Supporting Reliability Engineers in Exploiting the Power of Dynamic Bayesian Networks*

Luigi Portinale, Daniele Codetta Raiteri, Stefania Montani

Dipartimento di Informatica, Università del Piemonte Orientale
Via Bellini 25g, 15100 Alessandria (ITALY)

Abstract

In this paper, we present an approach to reliability modeling and analysis based on the automatic conversion of a particular reliability engineering model, the Dynamic Fault Tree (DFT), into Dynamic Bayesian Networks (DBN). The approach is implemented in a software tool called RADYBAN (Reliability Analysis with DYnamic BAyesian Networks). The aim is to provide a familiar interface to reliability engineers, by allowing them to model the system to be analyzed with a standard formalism; however, a modular algorithm is implemented to automatically compile a DFT into the corresponding DBN, In fact, when the computation of specific reliability measures is requested, classical algorithms for the inference on Dynamic Bayesian Networks are exploited, in order to compute the requested parameters. This is performed in a totally transparent way to the user, who could in principle be completely unaware of the underlying Bayesian Network. The use of DBNs allows the user to be able to compute measures that are not directly computable from DFTs, but that are naturally obtainable from DBN inference. Moreover, the modeling capabilities of a DBN, allows us to extend the basic DFT formalism, by introducing probabilistic dependencies among system components, as well as the definition of specific repair policies that can be taken into account during the reliability analysis phase. We finally show how the approach operates on some specific examples, by describing the advantages of having available a full inference engine based on DBNs for the requested analysis tasks.

Key words: Dynamic Bayesian Networks, Dynamic Fault Trees, Repair Policies, Reliability Analysis, DBN Inference.

* This work has been partially supported by the EU-Project Crutial IST-2004-27513.

Email address: {portinal,raiteri,stefania}@di.unipmn.it (Luigi Portinale, Daniele Codetta Raiteri, Stefania Montani).

Acronym list:

2TBN	2 Time-Slice Temporal Bayesian Network
BK	Boyen-Koller algorithm
BN	Bayesian Network
CAS	Cardiac Assist System
Cov	Coverage set
Cov_{BE}	basic Coverage set
CPT	Conditional Probability Table
CR	Component Repair policy
CTMC	Continuous Time Markov Chain
DBN	Dynamic Bayesian Network
DFT	Dynamic Fault Tree
FDEP	Functional DEPendency gate
FT	Fault Tree
JT	Junction Tree inference
PAND	Priority AND gate
PDEP	Probabilistic DEPendency gate
RB	Repair Box
SGR	Subsystem Global Repair policy
SLR	Subsystem Local Repair policy
TE	Top Event
WSP	Warm SPare gate

1 Introduction

It is well-known that the modeling possibilities offered by *Fault Trees* (*FT*), one of the most popular techniques for dependability analysis of large, safety critical systems, can be extended by relying on *Bayesian Networks* (*BN*) [1,3,13,21,23,20]. In [15], we have shown how BNs can provide a unified framework in which also *Dynamic Fault Trees* (DFT) [8], a rather recent extension to FTs able to treat several types of dependencies, can be represented. However, while reliability engineers are quite familiar with FT-based formalisms, they are not usually comfortable with the use of formalisms like BN and their extensions. This is also due to the fact that, for reliability purposes, simple and modular techniques are often sufficient for the definition of the required analysis framework. Of course, a clear trade-off exists between the simplicity of the formalism and its modeling, as well as analysis capabilities. FTs are maybe the most simple combinatorial formalism in reliability analysis, but they fail in capturing important aspects like several kind of dependencies among the system components [19,1,13]. DFTs overcome some of the limitations of standard FTs, by allowing some kind of dynamic dependencies among

components. They offer a quite simple and structured framework very useful for modeling purposes, but still quite limited from the analysis point of view, especially if more complex modeling features (like for instance repair policies or special probabilistic dependencies) are required by the application.

The starting point of our work is to make available to reliability engineers a tool where they can take advantage of the simplicity and modularity of either plain FTs or DFTs, by making them available at the same time a more powerful analysis engine based on *Dynamic Bayesian Networks* (DBN). In fact, the quantitative analysis of DFTs typically requires to expand the model in its whole state space, and to solve the corresponding Continuous Time Markov Chain (CTMC) [8]. The above approach, even if can be improved through modularization techniques [10], usually suffers from a state explosion problem. With respect to CTMC, the use of a DBN allows one to take advantage of the factorization in the temporal probability model, via the conditional independence assumptions represented in the DBN. Moreover, by taking into account the basic features of DBNs, we aim at offering to the user a set of useful extensions to the basic DFT formalism, like the introduction of sophisticated probabilistic dependencies among the failure of system components, the introduction of specific repair policies and the possibility of analyzing the system behavior under the gathering of specific observations on the system components. Our approach is based on a translation of the extended DFT model into an equivalent DBN. and has been implemented in a tool called RADYBAN (Reliability Analysis with DYnamic BAyesian Networks) [16]. It allows the design of the reliability model through a graphical interface where the analyst can access and exploit all the familiar modeling constructs of DFTs, as well as the supported extensions cited above; the resulting model is then compiled into an equivalent DBN and the analysis is performed in a transparent way to the user, who has just to specify the desired type of analysis algorithm. This has the advantage of avoiding the reliability engineer to learn the details of a completely new formalism; in case she/he is not willing to forsake her/his traditional formalism, she/he can still use it (possibly with few simple extensions) while having available a more powerful analysis engine. // The rest of the paper is organized as follows: in section 2 we briefly review the basic frameworks of DFTs, while in section 3 we introduce the extensions we have provided to augment the modeling power of the basic formalism. In section 4 we recall DBN basics and in section 5 we sketch the main functionalities of our approach, in particular by taking into consideration, through a running example, the compilation process concerning the provided modeling extensions. Finally in section 6, we show a more general application of the approach on a case study taken from [4] and based on a real-world system. Conclusions are then reported in section 7.

2 Dynamic Fault Trees

Fault Trees allow one to represent the combination of elementary causes that lead to the occurrence of an undesired catastrophic event named the *Top Event* (*TE*) [1,13]. By specifying failure probabilities on the basic components of the modeled system (the elementary causes of the *TE*, also called *basic events*), the whole system unreliability (probability of the *TE*) at a given mission time can be computed. The model assumes every event to be Boolean, with value `true` corresponding to a failure event.

In recent years, an effort has been documented in the literature, aimed at increasing the modeling power of *FT* by including new primitive gates, able to accommodate complex kinds dependencies. This augmented *FT* language is referred to by the authors as *Dynamic FT* [8,14]. *DFT* introduce three basic (dynamic) gates: the warm spare (*WSP*), the functional dependency (*FDEP*) and the priority AND (*PAND*) [1] .

A WSP dynamic gate models one primary component that can be substituted by one or more backups (spares), with the same functionality (see Fig. 1(a), where spares are identified by "circle-headed" arcs). The WSP gate fails if its primary fails and all of its spares have failed or are unavailable (a spare is unavailable if it is shared and being used by another spare gate). Spares can fail even while they are dormant, but the failure rate of an unpowered (i. e. dormant) spare is lower than the failure rate of the corresponding powered one. More precisely, being λ the failure rate of a powered spare, the failure rate of the unpowered spare is $\alpha\lambda$, with $0 \leq \alpha \leq 1$ called the dormancy factor. Spares are more properly called "hot" if $\alpha = 1$ and "cold" if $\alpha = 0$.

In the FDEP gate (Fig. 1(b)), one trigger event T (connected with a dashed arc in the figure) causes other dependent components to become unusable or inaccessible. In particular, when the trigger event occurs, the dependent components are immediately forced to fail; the separate failure of a dependent component, on the other hand, has no effect on the trigger event. FDEP may also have a non-dependent output, that simply reflects the status of the trigger event; since this dummy output is not used in the analysis, it can be safely ignored.

Finally, the standard DFT formalism provides another gate called PAND (Fig. 1(c)); the PAND gate reaches a failure state if and only if all of its input components have failed in a preassigned order (from left to right in graphical notation). For example, in Fig. 1(c) a failure occurs iff A fails before

[1] Actually a fourth kind of gate is introduced in the standard DFT model, namely the sequence enforcing (*SEQ*) gate; however it can be modeled as a special kind of *WSP* [14], so it will not be considered in the following.

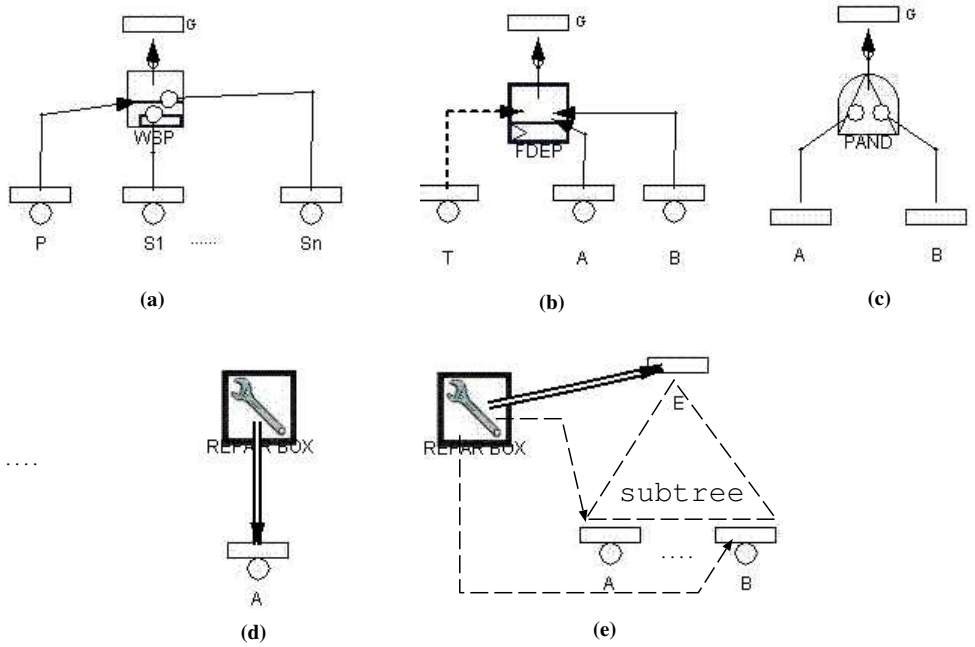

Fig. 1. Dynamic gates in a DFT.

B, but B may fail before A without producing a failure in G.

In the next section we will discuss some particular extensions we have provided to the standard DFT formalism, in particular by generalizing in a probabilistic way the FDEP gate and by introducing repair modeling features.

3 Extending DFTs

We propose some extensions to the DFT formalism, in order to provide the reliability engineer with more flexible modeling features. In particular, we deal with a couple of new modeling primitives: the *Probabilistic Dependency gate* (PDEP), and the RepairBox (RB) [2,6]. The PDEP gate is a generalization of the standard FDEP gate of Fig. 1(b), where one trigger event probabilistically influences the failure of a set of dependent components. Concerning the second extension, the RB allows to model the presence of a repair process in the DFT model, in order to turn a set of failed components back to the working state (Fig. 1(d,e)). The RADYBAN tool has been recently extended, in order to evaluate DFT models where RB or PDEP nodes are present, without adding complexity to the underlying DBN-based analysis engine.

We propose to generalize the FDEP by defining a new gate, called probabilistic dependency (PDEP). In the PDEP, the probability of failure of dependent components, given that the trigger has failed, is parametrized through a specific parameter $0 \leq \pi_d \leq 1$. The meaning of this parametrization is that, if the trigger event occurs, then there is a given chance (determined by π_d) that the failure on a dependent component will occur as well. We assume that if $\pi_d = 0$ then the occurrence of the trigger has no influence on the failure of the dependent components[2], while if $\pi_d = 1$, then the occurrence of the trigger immediately causes the failure of dependent components, modeling the standard FDEP gate.

For any $0 < \pi_d < 1$ we can assume two possible interpretations or semantics of the gate behavior:

- the trigger, once occurred, persists as active in time, increasing the failure rate of the dependent components, thus accelerating their eventual failure;
- the trigger, once occurred, is de-activated, with no future chance of causing the failure of the dependent components, in case such a failure did not actually occur.

We call *persistent PDEP* the former interpretation and *one-shot PDEP* the latter [3].

In case of a persistent PDEP gate, the trigger occurrence is assumed to increase the failure rate λ_A of a dependent component A; in particular, the failure rate of A given that the trigger has occurred is determined as $\beta\lambda_A$, being $\beta = \frac{1}{1-\pi_d}$. Of course, if $\pi_d = 0$, then the failure rate of A is unchanged, while in case $\pi_d = 1$, then $\lambda_A \to \infty$ modeling the behavior of a standard FDEP gate.

In case of a one-shot PDEP gate, the interpretation of the π_d is indeed that of a *failure probability*. Given that the trigger has occurred, then the dependent component A immediately fails with probability π_d. If A does not fail, then the trigger has no longer effect and the failure rate of A remains unchanged. Again if $\pi_d = 0$ the trigger has no influence on A, while if $\pi_d = 1$ the PDEP transforms into the FDEP gate.

[2] It should be clear that, for practical reasons, there is no need of modeling a PDEP gate with $\pi_d = 0$, since this is a degenerate case where the so-called "dependent components" are actually independent from the trigger.

[3] Another probabilistic extension to the FDEP gate is presented in [24]; however, the semantics is different from our PDEP gate, since the dependent events are forced to occur probabilistically in mutual exclusion.

Persistent PDEP corresponds to potential causes of the failure of dependent components whose physical influence persists over time; this kind of gate can model situations of component degeneration due to external causes. Consider for instance a set of components, whose function is accomplished in normal conditions under the presence of a control subsystem: the unavailability of the control subsystem increases the probability of components' failure over time.

One-shot PDEP, on the other hand, can model a situation of imperfect coverage [**?**]; suppose that a set of components depends on the availability of a given subsystem: the unavailability of such a subsystem can be "covered" by some backup mechanisms that can be activated only with a given probability. A one-shot PDEP with the subsystem event as a trigger (and with the dependent components as other inputs) can model such a situation.

3.2 The Repair Box and the Corresponding Repair Policies

A repair process can be characterized by several aspects; for instance, the target of a repair process can be a single component or a subsystem composed by a set of components. In the second case, the repair can be completed at the same time for all the components, or the time to repair may change according to the specific component under repair. When we repair a subsystem, we may be interested in repairing all of its components, or a minimal subset allowing the subsystem to be operative again. Moreover, a repair process of a single component or of a subsystem, can be activated by the failure of the same component or subsystem, or such process can be triggered by another particular event, such as the failure of a higher level subsystem. Other aspects concerning the repair can be the time to detect the failure, the number of components that can be under repair at the same time, the order of repair of the components, etc.

Defining a *repair policy* for a repair process means setting all such aspects concerning the repair. In this paper, we consider several repair policies to be associated with the RB node in the DFT formalism. The first policy, called *Component Repair* policy (CR), concerns the repair of a single component and is activated by the failure of the same component, as soon as this happens. The time to repair the component is a random variable obeying to the negative exponential distribution according to the repair rate of the component.

We represent the presence of a repair process in a DFT model, by means of a RB node. Such node graphically appears as a wrench surrounded by a square. The attributes of the RB are the associated repair policy, and eventually a repair rate. If a RB acts according to the CR policy, then the RB has to be connected by means of an arc (a double-line arc), to the basic event

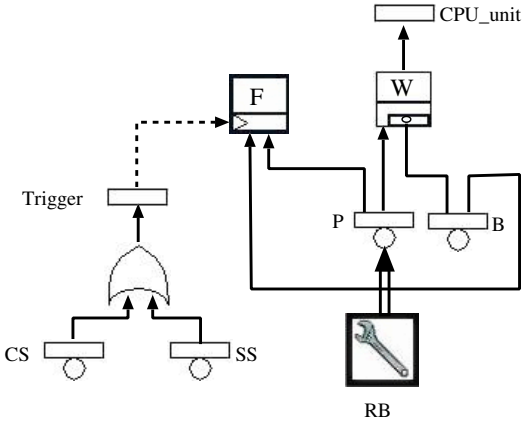

Fig. 2. A Repair Box (RB) modeling the repair of a component (*Component Repair* (CR) policy).

representing the component to be repaired (see Fig. 1(d)). In this case, a repair rate has to be defined as a parameter of the RB; this rate rules the time to repair the component. The effect of the repair is turning the component to the working state; this is modeled by the RB setting to `false` the Boolean value of the basic event corresponding to the component (meaning the component is not failed, i.e. it is working correctly). Actually a RB according to the CR policy influences indirectly the Boolean value of the events whose Boolean value depends on the value of the basic event connected to the RB. The set of the events whose Boolean value is influenced in a direct or indirect way by the RB is called coverage set (Cov) [6] of the RB.

An example of RB according to the CR policy is shown in Fig. 2; this figures depicts a portion of the larger DFT model in Fig. 9; this portion represents a subsystem of the *Cardiac Assist System* (CAS) [4] described in Sec. 6. In Fig. 2, we model the presence of the components P, B, CS, SS composing the subsystem *CPU_unit* (their role is explained in Sec. 6). The failure relations among these components are expressed by the Boolean and dynamic gates: P and B are the input events of a WSP gate; this means that P is the main component and in case of failure, it can be replaced by the warm spare component B. The FDEP gate forces the failure of both P and B if the event named *Trigger* occurs, i. e. if the component CS or SS fails. In Fig. 2, a RB is present as well, it acts according to the CR policy and is connected to the basic event P. This means that as soon as P fails, the corresponding repair process is activated. The repair process involves the component P, but it may indirectly influence the state of the subsystem containing P. Actually in Fig. 2, if both P and B are failed, then the subsystem *CPU_unit* is failed. If we repair the main component P, then also *CPU_unit* turns back to the working state. Therefore the Cov of the RB in Fig. 2 contains P and *CPU_unit*.

A repair process may concern a subsystem instead of a single component. In this case, several components are influenced by the repair process. The

repair of a subsystem can take place according to several policies. We first define the *Subsystem Global Repair* policy (SGR), where the repair of the subsystem is activated by the failure of the subsystem itself, as soon as this occurs. Moreover, all the components in the subsystem are under repair and the repair is completed at the same time for all the components; such time is ruled by a negative exponential distribution according to the subsystem repair rate.

When we have to repair a subsystem, another possible situation is the case where the time to repair changes according to the component to be recovered. This holds in the repair policy that we call *Subsystem Local Repair* policy (SLR) where the repair of a subsystem is triggered by the failure of the subsystem itself, as soon as it occurs; all its components are influenced by the repair process, but the repair of each of them may take a time different from the time to repair another component. In this case, a specific repair rate has to be set for each component in the subsystem. In this policy, we suppose that the repair of each component is always completed, even though the repair of a subset of the components may be enough to recover the subsystem.

The SLR policy can be extended in such a way that the repair of the subsystem components is interrupted as soon as the subsystem is available again, i. e. when a minimal subset of components necessary to recover the system, is repaired. In this case, some of the subsystem components may not be repaired. We call such policy SLR-min.

In a DFT model, the repair of a subsystem can be modeled by a RB connected by means of arcs to several events (see Fig. 1(e)): one arc (the double-line arc) connects the event modeling the subsystem failure to the RB; such event activates the RB and is called the *trigger repair event* [6] of the RB. Several arcs connect the RB to the set of basic events modeling the components to be repaired; such set is called *basic coverage set* (Cov_{BE}) [6] of the RB. The effect of the RB is setting to `false` the Boolean value of the elements in Cov_{BE}, after the repair process has been completed. Actually the effect of the RB is not limited to its basic coverage set, but it influences indirectly also those events whose Boolean value depends on the basic events in Cov_{BE}. The union of the Cov_{BE} and the set of the events indirectly influenced by the RB provides the coverage set of the RB (Cov) [6].

An example of subsystem repair is shown in the DFT model in Fig. 3 dealing with the same subsystem modeled in Fig. 2, but with a different repair policy: the RB in Fig. 3 represents the repair of the subsystem *CPU_unit* instead of a single component. This RB is connected to the event *CPU_unit* triggering the activation of the RB. The Cov_{BE} of the RB in Fig. 3 is composed by the basic events P, B, CS, SS; they can be identified by the arcs connecting the RB to each of them. The coverage set of the RB contains the basic events in Cov_{BE},

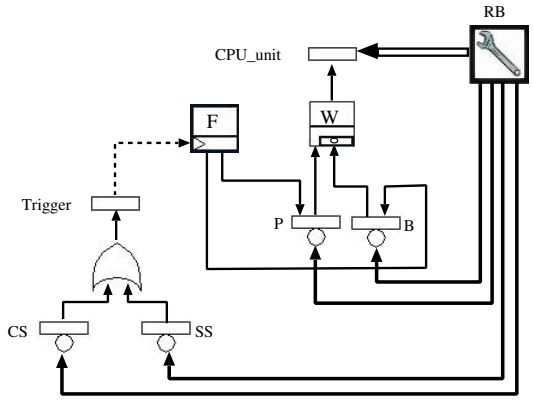

Fig. 3. A Repair Box (RB) modeling the repair of a subsystem, in case of *Subsystem Global Repair* (SGR) policy, *Subsystem Local Repair* (SLR) policy, or SLR-min policy.

together with the events named *Trigger* and *CPU_unit*. So, when the subsystem *CPU_unit* fails, its components (P, B, CS, SS) will be directly influenced by the repair process according to the repair policy associated with the RB. This may influence the Boolean value of the events *Trigger* and *CPU_unit*.

If the RB acts according to the SGR policy, the repair rate of the subsystem has to be set as an attribute of the RB: when *CPU_unit* fails, its repair is activated leading to the working state all its components (P, B, CS, SS) and consequently to the working state of *Trigger* and *CPU_unit*; this happens after a random period of time according to the repair rate of the subsystem defined in the RB.

If instead the RB in Fig. 3 acts according to the SLR or SLR-min policy, then a repair rate has to be set for each component as an attribute of the corresponding basic event. If the policy ruling the RB is SLR, when *CPU_unit* fails, the repair of each component of the subsystem starts and will be surely completed after a random period of time depending on the repair rate of the component. The subsystem *CPU_unit* may be turn available before that the repair of all the components is completed. For instance, if CS, SS, P are working, but B is still under repair, *CPU_unit* is available; the repair of B will be completed even though it is not strictly necessary to the availability of *CPU_unit*.

If the policy ruling the RB in Fig. 3 is SLR-min, the repair of each component of *CPU_unit* (P, B, CS, SS) still starts as soon as the failure of *CPU_unit* occurs, but it is interrupted as soon as *CPU_unit* is available again. For instance, let us suppose that the failure of *CPU_unit* was caused by the failure of both P and B (CS and SS are still working). Then, the repair of P and the repair of B start. If P is repaired first, then *CPU_unit* turns working again, the repair of B is interrupted and B keeps its failed state.

4 Dynamic Bayesian Networks

DBNs [7,18,17] extend the BN formalism by providing an explicit discrete temporal dimension. Each time slice contains a set of (time-indexed) random variables, some of which are typically not observable [7,18,17]. When the Markov assumption holds (and in particular when we are dealing with a first order Markov process) the future slice at time $t + \Delta$ (Δ being the so called discretization step) is conditionally independent of the past ones given the present slice at time t [12]. In this case, it is sufficient to represent two consecutive time slices called the *anterior* and the *ulterior* layer and the network is fully specified if it is provided with: *i)* - the prior probabilities for root variables at time $t = 0$; *ii)* - the intra-slice conditional dependency model, together with the corresponding Conditional Probability Tables (CPTs); *iii)* - the inter-slice conditional dependency model and CPTs (i.e. the transition model), which explicit the temporal probabilistic dependencies between variables. The above model of a DBN is usually called 2TBN (two time-slice Temporal Bayesian Network)[17,5]. A DBN (2TBN) is in *canonical form* if the anterior layer contains only variables having influence on the same variable or on another variable at the ulterior level. Given a DBN in canonical form, inter-slice edges connecting a variable in the anterior layer to the same variable in the ulterior layer are called *temporal arcs*; in other words, a temporal arcs connect variable X_i^t to variable $X_i^{t+\Delta}$ (being X_i^t the copy of variable X_i at time t). RADYBAN explicitly uses the notion of temporal arc in its representation [4]. // Concerning the analysis of a DBN, different kinds of inference algorithms are available. In particular, let X^t be a set of variables at time t and $y_{a:b}$ any stream of observations from time point a to time point b (i.e. a set of instantiated variables Y_i^j with $a \leq j \leq b$). The following tasks can be performed over a DBN:

- **Prediction**: computing $Pr(X^{t+h}|y_{1:t})$ for some horizon $h > 0$, i.e. predicting a future state taking into consideration the observation up to now; this task is called *filtering* or *monitoring* if $h = 0$.
- **Smoothing**: computing $Pr(X^{t-l}|y_{1:t})$ for some $l < t$, i.e. estimating what happened l steps in the past given all the evidence (observations) up to now.

Different algorithms, either exact or approximate can be exploited in order to implement the above tasks. In the RADYBAN tool, the user can select either the filtering/prediction or the smoothing task, and for each given task she/he may choose between using a *Junction Tree* (JT) inference [11,17] or the *Boyen-Koller* (BK) algorithm [5], a parameterized inference algorithm that, depending on the parameters provided (disjoint sets of variables called *clusters*), may

[4] A commercial tool using the same notion in modeling DBNs is BayesiaLab (`www.bayesia.com`).

return exact as well as approximate results. Such algorithms have been implemented by resorting to Intel PNL (Probabilistic Networks Library), a set of open-source C++ libraries, (`http://www.intel.com/research/mrl/pnl`), to which we have provided some minor adjustments.

5 Masking DBNs: the RADYBAN Tool and the Compilation Process

Modeling the failure modes of a system as a DBN might be complicated for the standard user of a reliability analysis tool. Indeed, very often high-level formalism are needed in order to exploit powerful analysis frameworks like the one provided by DBN. The main philosophy of the RADYBAN tool is that, in the reliability field, drawing the DFT model and generating automatically the corresponding DBN, is definitely more practical for the modeler. Using such a tool, the user can build its own DFT model and then automatically compile the model into the corresponding DBN, on which both predictive and diagnostic inference can then be drawn[5]. In this way, the DFT becomes a high level formalism allowing the reliability engineer to express in a straightforward way the relations between the components of the system, whose modeling in terms of DBN primitives would be less comfortable. On the other hand, since the analysis framework is guaranteed by the more powerful DBN formalism, we can exploit the DFT to DBN conversion to make available significant modeling extension at the user modeling level.

In previous works we have shown how to compile standard (D)FT models into (D)BNs [1,15,16]; here we will provide some details concerning the compilation process of the extensions we provided to the DFT model (i.e. PDEP and RB with different policies). Let us consider again the simple subsystem of Fig. 2, leaving out for the moment the RepairBox RB and by considering gate F as a standard FDEP gate. The structure of the corresponding DBN (in canonical form) is shown in Fig. 4. The nodes in the ulterior layer are marked with a # sign following the node's name, thus if A is a node in the anterior layer, $A\#$ is its temporal copy at the ulterior layer. Temporal arcs are drawn as thicker lines (and connect copies of the same nodes between the anterior and ulterior layer). In the example, nodes $CS\#$ (modeling component CS at the ulterior level) and $SS\#$ (modeling component SS at the ulterior layer) only depend on their "historical" copy at the anterior layer: in particular, if we assume an exponential failure rate λ_{CS} for CS and if we model the failure of a component as the Boolean value `true` of the corresponding node in the DBN, then we compute the conditional probability $Pr[CS\#|\overline{CS}] = 1 - e^{-\lambda_{CS}\Delta}$, where Δ

[5] RADYBAN allows the user to access the DBN generated by the compilation process and to edit it, as well as to directly build a DBN model, if the user feels comfortable with such a formalism.

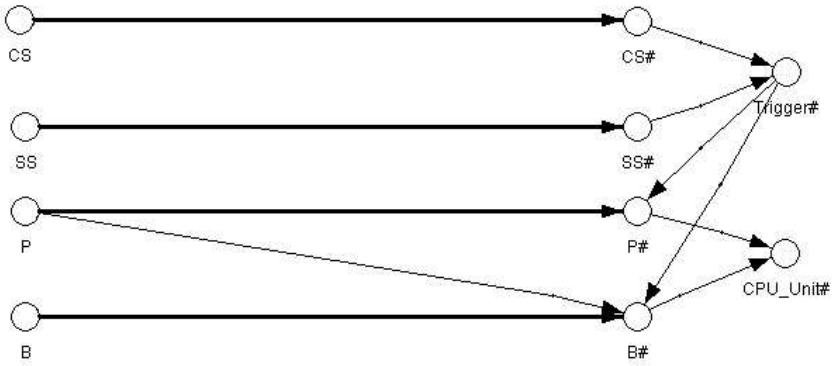

Fig. 4. A DBN for the CPU Unit subsystem.

is the *discretization step* of the DBN, i.e. the amount of time separating the anterior layer from the ulterior layer[6]. Of course, if repair is not modeled, a failure on a component persists over time, thus $Pr[CS\#|CS] = 1$. Similarly for component SS. The $Trigger$ node is simply the logical OR of CS and SS. The temporal evolution of component P depends on both its failure rate (temporal arc from node P to node $P\#$) and on the Boolean value of the trigger (intra-slice edge from $Trigger\#$ to $P\#$): in particular, when gate F is assumed to be an FDEP, then the CPT on node $P\#$ will set the boolean value `true` with probability 1 when node $Trigger\#$ is `true`. Finally, component B acts as a spare component of P, thus its temporal evolution depends on its failure rate (temporal arc from node B to node $B\#$), on the value of the trigger (similarly to P) and on the value of node P modeling the primary component of gate W (inter-slice edge from P to $B\#$). In particular, when component B is operating in place of component P (i.e. node B is `false` and node P is `true`), then the CPT entry of $B\#$ is determined by the trigger value and by the failure rate λ_B (i.e. $Pr[B\#|\overline{B}, P, Trigger\#] = 1$ and $Pr[B\#|\overline{B}, P, \overline{Trigger\#}] = 1 - e^{-\lambda_B \Delta}$), otherwise it is determined by the trigger value and the discounted (by the dormancy factor α_B) failure rate $\alpha_B \lambda_B$ (i.e. $Pr[B\#|\overline{B}, \overline{P}, Trigger\#] = 1$ and $Pr[B\#|\overline{B}, \overline{P}, \overline{Trigger\#}] = 1 - e^{-\alpha_B \lambda_B \Delta}$).

The considered subsystem is modeled through a standard DFT and the details of such a compilation into a DBN have been deeply discussed elsewhere ([1,15,16]). In the present paper, we will detail how the PDEP and RB extensions can be introduced into the compilation process. First of all, suppose that gate F is actually a PDEP gate. If F is a *persistent* PDEP, then the structure of the resulting DBN is exactly that of Fig. 4. Indeed, the only difference stands in the CPT of nodes $P\#$ and $B\#$. If gate F has an associated PDEP parameter π_F, then the CPT entries corresponding to the `true` value of dependent

[6] The choice of a suitable Δ should be performed at the analysis level, by taking into account the order of magnitude of the model rates; in particular, the determination of Δ should be guided by the fastest changing events, in such a way of being able to capture their relevant consequences.

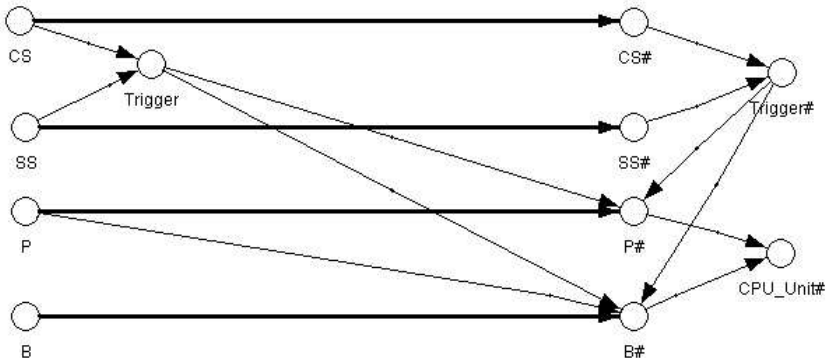

Fig. 5. A DBN for the CPU Unit subsystem with one-shot PDEP.

nodes and conditioned by a `true` value of the trigger, will be set according to a modified failure rate $\beta_F\lambda$ being $\beta_F = \frac{1}{1-\pi_F}$ and λ the failure rate of the component. For example, the conditional probability $Pr[B\#|\overline{B}, \overline{P}, Trigger\#] = 1 - e^{-\beta_F \alpha_B \lambda_B \Delta}$, while $Pr[B\#|\overline{B}, P, Trigger\#] = 1 - e^{-\beta_F \lambda_B \Delta}$. The other CPT entries are unchanged with respect to the FDEP case.

In case F is a *one-shot* PDEP, then also the DBN structure changes with respect to the FDEP case, resulting in the net of Fig. 5. In this case, the dependent components P and B must also depend on the trigger status at the anterior layer (inter-slice edges from node $Trigger$ to $P\#$ and $B\#$ nodes), since only the transition of the trigger from `false` to `true` may set, with probability π_F, their status to the `true` value. For instance $Pr[P\#|\overline{P}, \overline{Trigger}, Trigger\#] = \pi_F$, while $Pr[P\#|\overline{P}, Trigger, Trigger\#] = 1 - e^{-\lambda_P \Delta}$ since this models the situation where the trigger has been activated, but has failed in forcing the fault on P. Similar entries are determined on node $B\#$.

Fig. 6 reports an analysis of the CPU_unit failure probability (i.e. the subsystem unreliability), depending on the different functional dependencies that may be modeled by gate F. The analysis assumes exponential failure rates $\lambda_P = \lambda_B = 0.5E-3$ for components P and B, $\lambda_{CS} = \lambda_{SS} = 0.2E-3$ for components CS and SS, a dormancy factor $\alpha_B = 0.5$ for the spare component B, and a PDEP parameter $\pi_F = 0.5$ for gate F (resulting in $\beta_F = 2$, doubling the failure rates of the dependent components in case of persistent PDEP). Notice that, a one-shot probability of failure of 0.5 has a greater impact on the subsystem unreliability than doubling the failure rates of the dependent components [7].

Consider now the introduction of repair boxes in the example subsystem. First suppose that component P is reparable following a CR policy, as shown in

[7] Of course, given the order of magnitude of the failure rates in the example, simply doubling the failure rate is not increasing very much the probability of failure of a dependent component at the next time instant.

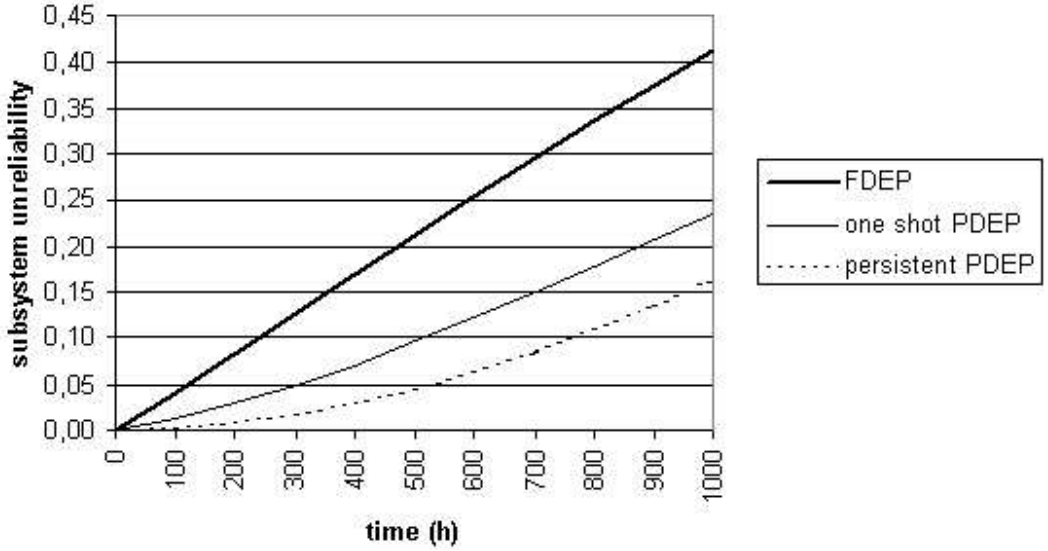

Fig. 6. The subsystem failure probability according to the type of functional dependency.

Fig. 2, with an exponential repair rate μ_P. For the sake of simplicity, let us suppose that gate F is an FDEP. The resulting DBN structure is that presented in Fig. 4. A CR policy can indeed be captured at the CPT level, by substituting the fault persistence assumption (i.e. $Pr[P\#|P,\overline{Trigger\#}] = 1$), with the corresponding repair probability. In the example, the above CPT entry transforms into $Pr[P\#|P,\overline{Trigger\#}] = e^{-\mu_P \Delta}$. Notice that, having a repair policy only on the P component does not allow to restore the CPU_unit subsystem in case of trigger occurrence, since once P is repaired, the semantics of the FDEP gate assumes the immediate fault of P again (i.e. $Pr[P\#|\overline{P},Trigger\#] = 1$).

Let us suppose now to have a repair policy defined at the subsystem level as shown in Fig. 3. Depending on the particular policy adopted, different DBNs can be generated (see Fig. 7).
The main idea is to directly model the repair policy through suitable nodes representing the stochastic evolution of the different processes composing the particular policy. In case a SGR policy is employed, the resulting net structure is presented in Fig. 7(a). Repair box nodes RB and $RB\#$ are intended to represent (at the anterior and ulterior layer respectively) the "global" repair process modeled by the RepairBox RB of Fig. 3. Such nodes are Boolean nodes where, in case of a SGR policy, the value `true` represents a completed repair process. This means that: *(a.)* if the repair is triggered, then the node evolves following the corresponding (global) repair rate μ_{RB}; *(b.)* when the node becomes `true`, the components under repair are immediately set to be functional again (i.e. the corresponding node is reset to `false`). Table 1 shows the CPT for node $RB\# =$ `true`; notice that, once $RB\#$ is set to `true`, all the subsystem components are repaired, so the CPU_unit is reset to `false`; this means that the conditioning event of the last entry is impossible (thus any

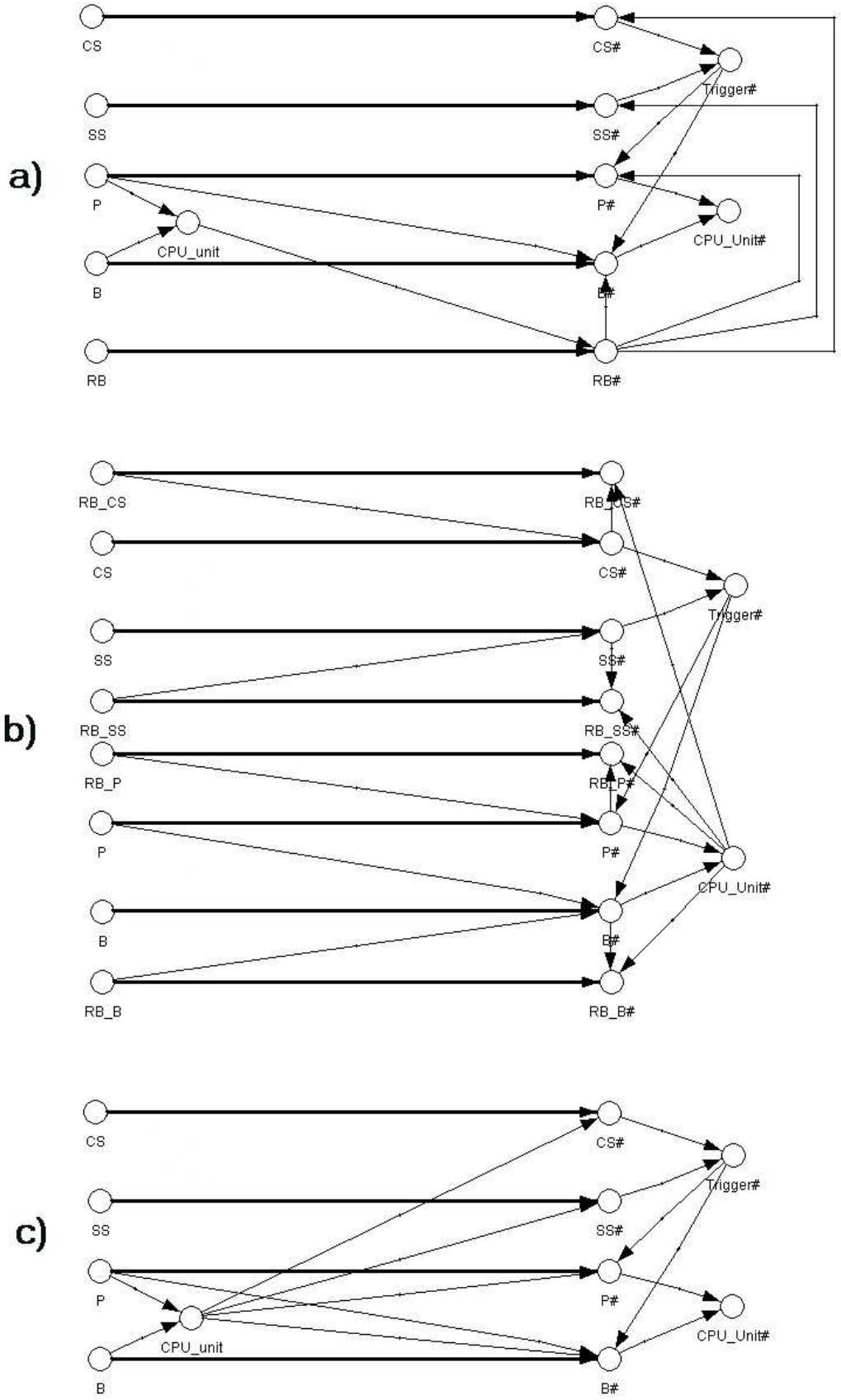

Fig. 7. DBN structure for SGR, SLR and SLR-min policies.

$Pr(RB\#)$	RB	CPU_unit
0	`false`	`false`
$e^{-\mu_{RB}\Delta}$	`false`	`true`
0	`true`	`false`
any value	`true`	`true`

Table 1
CPT for node $RB\#$ =`true` under SGR policy.

value between 0 and 1 can be used there). In order to clarify the influence of the repair box node, Table 2 reports the CPT for node $CS\#$ =`true` (similar CPTs are derived for nodes corresponding to the other components under repair).

$Pr(CS\#)$	RB	CS
$1 - e^{-\lambda_{CS}\Delta}$	`false`	`false`
1	`false`	`true`
0	`true`	`false`
0	`true`	`true`

Table 2
CPT for node $CS\#$ =`true` under SGR policy.

The second kind of repair policy we introduced is the SLR policy, concerning the independent repair of a set of subsystem components, triggered by the failure event of the subsystem. If a SLR policy is adopted in the DFT of Fig. 3, the resulting DBN is shown in Fig. 7(b). Repair box nodes are separately introduced for each repairable component; however, in this case it is convenient to have a different interpretation of the truth value of such nodes: a repair box node RB assumes value `true` if and only if the corresponding repair process is activated (and `false` otherwise); this differs from the case of SGR policy, when the `true` value means that the repair is "completed". Given such an interpretation we can set a dependency between each repairable components and its repair process (the inter-slice edges between repair box nodes and basic event nodes in Fig. 7(b)). The CPT of the nodes corresponding to repairable components will then be set by considering both the failure rate (if the repair is not active) and the repair rate (if, on the contrary, the repair is active). As an example, Table 3 reports the CPT for node $CS\#$=`true` in Fig. 7(b), under a SLR policy with repair rate μ_{CS} and failure rate λ_{CS} for component CS. Of course, if the component is functional, its failure is determined by its failure rate (first two entries of Table 3), if the component is faulty and the repair is not active, it will persists as faulty (third entry), while if it is faulty with an active repair process, then it will be probabilistically repaired following the

$Pr(CS\#)$	CS	RB_CS
$1 - e^{-\lambda_{CS}\Delta}$	`false`	`false`
$1 - e^{-\lambda_{CS}\Delta}$	`false`	`true`
1	`true`	`false`
$e^{-\mu_{CS}\Delta}$	`true`	`true`

Table 3
CPT for basic event node $CS\#$=`true` under SLR policy.

corresponding repair rate (fourth entry).

Repair box nodes are, on the other hand, deterministic nodes whose value is determined by: their previous status, the component under repair and the trigger repair event. As an example, Table 4 reports the CPT for node $RB\_CS\#$=`true`. The repair box node is set to `true` when the trigger repair

$Pr(RB\_CS\#)$	RB_CS	CS	$CPU\_unit\#$
0	`false`	`false`	`false`
1	`false`	`false`	`true`
0	`false`	`true`	`false`
1	`false`	`true`	`true`
0	`true`	`false`	`false`
1	`true`	`false`	`true`
1	`true`	`true`	`false`
1	`true`	`true`	`true`

Table 4
CPT for node $RB\_CS\#$=`true` under SLR policy.

event ($CPU\_unit\#$) is `true`; it is also kept to value `true` in case the repair is active, the component is under repair and the trigger repair event is reset to `false` by the repair of another subsystem component (seventh entry of the CPT). It is set to `false` otherwise; in particular, the fifth entry of the CPT models the situation when the repair of the component is terminated.

The DBN in Fig. 7(c) models the net resulting from the compilation of the DFT of Fig. 3 in case a SLR-min policy is adopted. In this particular case, the net structure can be simplified, without explicitly using repair box nodes, and by capturing (as in the CR policy) the repair processes at the CPT level. The only dependencies that must be taken into account are in fact, those induced by the trigger repair event on the repairable components. Table 5 reports the CPT for node $CS\#$=`true` under a SLR-min policy, with repair rate μ_{CS} and failure rate λ_{CS} for component CS. Since the component is assumed to persists

$Pr(CS\#)$	CS	CPU_unit
$1 - e^{-\lambda_{CS}\Delta}$	`false`	`false`
$1 - e^{-\lambda_{CS}\Delta}$	`false`	`true`
1	`true`	`false`
$e^{-\mu_{CS}\Delta}$	`true`	`true`

Table 5
CPT for node $CS\#$ under SLR-min policy.

as faulty in case the trigger repair event is not active (i.e. third entry of the CPT in Table 5), then if the trigger repair event is reset to `false` by the repair of another component, the repair process of CS is stopped. Of course, the repair process is started (and kept active) as long as both the trigger repair event and the component are faulty (fourth entry). Notice that the CPU_unit in Table 5 plays the same role of the repair box node RB_CS in Table 3.

Fig. 8 reports the analysis of the example subsystem, under different repair policies, using a repair rate $\mu = 0.1$ for every local and global repair process, as well as the same failure rates used for the analysis of Fig. 6. The analysis for the CR policy involves the repair of component P (Fig. 2). As we can see, this does not increase the subsystem reliability in a significant way, with respect to the unrepairable case; on the contrary, significant improvements can be obtained by using system repair policies like those of Fig. 3.

6 A case-study example

In this section, we aim at showing the general capabilities of our approach, by means of an extensive case study, inspired by a real-world system illustrated in [4] and representing a *Cardiac Assist System* (CAS). It is composed of three different modules named the CPU, the Motor and the Pump units. The failure of either one of the above modules causes the whole system failure. The CPU unit subsystem has already been introduced in section 3.2. Failure rates are supposed to be exponentially distributed and given as `fault/hour`. The CPU unit consists of two distinct CPUs: a primary CPU P (with exponential failure rate $\lambda_P = 0.5E-3$) and a backup warm spare CPU B (with dormancy factor $\alpha_B = 0.5$ and $\lambda_B = 0.5E-3$). Both CPUs are functionally dependent on two other components: a cross switch CS (with $\lambda_{CS} = 0.2E-3$) and a system supervision SS (with $\lambda_{SS} = 0.2E-3$); the failure of either the above components will trigger the definite failure of the CPUs (and so the definite failure of the whole CPU unit). However, both CPUs are considered as repairable, under a CR policy, with an exponential repair rate $\mu_{CPU} = 0.1$ (corresponding to a mean time to repair $MTTR = 10$ hours).

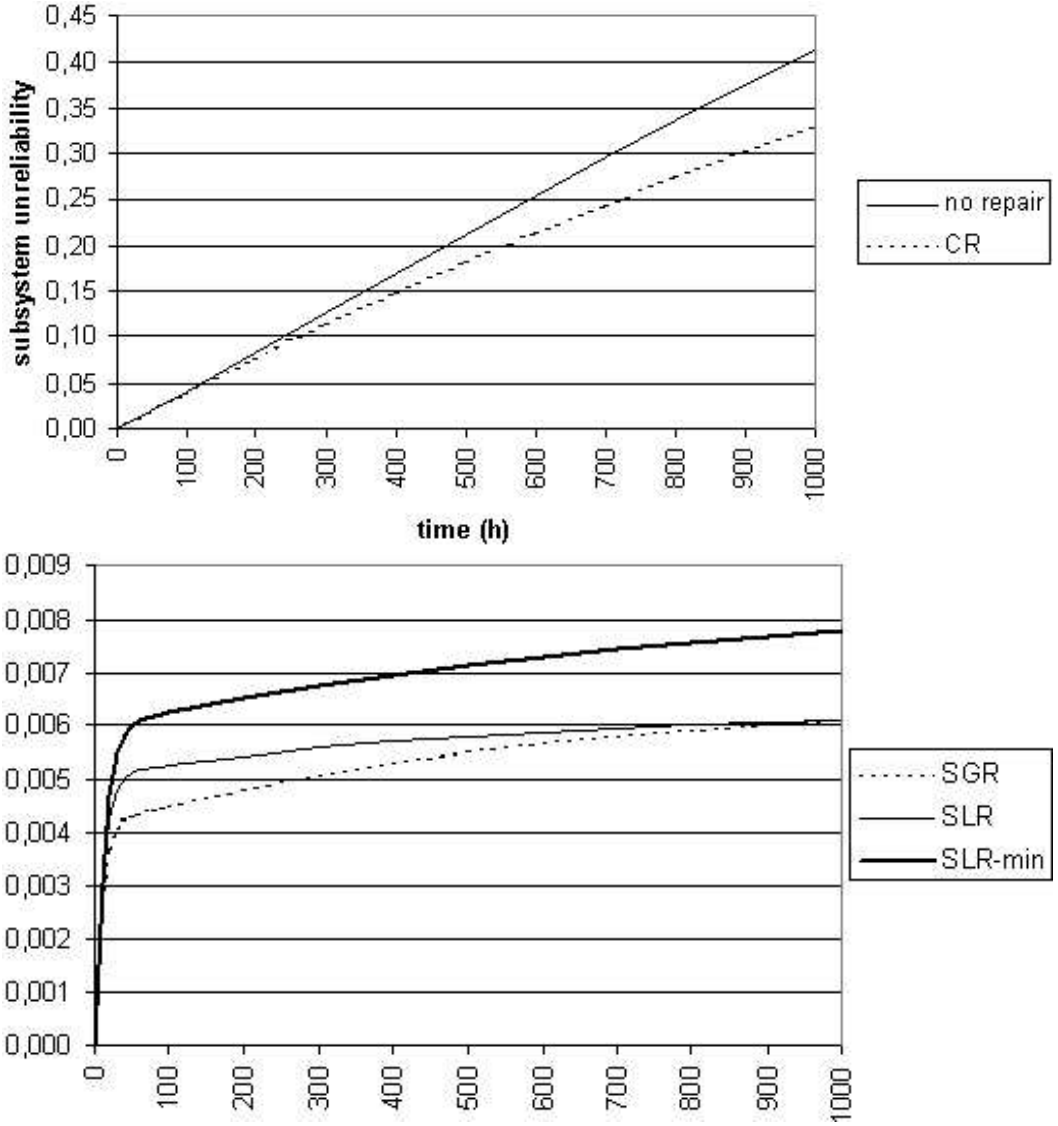

Fig. 8. The subsystem failure probability according to the type of functional dependency.

The Motor unit consists of two motors: a primary motor MA ($\lambda_{MA} = 1E-3$) and a cold spare motor MB ($\lambda_{MB} = 1E-3$). The spare motor turns into operation when the primary fails, because of a motor switching component MS ($\lambda_{MS} = 0.01E-3$); this means that if MS fails before the necessity of the switch (i.e. before the failure of MA), then the spare cannot become operational and the whole Motor unit fails. Finally, the Pump unit is composed by three pumps: two primary pumps PA and PB (with $\lambda_{PA} = \lambda_{PB} = 1E-3$) running in parallel and a cold spare pump PS ($\lambda_{PS} = 1E-3$). The Pump unit is operational if at least one of the pump is operational.
Fig. 9 shows the DFT for the CAS system, drawn by means of RADYBAN. The structure of the DBN, in canonical form, corresponding to the DFT in Fig. 9, is shown in Fig. 10 and is automatically generated given the DFT model, by

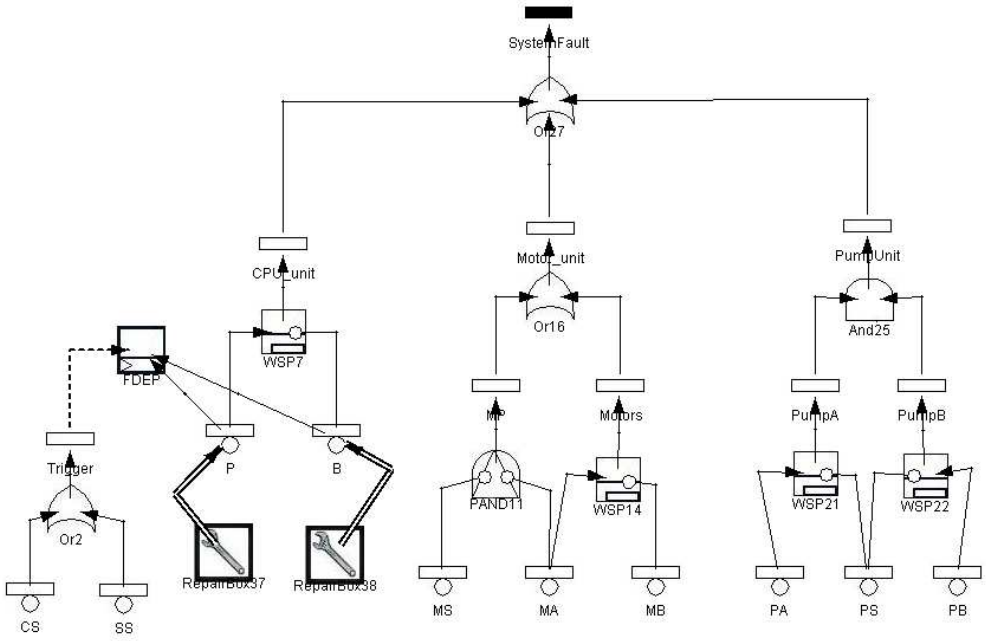

Fig. 9. The DFT model of CAS.

using our tool, together with the corresponding CPT quantification.

After the conversion of the DFT in the DBN, we can perform the analysis of the latter by means of our tool. First of all, let us consider the situation when the CPUs are not repairable (i.e. when the two repair boxes of the DFT in fig. 9 are not present or, alternatively, when the associated repair rates are 0).

Tab. 6 shows the unreliability of the system versus the mission time varying between 0 and 1000 hours (with time step 100); the table compares the measures computed by RADYBAN (with discretization steps $k = 1$ hour and $k = 0.1$ hours) with those obtained by GALILEO, a software tool for DFT analysis without explicit repair modeling features and based on CTMC analysis [9].

We can notice that the results are essentially in agreement, with a small difference due to the different nature of the underlying models of the tools (namely a discrete time model for RADYBAN and a continuous time model for GALILEO). Such a difference becomes typically smaller if a smaller discretization step is used. To perform the unreliability computation using RADYBAN, we just adopted a filtering task by querying node `System` (i.e. the Top Event) without providing any observation stream; in other words we performed standard prediction.

In case we want to analyze the impact of CPU repair, we can compute the system unreliability by taking into consideration the repair boxes of fig. 9

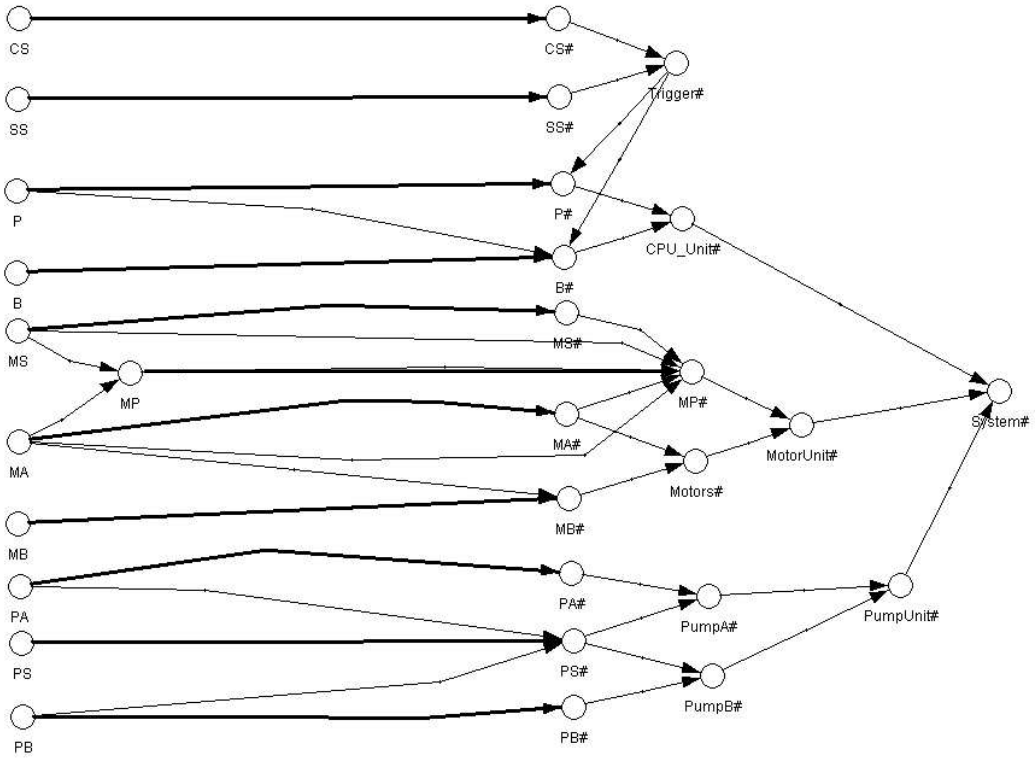

Fig. 10. The DBN corresponding to the DFT in Fig. 9.

time	*RADyBaN (k=1)*	*RADyBaN (k=0.1)*	*Galileo*
$100h$	0.045978	0.046026	0.0460314
$200h$	0.103124	0.103214	0.103222
$300h$	0.169204	0.169327	0.169336
$400h$	0.241328	0.241474	0.241483
$500h$	0.316482	0.316645	0.316651
$600h$	0.391893	0.392060	0.392066
$700h$	0.465241	0.465408	0.465411
$800h$	0.534745	0.534908	0.534908
$900h$	0.599169	0.599322	0.59932
$1000h$	0.657763	0.657908	0.6579

Table 6
The unreliability results obtained by RADYBAN and by GALILEO if repair is not available.

with the given repair rates $\mu_{RB\_P} = \mu_{RB\_B} = 0.1$. Since GALILEO cannot explicit model the repair of system components, we resort to a comparison with another tool called *DRPFTproc* [2], based on modularization [10], conversion to Stochastic Petri Nets of dynamic gates and having repair modeling features [6]. The results of this comparison (when a discretization step $k = 1$ has been used for RADYBAN) are shown in table 7 (second and fourth columns).

time	*RADyBaN*		*DRPFTproc*	
	CPU repair	CPU+Trigger repair	CPU repair	CPU+Trigger repair
$100h$	0.044283796102	0.011243030429	0.0443301588	0.0112820476
$200h$	0.096916869283	0.027566317469	0.0951982881	0.0276517226
$300h$	0.156659856439	0.054836865515	0.155093539	0.0549629270
$400h$	0.221550568938	0.091957211494	0.220137459	0.0921166438
$500h$	0.289382189512	0.137252241373	0.288119742	0.137437204
$600h$	0.358023554087	0.188778832555	0.356905021	0.188981668
$700h$	0.425606846809	0.244557544589	0.424624354	0.244770740
$800h$	0.490624904633	0.302729338408	0.489768367	0.302945892
$900h$	0.551952958107	0.361649900675	0.551211316	0.361864672
$1000h$	0.608829379082	0.419938921928	0.608191065	0.420148205

Table 7
The unreliability results obtained by RADYBAN and by *DRPFTproc* if different repair possibilities are available.

Again we can notice the agreement between the different tools (despite the very different analysis techniques). Moreover, we can also notice that, adopting a repair policy only on the CPUs, does not increase very much the reliability of the system with respect to the unrepairable case (see Tab. 6 for a comparison with the second and fourth column of Tab. 7). This is explained by the fact that both CPUs are functionally dependent from the `Trigger` event of the DFT; this means that, once the trigger is active, then a repair on the CPUs has no effect (since the trigger is going to induce a CPU failure again, because of the FDEP gate of Fig. 2). Of course, if we want to limit the effect of the trigger, then we can impose a repair policy on the trigger components as well. Let us suppose that also components CS and SS are subject to a CR policy with rate $\mu_{CS} = \mu_{SS} = 0.1$: this simply corresponds to add two repair boxes (with such rates) on the DFT of Fig. 9, connected with CS and SS respectively. The results concerning the unreliability of the system if also such repair boxes are available are again reported in Tab. 7 (third and fifth columns). Not surprisingly, the unreliability of the system is reduced (of about 36% on average on the time slices considered), with respect to the previous case.

Notice that using four repair boxes with CR policy on the components of the *CPU_unit* subsystem is not equivalent to have a single RB with SLR policy on the *CPU_unit* event and with $Cov_{BE} = \{P, B, CS, SS\}$ (see Fig. 3). In the latter, no repair process is started until the whole *CPU_unit* subsystem is failed, while in the example presented here the local repair processes are started as soon as the corresponding component fails [8]

Moreover, DBNs (and RADYBAN in particular) offer additional analysis capabilities with respect to standard (D)FT-based tool lke GALILEO and *DRPFT-proc*. In particular, the possibility of exploiting explicit observations on monitorable parameters, as well as the possibility of performing smoothing inference; this allows one to rebuild the past history of the system, given a stream of observations. Consider a situation in which the CAS system (with CPU and Trigger repair available) was observed operational at time $t = 100h$, failed at time $t = 300h$ and then operational again at time $t = 500h$. First of all, this stream of observations is consistent with the model, since a repair on the CPU unit can actually repair the whole system; notice that, if this stream of observation would be provided to the model without repair, an exception would be raised by the DBN inference algorithms, pointing out an inconsistency with respect to the model provided. Let σ be the observation stream about the system behavior introduced above; a first kind of analysis we can perform consists in querying the three modules potential causes of the fault (CPU, Motor and Pump units), asking for their posterior probabilities given σ.

The results concerning the marginal probability of each subsystem over time, from a filtering procedure are reported in Fig. 11; the joint probability of each possible configuration of the queried subsystems is reported on Table 8. We can notice that, when the system is observed to be operational, all the three modules must be operational, while when the system is observed to be faulty ($t = 300h$), then there is a high probability that such fault is due to the Motor unit. In particular, by looking at the joint probability entries, the most probable diagnosis is that the Motor unit is the only module to be faulty. The CPU unit module is the less probable cause of the fault, since it is repairable. However, since the system is then observed to be operational again, the most probable diagnosis at time $t = 300h$ is definitely wrong, since it should be clear that the observed system behavior can be explained only by a failure of the CPU unit in the interval $(100, 300]$, with a consequent repair of such a unit (which is the only repairable unit) in the interval $(300, 500]$. This cannot be captured by the filtering procedure (which is just a system monitoring over time); on the other hand, the explanation can be confirmed by performing the

[8] If an SLR policy was adopted, the upper part of the DBN in Fig 10 (corresponding to the subnet structure of Fig 4) would be substituted with the subnet shown in Fig 7(b).

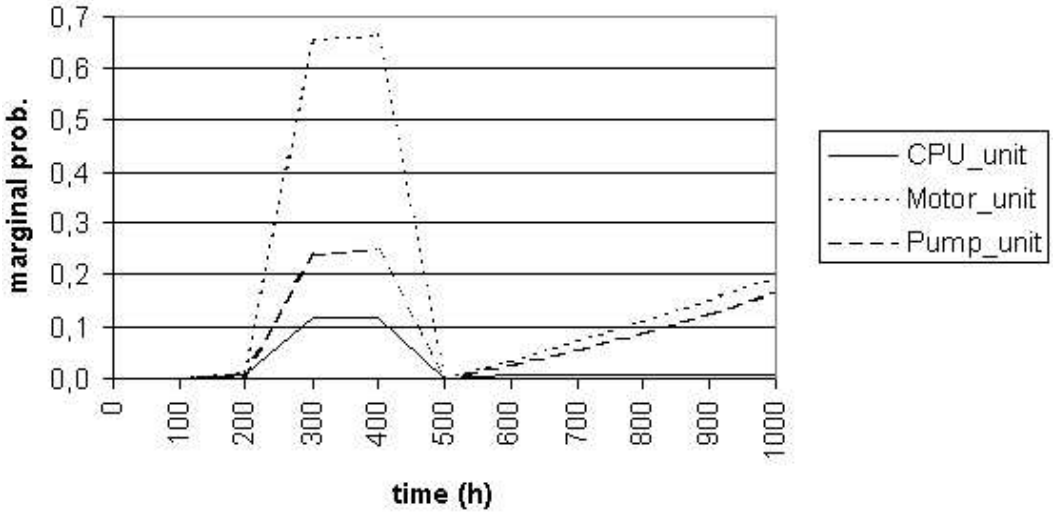

Fig. 11. The marginal probabilities in case of filtering. Each curve indicates the failure probability of each subsystem.

same posterior probability computation above, but under a smoothing inference procedure.
The results from a smoothing inference are reported in Fig. 12 (marginal probabilities of the subsystems over time) and in Table 9 (joint probabilities of the configuration of the subsystems).

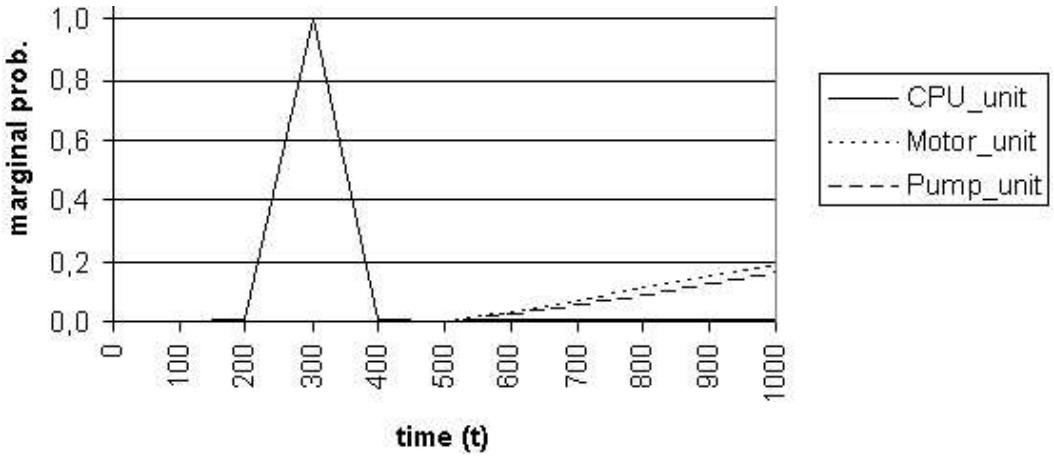

Fig. 12. The marginal probabilities in case of smoothing. Each curve indicates the failure probability of each subsystem.

Differently from filtering inference, the above results show that before time $t = 500h$ Motor and Pump units cannot be faulty, since at $t = 500h$ the system must be operational; moreover, by considering that the system behavior has changed from faulty to operational in the interval $[300, 500]$, then it is certain that in such interval the CPU unit has been repaired. In particular, since repair is significantly faster than failure (because of the given repair and

time	0, 0, 0	0, 0, 1	0, 1, 0	0, 1, 1
100 h	**1.000000**	0.000000	0.000000	0.000000
200 h	**0.977576**	0.003501	0.012862	0.000046
300 h	0.000000	0.228510	**0.643095**	0.007708
400 h	0.110162	0.224175	**0.637081**	0.022560
500 h	**1.000000**	0.000000	0.000000	0.000000
600 h	**0.934621**	0.024475	0.033999	0.000890
700 h	**0.870357**	0.051434	0.068166	0.004028
800 h	**0.803337**	0.079515	0.101124	0.010009
900 h	**0.735453**	0.107478	0.131794	0.019260
1000 h	**0.668297**	0.134277	0.159387	0.032024

time	1, 0, 0	1, 0, 1	1, 1, 0	1, 1, 1
100 h	0.000000	0.000000	0.000000	0.000000
200 h	0.005916	0.000021	0.000078	0.000000
300 h	0.115366	0.001383	0.003891	0.000047
400 h	0.000673	0.001357	0.003855	0.000137
500 h	0.000000	0.000000	0.000000	0.000000
600 h	0.005655	0.000148	0.000206	0.000006
700 h	0.005267	0.000311	0.000413	0.000024
800 h	0.004861	0.000481	0.000612	0.000061
900 h	0.004450	0.000650	0.000798	0.000117
1000 h	0.004044	0.000813	0.000964	0.000194

Table 8
The joint probabilities in case of filtering. The label of each column indicates the state (0 means working, 1 means failed) of the subsystems *CPU_unit*, *Motor_unit*, *Pump_unit* respectively. Bold entries correspond to most probable configurations.

failure rates), smoothing can suggest that such a repair has been probably performed before $t = 400h$ (since the probability of the CPU unit being operational at time $t = 400h$ is close to 1). ¿From time $t = 500h$, no observation is available, so both filtering and smoothing will produce standard prediction results.
In the examples reported in this section exact algorithms (based on the calculation of the junction tree) were adopted, both for filtering and for smoothing procedures.

time	0, 0, 0	0, 0, 1	0, 1, 0	0, 1, 1
100 h	**1.000000**	0.000000	0.000000	0.000000
200 h	**0.993925**	0.000000	0.000000	0.000000
300 h	0.000000	0.000000	0.000000	0.000000
400 h	**0.993925**	0.000000	0.000000	0.000000
500 h	**1.000000**	0.000000	0.000000	0.000000
600 h	**0.934620**	0.024475	0.034000	0.000890
700 h	**0.870357**	0.051434	0.068166	0.004028
800 h	**0.803337**	0.079515	0.101124	0.010009
900 h	**0.735453**	0.107478	0.131794	0.019260
1000 h	**0.668297**	0.134277	0.159387	0.032024

time	1, 0, 0	1, 0, 1	1, 1, 0	1, 1, 1
100 h	0.000000	0.000000	0.000000	0.000000
200 h	0.006075	0.000000	0.000000	0.000000
300 h	**1.000000**	0.000000	0.000000	0.000000
400 h	0.006075	0.000000	0.000000	0.000000
500 h	0.000000	0.000000	0.000000	0.000000
600 h	0.005655	0.000148	0.000206	0.000006
700 h	0.005267	0.000311	0.000413	0.000024
800 h	0.004861	0.000481	0.000612	0.000061
900 h	0.004450	0.000650	0.000798	0.000117
1000 h	0.004044	0.000813	0.000964	0.000194

Table 9
The joint probabilities in case of smoothing. The label of each column indicates the state (0 means working, 1 means failed) of the subsystems *CPU_unit*, *Motor_unit*, *Pump_unit* respectively. Bold entries correspond to most probable configurations.

7 Conclusions

In this paper, we have described an approach, implemented in the RADYBAN tool, allowing reliability engineers to work at two different modeling levels, while still having available the full inference power of the DBN formalism. The user who is not familiar with DBNs can work at the modeling level with DFTs, obtain an automatic transparent conversion into the corresponding DBN, and ask for reliability measures by means of DBN inference algorithms. We have

also described some major extensions that can be provided to the standard DFT model (namely probabilistic dependency gates and repair boxes), that allows reliability engineers to augment their modeling primitives, while still exploiting the full power of the underlying DBN model. In particular, modeling repair processes is an important aspect of reliability analysis. However, the standard way of addressing related issues in FT analysis is typically through an indirect modeling. In fact, it is well known that in standard *FT* the basic components (assumed as independent) can be instantiated with a value corresponding to their *unavailability*[9]; this value implicitly accounts for the repair capabilities. The new gate REPAIR BOX in our formalism (and its compilation to a DBN model) has a double advantage: *i)* - the repair can be explicitly included in the model and quantified according to several repair policies; *ii)* - in the analysis phase, when a stream of observations is assumed for the BN inference, the components with a repair box may be observed either working or failed (see Section 6).

Concerning the resulting DBN models, we would like to spend a few words about the discretization step; a DBN is a discrete-time model and this differs from the underlying model usually adopted in the reliability analysis of a DFT which is usually a CTMC. As a matter of fact, the two models are not exactly equivalent, since in a CTMC transitions occur in a continuous fashion. There is a trade-off between the approximation provided by discretization and the computational effort needed for the analysis: smaller is the discretization step, more accurate are the results obtained (and closer to the continuous case computation, see table 6 in section 6), but greater is the time horizon required for the analysis (and thus the computation time). In fact, if failure rates are given as $fault/hour$ and we set a mission time of T hours, a discretization step $\Delta = 1h$ will require analysis up to step $t = T$, while a discretization step $\Delta = 10h$ will only require analysis up to step $t = T/10$ (since each step will count as 10 time units); this fact, in DBN inference, will result in a speed up of the result computation, because a smaller number of time slices have to be considered (i.e. a time slice in the latter case approximate 10 slices in the former).

We have presented some examples of the use of the approach, by considering a cardiac assist system (CAS) inspired from a case study presented in [4]. The results obtained using the automatically generated DBN models are in agreement with the ones obtained using other analysis techniques described in the literature, when such techniques are applicable to the case under study. The advantage of our approach is to address modeling features (e.g. probabilistic dependencies or repair policies) as well as analysis tasks (filtering

[9] In particular, if exponential failure and repair rates λ and μ are assumed (with $\sigma = \lambda + \mu$), then the probability of a basic event being failed at time t (i.e. its unavailability at time t) is computed as $1 - \frac{\mu}{\sigma} + \frac{\lambda}{\sigma} e^{-\sigma\, t}$ [22].

and smoothing under observation gathering) that are not easily modeled with other approaches.
We feel that this approach can be a step forward in making available formalisms based on Bayesian Nets to the reliability community, without asking reliability practitioners to renounce to their familiar constructs like those used in FTs or DFTs.

References

[1] A. Bobbio, L. Portinale, M. Minichino, E. Ciancamerla, Improving the analysis of dependable systems by mapping fault trees into bayesian networks, Reliability Engineering and System Safety 71 (2001) 249–260.

[2] A. Bobbio, D. C. Raiteri, Parametric fault-trees with dynamic gates and repair boxes, in: Proceedings Reliability and Maintainability Symposium RAMS2004, Los Angeles, USA, 2004.

[3] H. Boudali, J. Bechta-Dugan, A new bayesian network approach to solve dynamic fault trees, in: Proceedings Reliability and Maintainability Symposium RAMS2005, 2005.

[4] H. Boudali, P. Crouzen, M. Stoelinga, Dynamic Fault Tree analysis using Input/Output Markov Chains, in: Proceedings 37th IEEE/IFIP Conference on Dependable Systems and Networks (DSN'07), 2007.

[5] X. Boyen, D. Koller, Tractable inference for complex stochastic processes, in: Proceedings UAI 1988, 1998.

[6] D. Codetta-Raiteri, G. Franceschinis, M. Iacono, V. Vittorini, Repairable fault tree for the automatic evaluation of repair policies, in: Proc. Intern. Conference on Dependable Systems and Networks (DSN'04), Firenze, Italy, 2004.

[7] T. Dean, K. Kanazawa, A model for reasoning about persistence and causation, Computational Intelligence 5 (3) (1989) 142–150.

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

[9] J. B. Dugan, K. Sullivan, D. Coppit, Developing a low-cost high-quality software tool for dynamic fault-tree analysis, IEEE Transactions on Reliability 49 (1) (2000) 49–59.

[10] R. Gulati, J. Bechta-Dugan, A modular approach for analyzing static and dynamic fault trees, in: Proceedings Reliability and Maintainability Symposium RAMS1997, 1997.

[11] F. Jensen, Bayesian Networks and Decision Graphs, Springer, 2001.

[12] U. Kjaerulff, dHugin: a computational system for dynamic time-sliced bayesian networks, International Journal of Forecasting 11 (1995) 89–101.

[13] H. Langseth, L. Portinale, Bayesian networks in reliability, Reliability Engineering and System Safety 92 (1) (2007) 92–108.

[14] R. Manian, D. Coppit, K. Sullivan, J. Dugan, Bridging the gap between systems and dynamic fault tree models, in: Proceedings IEEE Annual Reliability and Maintainability Symposium, IEEE Computer Society Press, Washington, DC, 1999.

[15] S. Montani, L. Portinale, A. Bobbio, Dynamic bayesian networks for modeling advanced fault tree features in dependability analysis, in: K. Kolowrocki (ed.), Proc. European Safety and Reliability Conference 2005 (ESREL 05), Balkema Publ., Tri City, Poland, 2005.

[16] S. Montani, L. Portinale, A. Bobbio, D. Codetta-Raiteri, RADYBAN: a tool for reliability analysis of dynamic fault trees through conversion into dynamic bayesian networks, Reliability Engineering and System Safety 93 (2008) 922–932.

[17] K. Murphy, Dynamic Bayesian Networks: Representation, Inference and Learning, PhD Thesis, UC Berkley, 2002.

[18] P.Dagum, A. Galper, E. Horwitz, Dynamic network models for forecasting, in: Proc. UAI'92, 1992.

[19] L. Portinale, A. Bobbio, Bayesian networks for dependability analysis: an application to digital control reliability, in: 15-th Conference Uncertainty in Artificial Intelligence, UAI-99, 1999.

[20] K. Przytula, R. Milford, An efficient framework for the conversion of fault trees to diagnostic Bayesian network models, in: Proc. of the IEEE Aerospace Conference, Malibu, CA, 2006.

[21] J. Torres-Toledano, L. Sucar, Bayesian networks for reliability analysis of complex systems, in: Lecture Notes in Artificial Intelligence, vol. 1484, Springer Verlag, Berlin, 1998.

[22] K. Trivedi, Probability and Statistics with Reliability, Queuing and Computer Science Applications (2nd Ed.), John Wiley & Sons, 2001.

[23] P. Weber, L. Jouffe, Reliability modelling with dynamic bayesian networks, in: SafeProcess 2003, 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, Washington DC, 2003.

[24] L. Xing, H. E. Michel, Integrated modeling for wireless sensor networks reliability and security, in: Proceedings 52nd Annual Reliability and Maintainability Symposium (RAMS 2006), Newport Beach, CA, 2006.