\boldsymbol{q}_i(n+1) = \begin{cases} Q_{i,j+1}, & if \quad \boldsymbol{c}_{i,j\to j+1}(n) = 1, \ \forall i, j : T_{i,j\to j+1} \in T \\ \boldsymbol{q}_i(n), & otherwise \end{cases} . \qquad (7)$$

If the chosen transition is parent of passive transitions, those that are enabled are completed too.

HIGHLIGHTS

- A semantic for Repairable Dynamic Fault Tree (RDFT) was conceived;
- Practical motivation for the use of RDFT is presented;
- The conception of failure gates for the computation of the availability and reliability of complex sytems is shown;
- A mapping from DFT entity to ATS entity is proposed;
- RAATSS, an automated tool for the evaluation of RDFT, is presented;

Table 1 (a) Classification of BE, (b) type of ATS for BE classes; and (c) attributes to consider for each type of ATS.

a. Classification of BEs			
Input of SPARE\FDEP	not	Primary	secondary
not	C1	C1	C3
primary	C1	C1	C3
secondary	C2	C2	C4

b. ATS assignment to classes of BEs			
Classes\ATS	H-ATS	O-ATS	F-ATS
C1	X		
C2	X (D)	X	
C3	X		X
C4	X (d)	X	X

c. ATS objects available in RAATSS and main characteristics							
Type of ATS	H-ATS	H^d-ATS	O-ATS	F-ATS	G-ATS	SPARE-ATS	SEQ-ATS
depends on or might depend on				H-ATS	H-ATS	H-ATS	
				H^d-ATS	H^d-ATS	H^d-ATS	

			H^d-ATS	F-ATS	F-ATS	F-ATS	
	–	O-ATS	SPARE-ATS	G-ATS	G-ATS	G-ATS	H-ATS
				SPARE-ATS	SPARE-ATS	SPARE-ATS	
				SEQ-ATS	SEQ-ATS	SEQ-ATS	
Type of transitions	■2 timed	■2 timed	■1 passive	■2 passive	■all inst	■2 inst	■all timed
			■1 inst				
Important attributes		■Activation	■Parents				
	■–	■Reactivation	■Activation	■Parents	■Activation	■Activation	■–
		■Parameter		■Activation			