A unified modelling and operational framework for Fault Detection, Identification and Recovery in autonomous spacecrafts *

Daniele Codetta-Raiteri[1]**, Luigi Portinale[1],
Stefano Di Nolfo[2], Andrea Guiotto[2], and Yuri Yushtein[3]

[1] DiSIT, Computer Science Institute, University of Piemonte Orientale, Italy
[2] Thales Alenia Space, Torino, Italy
[3] European Space Research and Technology Centre, Noordwijk, Netherlands

Abstract. Recent studies focused on the achievement of autonomy of exploration spacecrafts, such as Mars rovers. The traditional approach for on-board FDIR (Fault Detection, Identification and Recovery) is based on the run-time observation of the system operational status in order to detect faults, while the initiation of the corresponding recovery actions uses static pre-compiled look-up tables. This approach is a purely reactive approach, lacking of preventive recovery capabilities, and puts the spacecraft into a known safe configuration and transfers control to the ground operations. In the VeriFIM study, we developed ARPHA, an on-board FDIR reasoning engine based on probabilistic graphical models. The approach followed in ARPHA provides a unified modeling and operational framework that integrates a high level modeling formalism (Dynamic Fault Tree (DFT)), a low level modeling formalism (Dynamic Bayesian Network (DBN)) and an inference oriented formalism (Junction Tree (JT)). The off-board process of ARPHA consists of the construction of the DFT by reliability engineers, the automatic transformation into DBN, the manual enrichment of the DBN to model the features that DFT can not represent, and then the JT automatic generation. The JT is the actual on-board model undergoing analysis conditioned by sensor data and plan data. The goal is the on-board evaluation of the system current state (diagnosis) and future state (prognosis), in order to detect (in a probabilistic way) current or imminent anomalies or failures, and choosing the most suitable recovery policies taking into account their effect on the system in the near future. All of this is performed in automatic way, without the assistance of the ground control. In this paper, we present the application of this approach to a case study concerning the power supply subsystem of a Mars rover.

Keywords: fault detection, identification and recovery; autonomous systems; Dynamic Fault Tree; Dynamic Bayesian Network.

* This work has been funded by European Space Agency (ESA/ESTEC) under the VeriFIM study (grant n. TEC-SWE/09259/YY).
** Corresponding author: dcr@di.unipmn.it

1 Introduction

In autonomous spacecraft operations, both the system behavior and the environment can exhibit various degrees of uncertainty; control software must then provide the suitable and timely reaction of the system to changes in its operational environment, as well as in the operational status of the system. The operational status of the system is dependent on the internal system dependability factors (e.g. sub-system and component reliability models), on the external environment factors affecting the system reliability and safety (e.g. thermal, radiation, illumination conditions) and on system-environment interactions (e.g. stress factors, resource utilization profiles, degradation profiles, etc.). Combinations of these factors may cause mission execution anomalies, including mission degradations and system failures. To address possible system faults and failures, the system under examination must be provided with some form of health management procedures, usually relying on the *Fault Detection, Identification and Recovery* (FDIR) process. Currently employed state-of-the-art of the FDIR is based on the design-time analysis of the faults and failure scenarios (e.g. *Failure Mode Effect Analysis* (FMEA), *Fault Tree Analysis* (FTA) [1]) and run-time observation of the system operational status (health monitoring). The goal is in general to timely detect faults and to start a predefined recovery procedure (by using look-up tables), having the goal of putting the spacecraft into a known safe configuration and transfer control to the ground operations for troubleshooting and planning actual recovery.

Standard FDIR approaches have multiple shortcomings which may significantly reduce effectiveness of the adopted procedures: **1)** the system, as well as its environment, is only partially observable by monitoring procedures; this introduces uncertainty in the interpretation of observations in terms of the actual system status, which is often disregarded in choosing the possible recovery. **2)** Recovery is essentially triggered following a reactive approach, a post-factum operation, not capable of preventive measures and that cannot provide and utilise prognosis for the imminent failures.

The main source of such limits is recognized to be the fact that knowledge of the general operational capabilities of the system (that should potentially be expressed in terms of causal probabilistic relations) is not usually represented on-board, making impossible to estimate the impact of the occurred faults and failures on these capabilities. Several studies have tried to address these problems, some by restricting attention to manned systems [2] or to systems requiring heavy human intervention [3], some others by emphasizing the prognostic phase and relying on heuristics techniques to close the FDIR cycle [4].

The goal of the VeriFIM study is an innovative approach to on-board FDIR: the FDIR engine exploits an on-board probabilistic graphical model which must take into account the system architecture, the system environment, the system-environment interaction, and the dynamic evolution in presence of uncertainty and partial observability. Moreover, the on-board FDIR engine must provide the system with diagnosis (fault detection and identification) and prognosis (fault prediction) on the operational status to be taken into account for autonomous

reactive or preventive recovery actions. To this aim, inside VeriFIM, we developed the software prototype called ARPHA (*Anomaly Resolution and Prognostic Health management for Autonomy*).

Before the execution of ARPHA (on-board process), the on-board model must be prepared. Because of the several aspects to be represented by the on-board model, the modelling phase (off-board process) integrates a high level modeling formalism (*Dynamic Fault Tree* (DFT) [5]), a low level modeling formalism (*Dynamic Bayesian Network* (DBN) [6]) and an inference oriented formalism (*Junction Tree* (JT) [7]). Basic notions about these formalisms are reported in Sec. 2. The on-board model (JT) is obtained through a sequence of model conversions and model enrichment. In Sec. 3, we present a case study concerning the power supply subsystem of a Mars rover. In Sec. 4, this case study provides a running example for the off-board process (modelling phase) and the on-board process (diagnosis, prognosis and recovery of the system, conditioned by sensor data and plan data).

2 Basic notions about DFT, DBN, JT

DFT. *Fault Tree* (FT) [1] is the most diffused and popular model in Reliability analysis. A FT is a *direct acyclic graph* (DAG) representing how several combinations of *Basic Events* (BE) lead to the occurrence of a particular event called *Top Event* (TE). Each BE (component failure) has a certain probability to occur according to its failure rate. So, it is possible to compute the probability of occurrence of TE (system failure). In FT, BEs are assumed to be independent and the combinations of BEs leading to TE can only be expressed by means of Boolean gates (AND, OR). Therefore the modeling power of FT is rather limited, so several extensions have been proposed in the literature, such as DFT (Fig. 2) introducing dynamic gates representing dependencies: a dependency arises when the failure behavior of a component depends on the state of another component or subsystem. Dynamic gates represent several kinds of dependencies: functional dependencies, dependencies concerning the events order, and the presence of spare components.

DBN. *Bayesian Networks* (BN) [8] have become a widely used formalism for representing uncertain knowledge in probabilistic systems. BN are defined by a DAG in which discrete random variables are assigned to each node, together with the conditional dependence on the parent nodes. In particular, each node has associated a *Conditional Probability Table* (CPT) specifying the probability of each value of the node, conditioned by every instantiation of parent nodes. Root nodes are nodes with no parents, and marginal prior probabilities are assigned to them. In this way, it is possible to include local conditional dependencies, by directly specifying the causes that influence a given effect. This allows computing the probability distribution of any variable given the observation of the values of any subset of the other variables. DBN (Fig. 3) extend BN by providing an explicit discrete temporal dimension. A DBN is essentially the replication of a BN over two time slices, $t - \Delta$ and t, where Δ is the time discretization step.

In other words, each variable has two instances, one for each time slice. If a variable is characterized by a temporal evolution, its instance in the time slice t depends on its instance in $t - \Delta$. Moreover, it is possible to establish intra-slice dependencies involving different variables in the same time slice, or inter-slice dependencies involving different variables in different time slices. Given a set of observations up to the current time t, it is possible to compute the probability distribution of a variable at t, in the future, or in the past, conditioned by the observations.

JT. A way to efficiently compute conditioned probabilities on a (D)BN, consists of generating and analyzing the JT according to the procedures detailed in [7]. A JT is an undirected unrooted tree where each node (also called a cluster) corresponds to a set of nodes in the original (D)BN. The Boyen-Koller (BK) algorithm [9] is a parametric JT-based inference strategy: the algorithm depends on some input parameters (set of nodes) called "bk-clusters"; according to the bk-clusters provided, it can produce approximate inference results with different degrees of accuracy. In particular, if the input is a unique bk-cluster containing all the so-called "interface nodes" of the DBN, the BK algorithm performs exact inference (see [9] for the details). The main reason for implementing approximate inference is that, in case of network models which are particularly hard to solve with exact inference, a reasonable approximation can trade-off time/space complexity and quality of the results[4].

3 A case study

An example case study we have used to test ARPHA, concerns the power supply subsystem of a Mars rover, with a particular attention to the following aspects.

Solar arrays. We assume the presence of three solar arrays (SA), namely SA1, SA2, SA3. In particular, SA1 is composed by two redundant strings, while SA2 and SA3 are composed by three strings. Each SA can generate power if both the following conditions hold: 1) at least one string is not failed; 2) the combination of sun aspect angle (SAA), optical depth (OD), and local time (day or night) is suitable. In particular, OD is given by the presence or absence of shadow or storm. The total amount of generated power is proportional to the number of SAs which are actually working.

Load. The amount of load depends on the action performed by the rover.

Battery. We assume the battery to be composed by three redundant strings. The charge of the battery may be steady, decreasing or increasing according to the current levels of load and generation by SAs. The charge of the battery may be compromised by the damage of the battery occurring in two situations: all the strings are failed, or the temperature of the battery is low.

Scenarios. We are interested in four failure or anomaly scenarios. Each scenario

[4] The assumption is also that, since the networks used by ARPHA have a reasonable number of observed variables (i.e. each relevant system component is a sensored component and sensors have a high accuracy), then the approximation error is bounded by conditioning on the next set of observations during a temporal inference.

can be recovered by specific policies:
- **S1)** low (anomaly) or very low (failure) power generation while SAA is not optimal. Recovery policies: **P1)** suspension of the plan in order in order to reduce the load; **P2)** change of inclination of SA2 and SA3 in order to try to improve SAA and consequently the power generation (the tilting system cannot act on SA1).
- **S2)** low (anomaly) or very low (failure) power generation while OD is not optimal. Recovery policies: **P3)** movement of the rover into another position in order to try to avoid a shadowed area and improve OD and the power generation as a consequence; **P4)** modification of the inclination of SA2 and SA3, retraction of the drill, and suspension of the plan.
- **S3)** low (anomaly) or very low (failure) battery level while drilling. Recovery policies: **P4)** as above; **P5)** retraction of the drill, suspension of the plan.
- **S4)** low (anomaly) or very low (failure) battery level while the battery is damaged. Recovery policies: **P4)** as above.

4 Modelling and operational framework

4.1 Off-board process

The off-board process starts with a fault analysis phase concerning some basic knowledge about the system faults and failures, together with some knowledge about environmental/contextual conditions and their effects and impacts on the system behavior (possibly either nominal or faulty). This phase is aimed at constructing (by standard and well-known dependability analysis procedures) a first dependability model that we assume to be a DFT. The DFT produced can then be automatically compiled into a DBN: the compilation process is detailed in [10]. The DBN model is enriched with knowledge about more specific system capabilities and failures, with particular attention to the identification of multi-state components and stochastic dependencies not captured at the DFT language level. The aim is to generate a DBN representing all the needed knowledge about failure impacts. During this phase, both knowledge about plan actions or recovery actions can be incorporated into the DBN. In ARPHA, we decided to implement the DBN analysis by resorting to JT inference algorithms. So, another role of the off-board process is the generation of the JT from the DBN. This is performed according to the procedures presented in [7].

The DFT definition, its compilation into DBN, and the enrichment of the DBN are supported by the RADYBAN tool [10], previously realized and exploiting *Draw-Net* [11] as graphical interface in order to edit both the DFT and the DBN. An ad-hoc JT generator has been implemented inside VERIFIM, and the resulting JT can be visualized still by means of *Draw-Net*. The XML interchange format [11] of *Draw-Net* has been exploited to define the formalisms DFT, DBN, JT, and generate the respective models.

DFT model of the case study. The DFT model of the case study (Fig. 2) represents the combinations of events or states leading to TE corresponding to

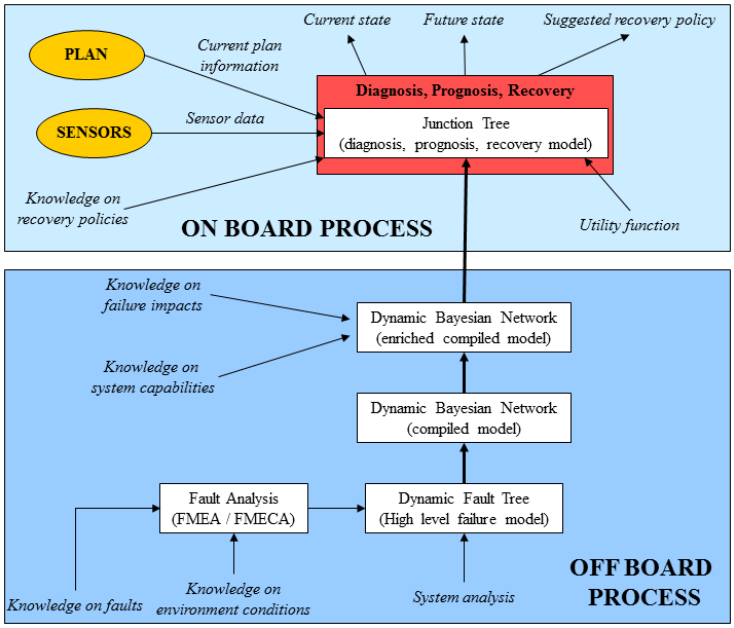


Fig. 1. ARPHA off-board process and on-board process.

the anomaly or failure of the whole system. TE is the output of an OR gate and occurs if the event *S1*, *S2*, *S3*, or *S4* happens. The event *S1* represents the scenario S1 and is the output of an AND gate. *S1* occurs if both the events *PowGen* and *AngleSA2* occur. They represent an anomaly/failure about the power generation (for instance, a low level of generated power) and a not optimal SAA for SA2 (we assume that SAA of SA2 is similar to SAA of SA1 and SA3). *PowGen* occurs if all the events *PowGenSA1*, *PowGenSA2* and *PowGenSA3* happen. Each of them represents the fact that a SA is not producing energy. For example, *PowGenSA1* concerns SA1 and happens if *StringsSA1* occurs (all the strings of SA1 are failed) or *SA1perf* occurs (the combination of local time (day or night), OD and SAA of SA1 does not allow the generation of energy). OD is not optimal in case of storm or shadow.

The event *S2* occurs if both *PowGen* and *OpticalDepth* happen. *S3* occurs if both *BattCharge* and *Drill* occur; they represent an anomaly/failure about the level of charge of the battery, and the drill actions in execution, respectively. *BattCharge* in turns occurs if both the events *Balance* and *BattFail* happen. *Balance* represents the fact that the use of the battery is necessary: *Balance* happens if both *PowGen* and *Load* occur. The second event represents the presence of a load (consume of energy). The event *BattFail* models the damage of the battery because of the failure of all its strings (event *BattStrings*) or a low temperature

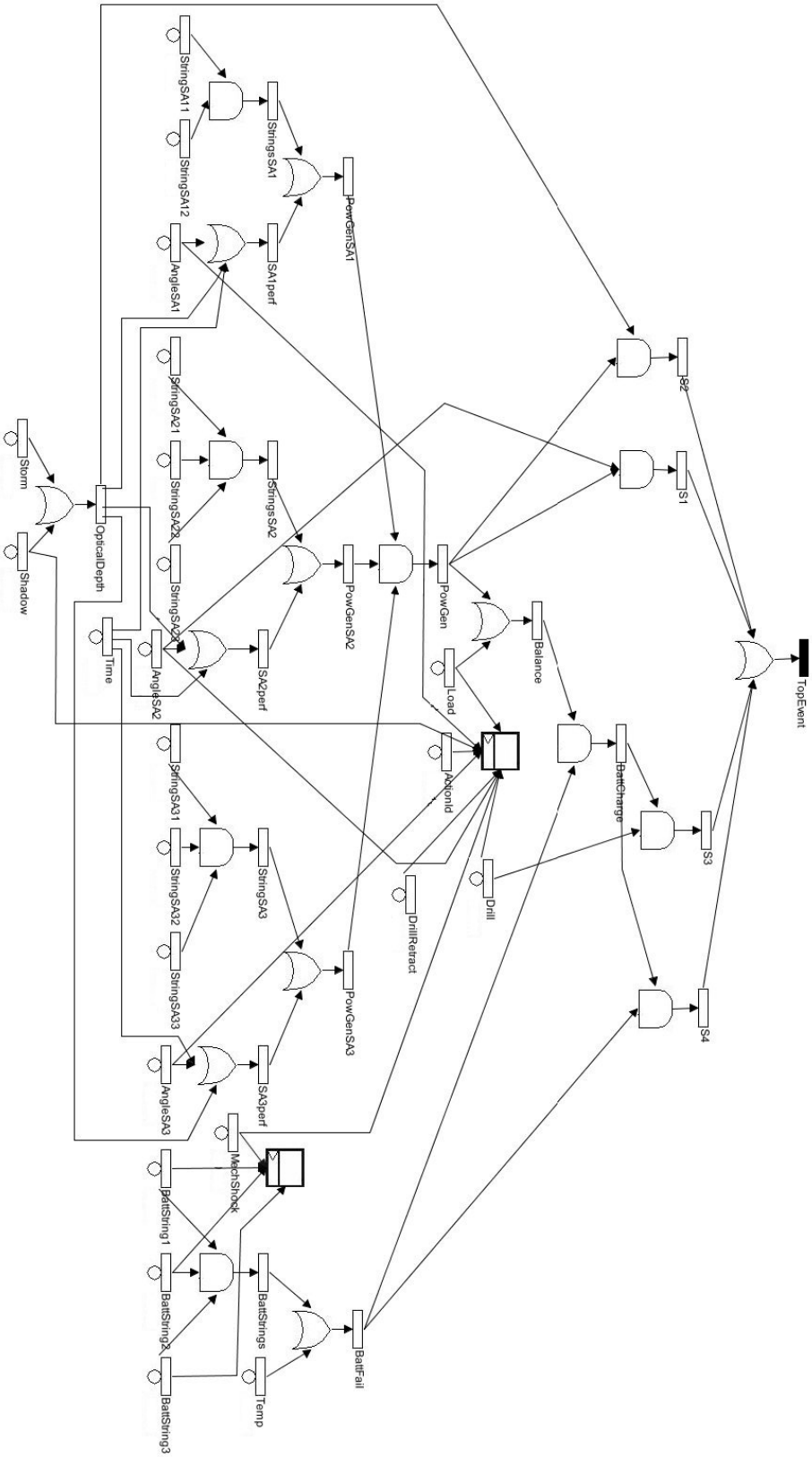


Fig. 2. DFT model of the case study.

(event *Temp*). Finally, *S4* occurs if both the events *BattCharge* and *BattFail* happen.

The model contains two functional dependency (FDEP) [5] gates. The first one represents the influence of *ActionId* on other events, such as *Load*, *Drill* (in case of drilling actions), *DrillRetract* (drill in or out), *AngleSA1*, *AngleSA2*, *AngleSA3* (in case of tilting actions), *Shadow* (in case of travelling actions), and *MechShock* (possibility of mechanical shock damaging the battery strings, in case of drilling or travelling actions). *MechShock* influences in turns the events *BattString1*, *BattString2*, *BattString3* by means of the second FDEP gate.

DBN model of the case study. The DBN of the case study reported in Fig. 3.a has been derived from the DFT model by following two steps: **1)** the DFT has been converted into the equivalent DBN: the structure of the DBN reflects the structure of the originating DFT: each event in the DFT corresponds to the variable in the DBN with the same name, while the DFT gates determine the influence arcs in the DBN. **2)** Then, the DBN has been enriched by increasing the size (number of possible values) of several variables and expressing more complicated relations among the variables, by editing the CPT of the variables.

The DFT contains Boolean (binary) events (variables) representing the state of components or subsystems. This lacks of modeling power in several cases. For instance, the level of power generation, battery charge, or load needs to be represented with a variable with more than two values, if the model has to be accurate enough to capture the aspects of the system behaviour causing its state. Moreover, the relations or dependencies holding between variables may be more complex than a Boolean or dynamic gate. For these reasons, the DBN resulting from the DFT conversion has been enriched in this sense: the variables representing SAA of each SA, and the variable *Temp* are ternary (good, discrete, bad). The size of *PowGen* and *BattCharge* is 4 (we can represent 4 intermediate levels of power generation and battery charge). The size of *Load* is 5 (5 levels of consume of energy). The variable *Balance* is ternary and indicates if *PowGen* is equal, higher or lower than *Load*. The size of *ActionId* is 8 in order to represent 8 actions of interest in the model. The variables *S1*, *S2*, *S3*, *S4* are ternary in order to represent the states *Normal*, *Anomalous* and *Failed* in each scenario (the *Normal* state indicates that the scenario is not happening).

In the DBN we added some support variables in order to reduce the number of entries in the CPT of the non binary variables by applying the so-called "divorcing" technique [10]. The support variables are: *TravelCom*, *DrillCom* and *RetractCom* depending on *ActionId*, and *Trend* depending on *Balance* and *BattFail*. In the DBN, each variable has two instances, one for each time slice ($t-\Delta$, t) (Sec. 2). If a variable has a temporal evolution, its two instances are connected by a "temporal" arc appearing as a thick line in Fig. 3.a. Still in Fig. 3.a, the observable variables are put in evidence (black nodes); the values coming from the sensors will become observations for such variables during the analysis of the model. The plan actions and the recovery actions will become observations for the variable *ActionId*.

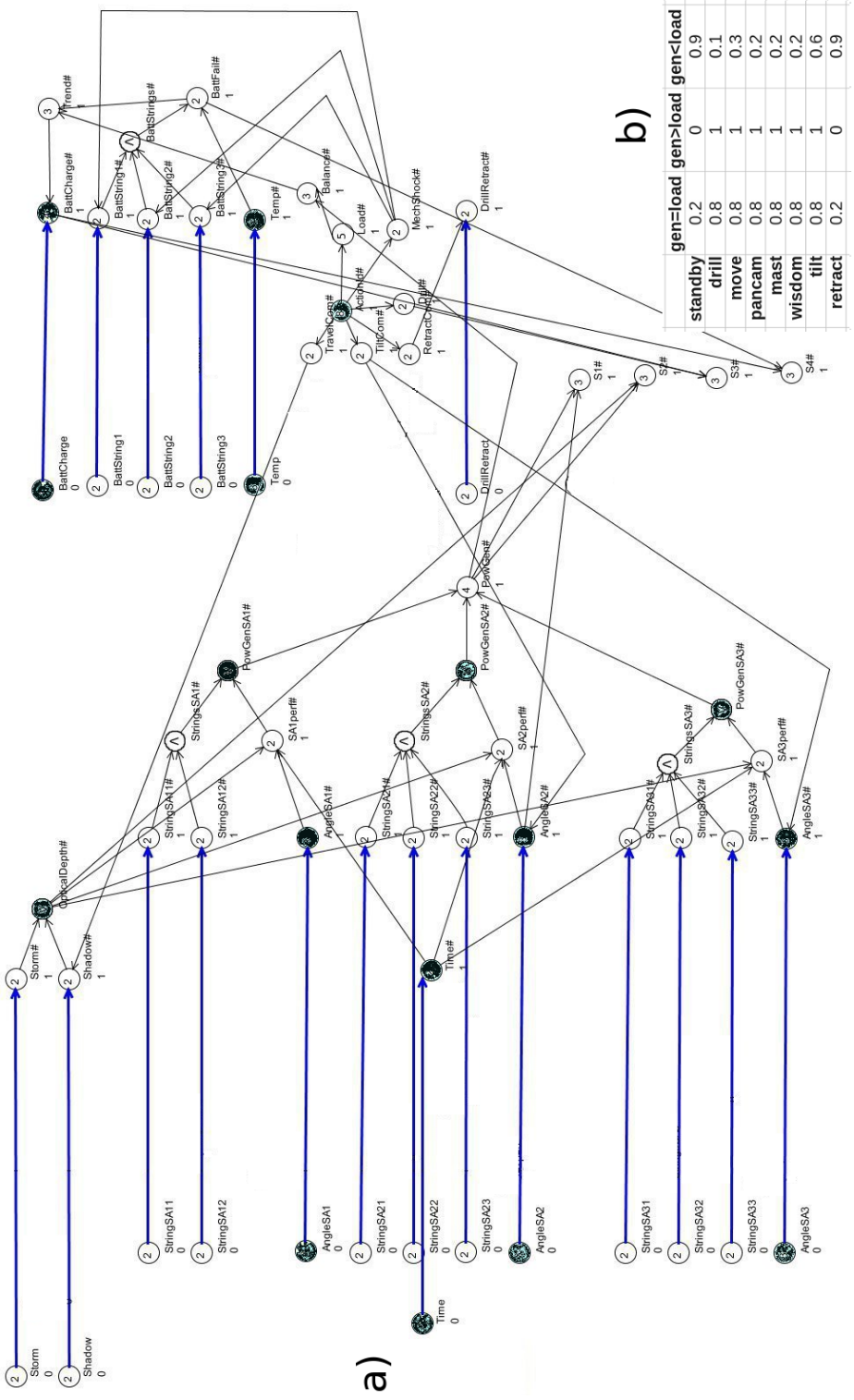


Fig. 3. a) DBN model of the case study. b) utility function.

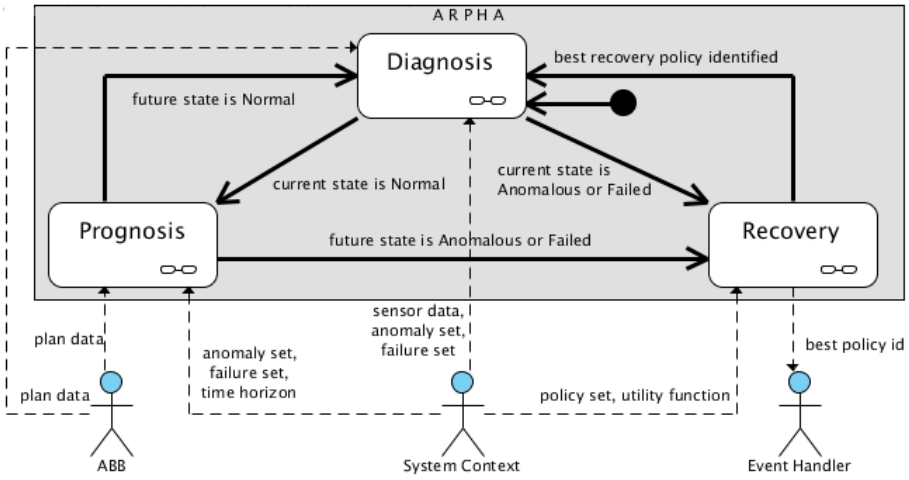


Fig. 4. The UML-like diagram of the functionalities of ARPHA.

The JT (Sec. 2) is derived from the DBN model in Fig. 3.a. The utility function in Fig. 3.b is exploited for Recovery (Sec. 4.2) and provides utility values for the combinations of the possible actions and the balance between power generation and load. The EU of recovery policies is computed according to the utility function and the probability distribution of the variables involved in such function.

4.2 On-board process

The on-board process is performed by the ARPHA prototype and operates on a JT as actual operational model, receiving observations from both sensors and plan actions (Fig. 1). It is intended to perform Diagnosis (current state detection), Prognosis (future state detection) and Recovery (evaluation of the best recovery policy). Fig. 4 shows the external components (actors) that interact with ARPHA: *System Context* (memory area that contains data received from sensors, and the configuration of the system), *Autonomy Building Block* or ABB (dedicated to plan execution and plan generation), *Event Handler* (the manager of events, receiving from ARPHA the id of the policy to be performed to recover the system).

ARPHA cyclically performs the following functionalities (Fig. 4). Since time is discrete in DBN, each cycle is repeated at the begin of a new time step.
Diagnosis begins with the retrieval of data necessary for on-board reasoning. In particular, sensor and plan data are retrieved from System Context and ABB respectively. Both kinds of data are converted in form of observations concerning specific variables of the on-board model. Observations are loaded into the on-board model; then, the model inference (analysis) is performed at the current

time and returns the probability distribution of the variables in the model. The inspection of the probabilities of specific variables representing the system state, can provide the diagnosis at the current mission time: the possible system states are *Normal* (no anomalies or failures are detected), *Anomalous* (an anomaly[5] is detected) or *Failed* (a failure is detected). If the current state detected is *Normal*, then Prognosis is performed, else Recovery is performed (as depicted in Fig. 4).
Prognosis is performed only if the current state is *Normal*, and consists of the future state detection. The on-board model is analyzed in the future according to a specific time horizon and taking into account observations given by future plan actions. Future state is detected according to the probability distribution obtained for the variable representing the system state. The future state can be *Normal*, *Anomalous*, or *Failed*. In case of *Normal* state, the ARPHA on-board process restarts at the next time step, with the Diagnosis phase, otherwise Recovery is performed (Fig. 4).
Recovery can be distinguished in *Reactive Recovery* (performed if the current system state is *Failed*) or *Preventive Recovery* (performed in case of *Anomalous* current state, *Anomalous* future state, or *Failed* future state). In both cases, Recovery is performed in this way: given the detected anomaly or failure, the recovery policies facing that anomaly or failure are retrieved from System Context. In particular, each recovery policy is composed by a set of recovery actions, possibly to be executed at different times. Each policy is evaluated in this way: **1)** the policy is converted into a set of observations for the on-board model variables representing actions; **2)** such observations are loaded in the on-board model which is analyzed in the future; **3)** according to the probability distribution returned by the analysis, and a specific utility function, the expected utility (EU) of the policy is computed. In other words, EU quantifies the future effects of the recovery policy on the system. The policy providing the best EU is selected and notified to the Event Handler for the execution. Then, the ARPHA on-board process restarts at the next time step, with Diagnosis (Fig. 4).

ARPHA is composed by several modules: *System Context Manager*, *Autonomy BB Manager*, *Observation Generator*, *JT Handler*, *State Detector*, *Policy Evaluator*, *Event Manager*, *Logger*. In particular, *JT_Handler* implements the BK inference algorithm (Sec. 2) with the goal of providing the posterior probabilities over the variables of interest to the other components that need them (e.g. *State Detector* and *Policy Evaluator*). The details about the internal architecture of ARPHA are provided in [12]. In order to perform an empirical evaluation of the approach, ARPHA has been deployed in an evaluation platform composed by a workstation linked to a PC via Ethernet cable. A rover simulator (called ROSEX) has been installed on the workstation. On the PC we installed the TSIM environment, emulating the on-board computing hardware/OS environment (LEON3/RTEMS), and the ARPHA executable. ARPHA will run in parallel to other processes of the on-board software.

[5] An anomaly is a malfunctioning possibly leading to a failure in the near future.

Executing ARPHA on the case study. We provide an example of ARPHA execution during a simulated mission; for the sake of brevity, we describe only the initial steps of the mission. Sensors data and plan data that simulator provides are the following: OD, power generated by each SA, SAA of SA1, SA2, SA3, charge of the battery, temperature of battery, mission elapsed time, action under execution, plan under execution. Fig. 5.a shows a graphical representation of the plan. In Fig. 5.b we show the OD profile generated by rover simulator. Fig. 5.c shows the power generation profile related to the OD profile in Fig. 5.b. At the begin of each cycle of the on-board process (Fig. 4), the current sensor data and plan data are retrieved and converted into current or future observations for specific variables in the on-board model. Such observations are expressed as the probability distribution of the possible variable values. For example, the "wait" action in the plan at time step 3, is converted in the probability distribution $1, 0, 0, 0, 0, 0, 0, 0$ concerning the variable *ActionId* in the same step. The first value of the distribution indicates that the first possible value of the variable (0) has been observed with probability 1. This is due to the fact that *ActionId* represents in the model the current plan action (or recovery action). In particular, the value 0 corresponds to the "wait" action. An example about sensor data is the sensor $pwrsa1$ providing the value 17.22273 at time step 3. This value becomes the probability distribution $1, 0$ for the values of the variable *PowGenSA1*. In other words, *PowGenSA1* is observed equal to 0 with probability 1 at the same mission step, in order to represent that SA1 is generating power in that step (the value 1 represents instead the absence of power generation).

Diagnosis. At time steps 0, 1, 2, ARPHA detects *Normal* state as the result of both diagnosis and prognosis. Fig. 5.d shows the output of ARPHA at time step 3 (218 sec.): lines 01-04 contain the values of the sensors (generated by the rover simulator) and the plan action under execution (*SVF_action*); lines 06-16 concern the diagnosis. In particular, at lines 07-08, the plan action (*SVF_action*=1="wait") performed in the current time step, is converted into the observation $ActionId = 0$; at lines 09-12, the sensor values are mapped into observations of the corresponding variable values: $pwrsa1 = 17.22273$ becomes $PowGenSA1 = 0$ (power generation by SA1 is high), $pwrsa2 = 26.67850$ becomes $PowGenSA2 = 0$ (power generation by SA2 is high), $pwrsa1 = 26.67641$ becomes $PowGenSA3 = 0$ (power generation by SA3 is high), $saa1 = 0.51575$ becomes $AngleSA1 = 0$ (SAA1 is optimal), $saa2 = 0.515750$ becomes $AngleSA2 = 0$ (SAA2 is optimal), $saa3 = 0.515750$ becomes $AngleSA3 = 0$ (SAA3 is optimal), $opticaldepth = 4.50000$ becomes $OpticalDepth = 1$ (OD is not optimal), etc. Given such observations, ARPHA performs the inference of the model at the current time step (line 13), querying the variables $S1\#$, $S2\#$, $S3\#$, $S4\#$ representing the occurrence of the scenarios (lines 14-15). The probability that $S1\# = 1$ (anomaly) or $S1\# = 2$ (failure) is lower than a predefined threshold, so S1 is not detected. The same condition holds for scenarios S2, S3, S4, so the result of diagnosis is *Normal* state (line 16).

Prognosis. Since Diagnosis has returned *Normal* state, Prognosis is activated (lines 17-26). The future actions in the plan become observations for the vari-

able *ActionId* (lines 18-20); then, the model inference is executed (line 21), still querying the variables $S1$, $S2$, $S3$, $S4$ (lines 22-25), but analyzing the model in the future (next four time steps). At line 22, $Pr(S2 = 2)$ is greater than a given threshold, so ARPHA detects S2 and in particular, the *Failed* state (line 26).
Recovery. Preventive recovery is activated (lines 27-42) due to Prognosis result, with the aim of evaluating the policies P3 and P4, suitable to deal with S2. At lines 28-30, the actions inside P3 become observations in the next time steps, for the variable $ActionId$. In particular, we observe the "move" action in the future time steps 4 and 5 (line 30). Given such observations, the model is inferenced for 10 time steps in the future (line 31) and EU is computed (line 32). The same procedure is applied to P4 (lines 33-41). The actions inside P4 become evidences for $ActionId$ (lines 34-39): the "tilt" action (SA inclination) is observed at time steps 4 and 5, "retract drill" is observed at time steps 6 and 7, "wait" is observed at time steps from 8 to 13 [6]. According to such observations, the model is analyzed in the future, still for the next 10 time steps (line 40) and EU is computed (line 41). P4 provides a better EU, so P4 is suggested by ARPHA for execution (line 42)[7].

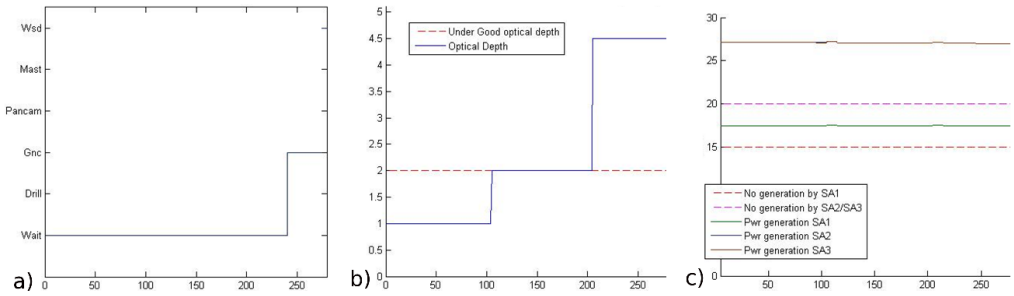
5 Conclusions

ARPHA aims at keeping as much standard as possible the fault analysis phase, by allowing reliability engineers to build their fault models using an intuitive and familiar modelling language such as DFT. By the enrichment of the DBN obtained from the DFT, we are able to address issues that are very important in the context of innovative on-board FDIR: multi-state components with different fault modes, stochastic dependencies among system components, system-environment interactions. A case study has been presented in order to show the steps of the modelling phase and the innovative capabilities of ARPHA: diagnosis under partial observability of the system and the environment, possibility to perform the prognosis, dealing with recovery policies composed by several actions performed at different times, evaluation of the future effects of recovery policies. Actually, ARPHA is a reasoning based FDIR system: diagnosis, prognosis and recovery decisions derive from the analysis (inference) of the on-board model, while traditional FDIR is simply based on sensor monitoring for diagnosis, and look-up tables for recovery actions, without any prognosis capability.

The DFT formalism is rather simple, so the design of the DFT model does not require a modeller with particular skills. The DBN enrichment instead, actually requires a modeller with a specific experience in Bayesian modelling. In particular, the editing of CPTs (Sec. 2) needs a particular attention in order to

[6] The number of actions inside a policy is not constant. Therefore the duration of a policy depends on the number of actions and the duration of each action. For instance, P3 and P4 generate observations for 2 and 10 time steps in the future, respectively, because of their internal actions.

[7] This is justified, since the movement in another position (P3) does not guarantee to improve power generation, while the tilting action in P4 is more effective.



d) ARPHA output:

```
00   *** MISSION STEP: 3 (MISSION TIME: 218 sec.) ***
01   ********** ROSEX VALUES **********
02   opticaldepht = 4.50000  pwrsa1 = 17.22273  pwrsa2 = 26.67850  pwrsa3 = 26.67641
03   saa1 = 0.51575          saa2 = 0.51575     saa3 = 0.51575     batterycharge = 90.28925
04   batttemp = 273.00000    time = 10.05112    SVF_action = 1     SVF_plan = 1
05   ********************************************
06      ## Diagnosis ##
07   Propagate PLAN STREAM
08   3:ActionId#:1 0 0 0 0 0 0 0
09   Propagate SENSORS STREAM
10   3:OpticalDepth#:0 1  3:PowGenSA1#:1 0   3:PowGenSA2#:1 0   3:PowGenSA3#:1 0
11   3:AngleSA1#:1 0 0    3:AngleSA2#:1 0 0  3:AngleSA3#:1 0 0  3:BattCharge#:0 0 0 1
12   3:Temp#:0 1 0        3:Time#:1 0
13   Current inference (STEP 3)
14   Pr{S1#=2}=0.000<0.590  Pr{S2#=2}=0.000<0.590  Pr{S3#=2}=0.000<0.590  Pr{S4#=2}=0.000<0.590
15   Pr{S1#=1}=0.000<0.590  Pr{S2#=1}=0.000<0.590  Pr{S3#=1}=0.000<0.590  Pr{S4#=1}=0.000<0.590
16   SYSTEM STATE: "Normal"
17      ## Prognosis ##
18   Propagate PLAN STREAM
19   4:ActionId#:1 0 0 0 0 0 0 0  5:ActionId#:1 0 0 0 0 0 0 0
20   6:ActionId#:1 0 0 0 0 0 0 0  7:ActionId#:0 0 1 0 0 0 0 0
21   Future inference (STEP 7)
22   Pr{S1#=2}=0.38471501<0.59  Pr{S2#=2}=0.60604805>=0.59          Pr{S3#=2}=0.01966910<0.59
23   Pr{S4#=2}=0.05214530<0.59  Pr{S1#=1} excluded because under recovery or minor criticality
24   Pr{S2#=2} excluded because under recovery or minor criticality  Pr{S3#=1}=0.09944675<0.59
25   Pr{S4#=1}=0.29860398<0.59
26   FUTURE SYSTEM STATE: "Failed" (S2#=2)
27      ## Preventive Recovery ##
28   Policy to convert: P3
29   Propagate POLICY STREAM
30   4:ActionId#:0 0 1 0 0 0 0 0   5:ActionId#:0 0 1 0 0 0 0 0
31   Future inference (STEP 13)
32   Utility Function = 0.0890
33   Policy to convert: P4
34   Propagate POLICY STREAM
35   4:ActionId#:0 0 0 0 0 0 1 0   5:ActionId#:0 0 0 0 0 0 1 0   6:ActionId#:0 0 0 0 0 0 0 1
36   7:ActionId#:0 0 0 0 0 0 0 1
37   8:ActionId#:1 0 0 0 0 0 0 0   9:ActionId#:1 0 0 0 0 0 0 0   10:ActionId#:1 0 0 0 0 0 0 0
38   11:ActionId#:1 0 0 0 0 0 0 0
39   12:ActionId#:1 0 0 0 0 0 0 0  13:ActionId#:1 0 0 0 0 0 0 0
40   Future inference (STEP 13)
41   Utility Function= 0.8764
42   Best policy for Preventive Recovery is P4
```

Fig. 5. Scenario S2: a) plan. b) Optical Depth (OD). c) Power generation by SA1, SA2, SA3. d) ARPHA output at time step (or mission step) 3.

consider any possible case and avoid cases not compatible with observations. In order to limit this problem, inside VeriFIM, DFT formalism has been extended to EDFT [13]. If an automatic translator from EDFT to DBN was developed, the effort to enrich the DBN would be less relevant because several features may be directly modelled in EDFT form, and translated into DBN in automatic way. In order to design an accurate stochastic model, knowledge about probability parameters, such as component failure rates, has to be provided. Such values may not be immediately available. Another not negligible aspect is the link between computing time and model accuracy. The complexity of the DBN model depends on the number of entries in the CPTs of variables. This number depends on the number of possible values and the number of parents of the variables. It is necessary to perform a trade-off between the model accuracy and the computing time, taking into account that on-board hardware has limited computing power.

References

1. Schneeweiss, W.G.: The Fault Tree Method. LiLoLe Verlag (1999)
2. Schwabacher, M., Feather, M., Markosian, L.: Verification and validation of advanced fault detection, isolation and recovery for a NASA space system. In: Proc. Int. Symp. on Software Reliability Engineering, Seattle, WA (2008)
3. Robinson, P., Shirley, M., Fletcher, D., Alena, R., Duncavage, D., Lee, C.: Applying model-based reasoning to the FDIR of the command and data handling subsystem of the ISS. In: Proc. iSAIRAS 2003, Nara, Japan (2003)
4. Glover, W., Cross, J., Lucas, A., Stecki, C., Stecki, J.: The use of PHM for autonomous unmanned systems. In: Proc. Conf. PHM Society, Portland, OR (2010)
5. Dugan, J., Bavuso, S., Boyd, M.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. on Reliability **41** (1992) 363–377
6. Murphy, K.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD Thesis, UC Berkley (2002)
7. Huang, C., Darwiche, A.: Inference in Belief Networks: A Procedural Guide. International Journal of Approximate Reasoning **15** (1996) 225–263
8. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
9. Boyen, X., Koller, D.: Tractable inference for complex stochastic processes. In: Proc. UAI 1998. (1998) 33–42
10. Portinale, L., Bobbio, A., Codetta-Raiteri, D., Montani, S.: Compiling dynamic fault trees into dynamic Bayesian nets for reliability analysis: The Radyban tool. CEUR Workshop Proceedings **268** (2007)
11. Codetta-Raiteri, D., Franceschinis, G., Gribaudo, M.: Defining formalisms and models in the Draw-Net Modelling System. In: Int. Workshop on Modelling of Objects, Components and Agents, Turku, Finland (2006) 123–144
12. Portinale, L., Codetta-Raiteri, D.: ARPHA: an FDIR architecture for Autonomous Spacecrafts based on Dynamic Probabilistic Graphical Models. In: Proc. ESA Workshop on AI in Space @ IJCAI 2011, Barcelona, Spain (2011)
13. Portinale, L., Codetta-Raiteri, D.: Using dynamic decision networks and extended fault trees for autonomous FDIR. In: Proc. Int. Conf. on Tools with Artificial Intelligence, Boca Raton, FL (2011)