Acta Futura 5 (2012) 99-110
DOI: 10.2420/AF05.2012.99

Acta Futura

ARPHA: a software prototype for fault detection, identification and recovery in autonomous spacecrafts

Daniele Codetta-Raiteri, Luigi Portinale *
Dipartimento di Informatica, Università del Piemonte Orientale, Alessandria, Italy

Stefano di Nolfo, Andrea Guiotto
Thales Alenia Space, Strada Antica di Collegno 253 - 10146, Torino, Italy

Abstract. This paper introduces a software prototype called ARPHA for on-board diagnosis, prognosis and recovery. The goal is to allow the design of an innovative on-board FDIR (Fault Detection, Identification and Recovery) process for autonomous systems, able to deal with uncertain system/environment interactions, uncertain dynamic system evolution, partial observability and detection of recovery policies taking into account imminent failures. We propose to base the inference engine of ARPHA on Dynamic Probabilistic Graphical Models suitable to reason about system evolution with control actions, over a finite time horizon. The model needed by ARPHA is derived from standard dependability modeling, exploiting an extension of the Dynamic Fault Tree language, called EDFT. We finally discuss the software architecture of ARPHA, where on-board FDIR is implemented and we provide some preliminary results on simulation scenarios for Mars rover activities.

1 Introduction

Autonomous spacecraft operation relies on the adequate 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 current state-of-the-art of the FDIR (Fault Detection, Identification and Recovery) process is based on the design-time analysis of the faults and failure scenarios (e.g. Failure Mode Effect Analysis or FMEA, Fault Tree Analysis or FTA [11]) and run-time observation of the system operational status (health monitoring). The goal is a timely detection of faults and the initiation of the corresponding recovery action, often using static pre-compiled look-up tables and basically concerned with the execution of the safing actions to put the spacecraft into a known safe configuration and transfer control to the Ground operations.

The classical FDIR approach however, suffers from multiple shortcomings. In particular, the system, as well as its environment, is only partially observable by the FDIR monitoring; this introduces uncertainty in the interpretation of observations in terms of the actual system status. Moreover, classical FDIR represents a reactive

*Corresponding author. E-mail: luigi.portinale@di.unipmn.it

approach, that cannot provide and utilise prognosis for the imminent failures. 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 [12] or to systems requiring heavy human intervention [9], some others by emphasizing the prognostic phase and relying on heuristics techniques to close the FDIR cycle [4]. A more formal approach to on-board FDIR seems to be needed, having the capability to reason about anomalous observations in the presence of uncertainty, dynamic evolution and partial observability. The main issue is to define a unifying formal framework providing the system with diagnosis and prognosis on the operational status to be taken into account for autonomous preventive recovery actions. In this paper, a formal model for knowledge-based reasoning based on Probabilistic Graphical Models is proposed, with the aim of enabling on-board FDIR reasoning. While the final goal of the study will be to develop a demonstrator performing proof-of-concept case studies for the innovative FDIR element of an autonomous spacecraft, the paper concentrates on the formal modeling, inference, specification and design of an on-board FDIR architecture called ARPHA (Anomaly Resolution and Prognostic Health management for Autonomy), designed to address on-board reasoning about the impact of system and environment state on spacecraft capabilities and mission execution.

The paper is organized as follows: Sec. 2 discusses issues concerning modeling causal probabilistic knowledge; Sec. 3 describes the functional requirements of ARPHA included in the off-board and on-board process, and introduces the DDN (Dynamic Decision Network) model to be used for the actual FDIR analysis by ARPHA; Sec. 4 describes the formal software architecture of ARPHA; finally, Sec. 5 presents a case study evaluated using ARPHA in terms of diagnosis, prognosis and recovery.

2 Modeling Causal Probabilistic Knowledge

Modeling probabilistic causal dependencies is one of the main capabilities of Probabilistic Graphical Models (PGM) like Bayesian Networks (BN), Decision Networks (DN) and their dynamic counterparts as Dynamic Bayesian Networks (DBN) and Dynamic Decision Networks (DDN) [5]. From an FDIR perspective, this class of models naturally captures dependencies and evolutions under partial observability; moreover, in decision models also the effect of autonomous actions can be modeled and utility functions can be exploited in order to select most useful actions. For this reason, we propose a formal architecture called ARPHA based on the model of DDNs. DDNs are essentially DBNs augmented with decision nodes and utility functions. DBNs are, in turn, a factored representation of a discrete time Markov process, where the global system state is determined by the Cartesian product of a set of discrete variables obeying to Markovian state transitions (see [5, 7] for more details). Solving a DDN means finding a sequence of decisions maximizing the total expected utility over a specified horizon; this means that, in principle every algorithm for solving a Markov Decision Process (MDP) [10] can be adopted. However, from an on-board FDIR perspective, globally optimal sequences can be too hard to be obtained, both in terms of time and computational resources. For these reasons, DDNs are proposed as the suitable target model for ARPHA, by adopting an on-line inference strategy [10], where observations on monitored parameters are processed as soon as they become available to the system. This allows for the choice of a locally (i.e. at the current time) best recovery action, given the current stream of observations and the future possible states of the modeled system, providing a tight connection between diagnosis, recovery and prognosis. Furthermore, by taking into account both the current "belief state" of the system (summarizing the history of the system uncertain evolution) and the effects of the recovery actions on future system states, the task of preventive recovery can be addressed.

Even if the ARPHA architecture is designed as an on-board inference engine, it has to rely on a suitable off-board modeling phase, producing the model on which on-board inference has to take place. As mentioned before, the target model on which ARPHA works is a DDN, which is however a class of models unfamiliar to most reliability engineers; they are usually more familiar with other formalisms and techniques supporting classical FDIR task like Fault Tree Analysis (FTA). However, Fault Trees (FT) [11] are limited to model systems with independent binary components (i.e. characterized by the "ok-faulty" dual behavioral modes, failing independently from other compo-

 DOI: 10.2420/AF05.2012.99

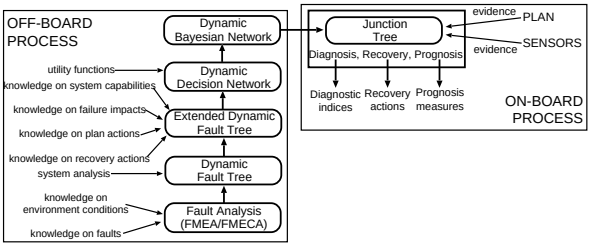

Figure 1. *ARPHA on-board reasoning process plus off-board process.*

nents in the system). For this reason, several extensions have been proposed, either to address specific stochastic dependencies as in Dynamic Fault Tree (DFT) [3] or to allow the modeling of "multi-state" components [2]. Concerning the off-board process of ARPHA we have proposed to extend the formalism of DFTs to another formal modeling language called Extended Dynamic Fault Tree (EDFT) [8]; in this extension, a generalization of both Boolean components to multi-state components, as well as a generalization of the stochastic dependencies allowed by the DFT formalism are introduced. The idea is to provide the modeler with a formal language able to express, in a FT-based style, a set of complex component interactions, while being at the same time, suitable for a general FDIR analysis. The proposed approach is then to compile a DDN (subsequently transformed into a DBN for analysis) from the input EDFT model, and then using a suitable algorithm for on-line inference to perform the FDIR task.

3 Specification of ARPHA

The ARPHA architecture puts emphasis on the on-board software capabilities; however, as we mentioned before, an off-board processing phase is necessary, in order to provide it with the inputs and the needed operational model. Fig. 1 summarizes the basic scheme of the ARPHA on-board reasoning process, involving the interactions with the off-board processing phase.

3.1 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. Starting from this first analysis, the DFT 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 an EDFT representing all the needed knowledge about failure impacts. During this phase, both knowledge about external actions (like plan actions) or control actions (useful to perform recovery) can be incorporated into the EDFT model.

The EDFT produced can then be compiled into a DDN: the compilation process is essentially based on the compilation of a DFT into a DBN (whose details can be found in [6]), with the addition of the compilation of stochastic dependencies not captured at the DFT modeling level (that can be mapped into suitable conditional probability entries of the variables concerning inputs and output of the gate), of external actions (that can be mapped into specific nodes, assumed to be always observed as evidence) and of control actions/policies (that can be mapped into states of a decision node). To complete the DDN, the analyst specifies the utility function by identifying the set of relevant variables, and by building the corresponding utility table taking into account such variables and the control actions available. The DDN is then transformed into a DBN for the analysis, by creating nodes for the actions of the modeled policies (i.e. each policy will be evaluated by instantiating such nodes, then computing the expected utility through DBN inference). Next sections will detail such aspects.

A DDN Model Characterization for On-board FDIR

As introduced in Sec. 2, DDN models are good candidates for addressing the innovative FDIR issues mentioned in Sec. 1. For this reason, ARPHA assumes a particular DDN model as the operational model on which to implement the whole FDIR algorithm. Since ARPHA is intended to provide FDIR capabilities to an autonomous device, interacting with an Autonomy Building Block (ABB) setting and executing a given mission plan, we assume the following characterization concerning DDN nodes:

1) Observable nodes:

- `Plan` nodes whose values (states) are the possible ac-

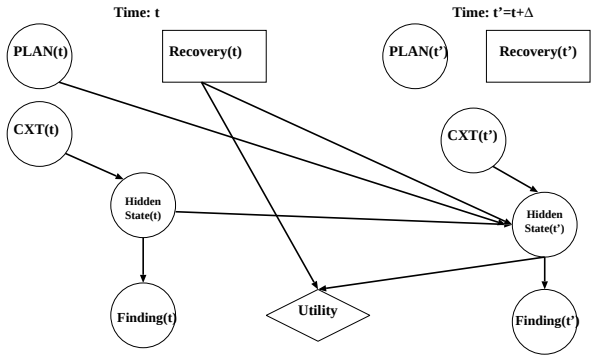

Figure 2. *The DDN scheme for ARPHA FDIR.*

tions the planner can execute: these nodes are assumed to be always set by the ABB;
- a decision node `Recovery` whose values (states) are the possible recovery and control policies the autonomous device can execute[1]; - a set of *Sensor Nodes* representing possible measurements from the device's sensors which in turn can be: *Context Nodes* representing contextual or environmental conditions; *Finding Nodes* representing monitored device parameters such as measurements of specific system variables.
2) Hidden Nodes: representing internal state conditions of the system which are not directly measurable. A subset of hidden nodes are identified as *Diagnostic Nodes* and represent variables target of the diagnostic process (see in the following).

The network high-level scheme of the DDN model used by ARPHA is shown in Fig. 2. The scheme encodes the following general assumptions: time is assumed to be discrete with a discretization step of Δ time units; contextual information influences system internal state within the same time slice; both plan as well as recovery actions have influence on the future system state (i.e. on system variables at the next time slice); system state transition model is then determined by plan and recovery actions and the current state[2]; the utility function to be optimized, in order to choose the best recovery action, depends on the chosen recovery action and the system state determined by the action.

[1]For the sake of simplicity, we assume that all possible recoveries are the values (states) of a single decision node. Actually, such states can represent *recovery policies*, that is set of atomic actions. So, setting a value (or state) of the decision node, means setting a specific recovery policy, which means in turn to set values to every model's variable involved in the policy. This will be more clear in the following.

[2]This is the standard assumption about state transition in MDP.

Inference Approach

In ARPHA, we decided to implement the DDN analysis by resorting to Junction Tree (JT) inference algorithms [5]. In this class of algorithms, once the JT structure is obtained, one can get rid of the original network, so another role of the off-board process is the generation of the JT from the DDN. It is worth noting that a specific instantiation of the decision node will transform the DDN in a DBN; in particular, given the actions/commands composing a policy, DBN nodes can be associated to such actions/commands and the instantiation of a given policy (i.e. the state of the decision node) can be obtained by suitably setting instances of such DBN nodes. In ARPHA, we implemented a parametric JT-based inference strategy: the Boyen-Koller (BK) algorithm [1]. The algorithm depends on some input parameters (set of nodes) called "clusters"; according to the clusters provided it can produce approximate inference results with different degrees of accuracy. In particular, if the input is a unique cluster containing all the so-called "interface nodes" of the DBN, the BK algorithm performs exact inference (see [1] 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[3]. Since the inference procedures will be performed on board, the JT will be the actual operational model undergoing analysis by the on-board process of ARPHA, with diagnosis, recovery, and prognosis purposes, as we will see next.

3.2 On-board process

The on-board process operates on a JT as actual operational model, receiving evidence from both sensors (for contextual as well as finding information) and the ABB (for plan actions); it is intended to produce recovery actions (to be translated into autonomous control action commands), as well as diagnostic and prognostic indices (see Fig. 1). We refer to the following characterization of the FDIR process:
- *Diagnosis* at time `t`: a belief state on the set of diagnostic nodes `D` at time `t`, i.e. the posterior probability at time `t` of each $\mathtt{d} \in \mathtt{D}$ given the evidence (from `Plan`

[3]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.

 DOI: 10.2420/AF05.2012.99

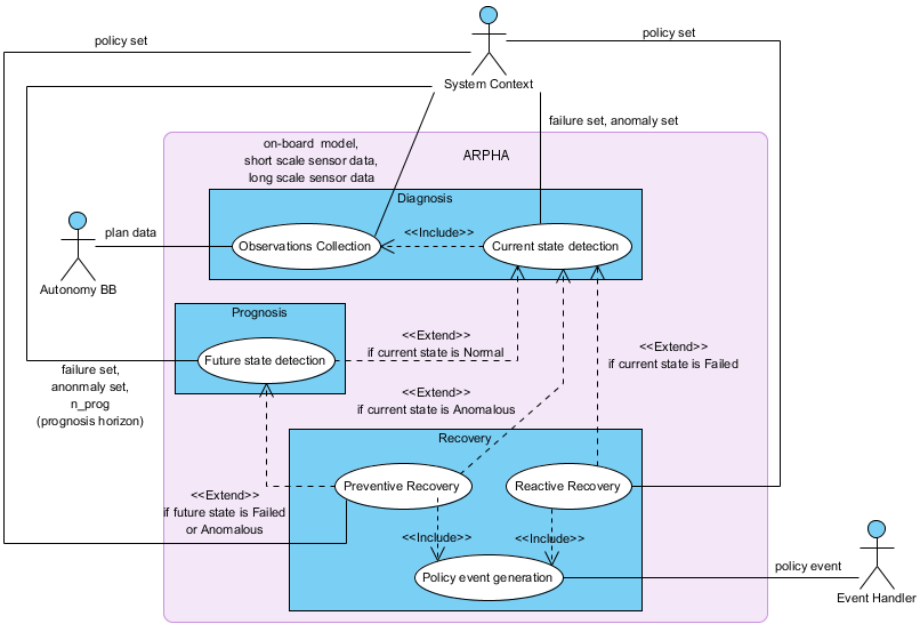

Figure 3. *The UML use case diagram of ARPHA.*

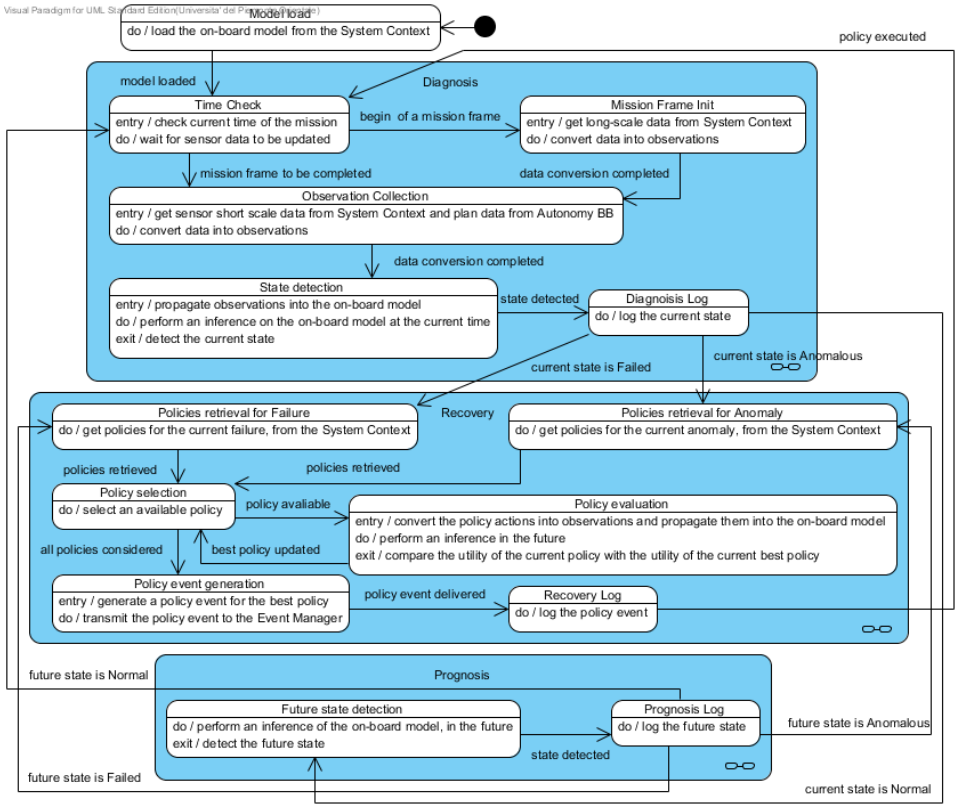

Figure 4. *The UML state-chart diagram of ARPHA.*

and Sensor Nodes) up to time t;
- *Recovery* at time t: choice of the "best" policy r from Recovery node at time t, given the evidence up to time t;
- *Prognosis* at time t' from time $t < t'$: the belief state of set D at time t, given the observations up to time t (and possibly plan information up to t' if available);
We also define the following notions:
- *Discretization step*: the time interval Δ between two consecutive inferences;
- *Mission Frame*: the time interval concerning the analysis, starting from an initial time instant t_0, ending in a time instant t_f and discretized into intervals of width Δ, i.e. $MF = [t_0, t_0 + \Delta, \ldots, t_f - \Delta, t_f]$.

The UML use case diagram in Fig. 3 represents the main functionalities of ARPHA. The actors that interact with ARPHA are the following:
- *System Context*: it represents memory area that contains data received from sensors and configuration of system;
- *Autonomy BB*: it represents an autonomy building block dedicated to plan execution and plan generation.
- *Event Handler*: it represents the manager of events receiving from ARPHA the id of the policy to be performed to recover the system.

ARPHA cyclically (at each time step) performs the following sequence of use cases:
- *Observation Collection*: it periodically retrieves data necessary for on-board reasoning. More specifically, ARPHA periodically checks the current mission time: if the mission has just begun, then ARPHA loads the initial version of the on-board model from the System Context; if a new mission frame has just begun, then ARPHA retrieves the long scale sensor data (available for the whole mission frame), still from the System Context. At each time slice, sensor and plan data are then retrieved from the System Context and the Autonomy BB respectively. Both kinds of data are converted in form of observations concerning the variables of the on-board model.
- *Current state detection*: observations are loaded into the on-board model; then, inference is executed by JT propagation. Inspection of the probabilities of the diagnostic variables can provide the diagnosis at the current mission time. The possible system states are *Normal* (no anomalies or failures are detected), *Anomalous* (an anomaly is detected) or *Failed* (a failure is detected).
- *Reactive Recovery*: this kind of recovery is performed if the current state detection returns a *Failed* state. After having incorporated the current evidence in the diagnostic phase, for each available recovery policy (i.e. for each possible state of the recovery node), the policy itself is loaded (propagated) into the on-board JT; this actually means to set a specific value to the DBN variables affected by the policy itself. The expected utility of each policy is then computed by setting in the JT the corresponding evidence and by propagating it. The policy with the maximum expected utility is then determined; such policy is converted into a sequence command to be executed by the actuator components (possibly at different times), which are then delivered to the Event Handler for the execution.
- *Future state detection*: if the current state is *Normal*, then the time horizon t' for prognosis is determined and JT inference is performed with a time step of Δ until t', by considering plan information at each time step as evidence.
- *Preventive Recovery*: this kind of recovery is performed if the current state detection returns an *Anomalous* state, or if the future state detection provides an *Anomalous* or *Failed* state. The choice of the best recovery policy follows the same approach applied for the Reactive Recovery use case, but according to a certain time horizon if the preventive recovery is a consequence of the future state detection.

The operations performed inside each use case are represented by the UML state-chart diagram in Fig. 4.

4 Design of ARPHA

The software architecture of ARPHA is composed by the following components represented by the UML component diagram in Fig. 5:
- *Main*: it implements the main program capabilities and controls the other components;
- *System_Context_Manager*: it implements functions dedicated to retrieve and manage data contained in System Context;
- *Autonomy_BB_Manager*: it implements functions dedicated to interface the Autonomy BB in order to obtain plan data;
- *Observation_Generator*: it converts sensor data and plan data into observations to be propagated into the on-board model (i.e. the JT); in particular, it maps each sensor with the corresponding variable in the model and sets the variable value according to the sensor reading (possibly performing discretization if the sensor reads a continuous value);
- *JT_Handler*: it implements propagation of observa-

 DOI: 10.2420/AF05.2012.99

tions and actions into the on-board model, it computes the expected utility and gives the current or future belief state;
- *State_Detector*: it examines the current or future belief state in order to detect the current or future state of the system respectively (*Normal, Anomalous, Failed*);
- *Policy_Evaluator*: it manages the evaluation of the best recovery policy;
- *Event_Manager*: it manages the Event Handler, in order to send the action to be performed;
- *Logger*: it implements the logger capabilities.
The *Main* component coordinates the components involved in each use case, while the core of the architecture is the *JT_Handler* component: it implements the BK inference algorithm with the goal of providing the posterior probabilities over the variables of interest to the other components that need them (e.g. *State_Detector*s and *Policy_Evaluator*). The *Logger* records all the probabilistic computations performed by the *JT_Handler* and can then provide such logs to Ground when requested.

5 A Case study

An example case study we have used to test ARPHA concerned the power supply system of a Mars rover, with a particular attention to the following aspects and their combinations: power supply by the solar arrays, load, power supply by the battery, a set of possible analysis scenarios.
Solar arrays. We assume the presence of 3 solar arrays, namely SA1, SA2, SA3. In particular, SA1 is composed by two redundant strings, while SA2 and SA3 are composed by three strings. Each solar array can generate power if two conditions hold: 1) at least one string is not failed; 2) the combination of sun aspect angle, optical depth, and local time (day or night) is suitable. In particular, the optical depth is given by the presence or absence of shadow or storm.

The total amount of generated power is proportional to the number of solar arrays which are actually working.
Load. The amount of load depends on the current 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 the solar arrays. 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 4 failure or anomaly scenarios. Each scenario can be recovered by specific policies:
S1) low power generation while the sun aspect angle is not optimal. Recovery policies: **P1)** suspension of the plan in order in order to reduce the load; **P2)** suspension of the plan and change of inclination of SA2 and SA3 in order to try to improve the sun aspect angle and consequently the power generation (the tilting system cannot act on SA1).
S2) low power generation while the optical depth is not optimal. Recovery policies: **P3)** movement of the rover into another position in order to try to avoid a shadowed area and improve 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 battery level while drilling. Recovery policies: **P4)** as above; **P5)** retraction of the drill, suspension of the plan.
S4) low battery level while the battery is damaged. Recovery policies: **P4)** as above.

5.1 DBN model

The off-board processing phase has produced, starting from an extended fault tree analysis, a DBN model to be suitably compiled in the operational JT model for on-board reasoning (see again Fig. 1). In particular the final DBN (Fig. 10.a) has the following features:
Solar arrays. The variables representing the functioning or the failure of basic components or subsystems, are binary; for example, *StringSA11* and *StringSA12* represent the state of the redundant strings composing the solar array SA1, while *StrinsSA1* represent the state of the set of strings. The variables modeling environmental conditions are binary in order to represent the presence or the absence of such conditions; this is the case of the variables *Storm* and *Shadow* influencing *OpticalDepth*, and the variable *Time* (day or night). The variable *AngleSA1* representing the sun aspect angle of SA1 is ternary (good, discrete, bad). The fact that the combination of *OpticalDepth*, *Time*, *AngleSA1* allows or compromise the power generation by SA1, is represented by the binary variable *SA1perf*. Such variable, together with *StringsSA1*, influences the binary variable *PowGenSA1* modelling the presence or absence of power generation by SA1. The solar arrays SA2 and SA3 are

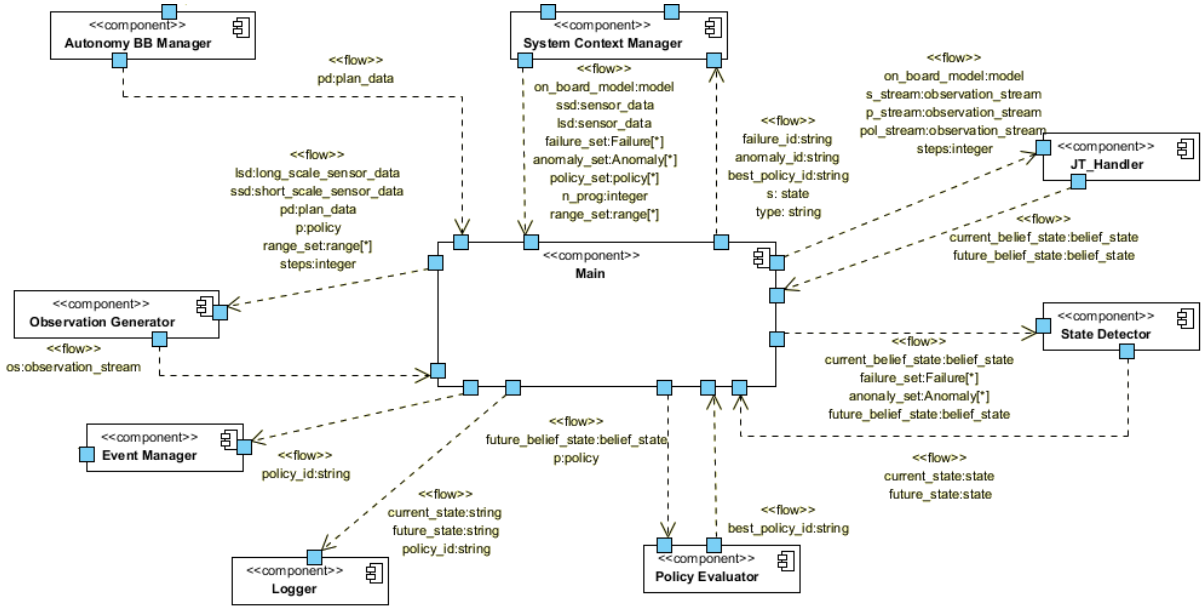

FIGURE 5. *The UML component diagram of ARPHA.*

modelled in the same way. The variable *PowGen* is influenced

by *PowGenSA1*, *PowGenSA2*, *PowGenSA3*. The size of *PowGen* is 4 in order to represent 4 intermediate levels of power generation depending on the number of solar arrays generating power.

Load. The size of the variable *ActionId* is 8 in order to represent 8 actions of interest in the model (in this example actions may concern either the plan or the recovery). Such variable influences *Load* whose size is 5 in order to represent 5 intermediate levels of power consumption according to the action under execution. The variable *Balance* is ternary and indicates if *PowGen* is equal, higher or lower than *Load*.

Battery. The state (working or failed) of each redundant string composing the battery is represented by *BattString1*, *BattString2*, *BattString3*, while the state of the set of strings is modelled by *BattStrings*. The variable *Temp* is ternary and represents the temperature of the battery (low, medium, high). *Temp* and *BattStrings* influence *BattFail* representing the damage of the battery. The variable *Trend* indicates if the battery charge is steady, increasing or decreasing, according to *Balance* and *BattFail*. Four intermediate levels of battery charge are represented by the variable *BattCharge*.

Scenarios. 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 does not currently occur).

In the DBN, each variable has two instances, one for each time slice $(t, t + \Delta)$. If a variable has a temporal evolution, its two instances are connected by a temporal arc (appearing as a thick line in Fig. 10.a). Still in Fig. 10.a, the observable variables appear as black nodes; the values coming from the sensors (and the recovery policies) will become observations for such variables during the ARPHA cycles and the inference analysis of the model.

5.2 Executing ARPHA

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 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. Sensors data and plan data that simulator provides are the following: optical depth, power generated by each solar array, sun aspect angle of SA1, SA2, SA3, charge of the battery, temperature of

 DOI: 10.2420/AF05.2012.99

```
*** MISSION STEP: 0 (MISSION TIME 11)***
  *** Diagnosis ***
Propagate PLAN STREAM
0:ActionId:1 0 0 0 0 0 0 0 0 :1:ActionId#:1 0 0 0 0 0 0 0 0 :2:ActionId#:1 0 0 0 0 0 0 0 0
:3:ActionId#:1 0 0 0 0 0 0 0 0 :4:ActionId#:1 0 0 0 0 0 0 0 0 :
Propagate SENSORS STREAM
0:OpticalDepth:1 0 :0:PowGenSA1:1 0 :0:PowGenSA2:1 0 :0:PowGenSA3:1 0 :0:AngleSA1:1 0 0
:0:AngleSA2:1 0 0 :0:AngleSA3:1 0 0 :0:BattCharge:0 0 0 1 :0:Temp:0 1 0 :0:Time:1 0 :
NO FAILURE Pr{S1=2} = 0.30000000 (0.99000000)      NO FAILURE Pr{S2=2} = 0.30000000 (0.59000000)
NO FAILURE Pr{S3=2} = 0.30000000 (0.99000000)      NO FAILURE Pr{S4=2} = 0.30000000 (0.99000000)
NO ANOMALY Pr{S1=1} = 0.30000000 (0.99000000)      NO ANOMALY Pr{S2=1} = 0.30000000 (0.59000000)
NO ANOMALY Pr{S3=1} = 0.30000000 (0.99000000)      NO ANOMALY Pr{S4=1} = 0.30000000 (0.99000000)
                                        STATE SYSTEM "NORMAL"
  ## Prognosis ##
NO FAILURE Pr{S1#=2} = 0.38482874 (0.99000000)     NO FAILURE Pr{S2#=2} = 0.56793149 (0.59000000)
NO FAILURE Pr{S3#=2} = 0.00720536 (0.99000000)     NO FAILURE Pr{S4#=2} = 0.01865137 (0.99000000)
NO ANOMALY Pr{S1#=1} = 0.08852395 (0.99000000)     NO ANOMALY Pr{S2#=1} = 0.14727087 (0.59000000)
NO ANOMALY Pr{S3#=1} = 0.10020495 (0.99000000)     NO ANOMALY Pr{S4#=1} = 0.27252826 (0.99000000)
                                    FUTURE STATE SYSTEM "NORMAL"
*** MISSION STEP: 2 (MISSION TIME 63)***
  *** Diagnosis ***
Propagate PLAN STREAM
1:ActionId#:1 0 0 0 0 0 0 0 0 :2:ActionId#:1 0 0 0 0 0 0 0 0 :3:ActionId#:1 0 0 0 0 0 0 0 0
:4:ActionId#:1 0 0 0 0 0 0 0 0 :5:ActionId#:1 0 0 0 0 0 0 0 0 :
Propagate SENSORS STREAM
1:OpticalDepth#:1 0 :1:PowGenSA1#:1 0 :1:PowGenSA2#:1 0 :1:PowGenSA3#:1 0 :1:AngleSA1#:1 0 0
:1:AngleSA2#:1 0 0 :1:AngleSA3#:1 0 0 :1:BattCharge#:0 0 0 1 :1:Temp#:0 1 0 :1:Time#:1 0 :
NO FAILURE Pr{S1#=2} = 0.00000000 (0.99000000)      NO FAILURE Pr{S2#=2} = 0.00000000 (0.59000000)
NO FAILURE Pr{S3#=2} = 0.00000000 (0.99000000)      NO FAILURE Pr{S4#=2} = 0.00000000 (0.99000000)
NO ANOMALY Pr{S1#=1} = 0.00000000 (0.99000000)      NO ANOMALY Pr{S2#=1} = 0.00000000 (0.59000000)
NO ANOMALY Pr{S3#=1} = 0.00000000 (0.99000000)      NO ANOMALY Pr{S4#=1} = 0.00000000 (0.99000000)
        STATE SYSTEM "NORMAL"
  ## Prognosis ##
NO FAILURE Pr{S1#=2} = 0.31664297 (0.99000000)      NO FAILURE Pr{S2#=2} = 0.27930299 (0.59000000)
NO FAILURE Pr{S3#=2} = 0.00421664 (0.99000000)      NO FAILURE Pr{S4#=2} = 0.01022689 (0.99000000)
NO ANOMALY Pr{S1#=1} = 0.06083531 (0.99000000)      NO ANOMALY Pr{S2#=1} = 0.07766408 (0.59000000)
NO ANOMALY Pr{S3#=1} = 0.09709793 (0.99000000)      NO ANOMALY Pr{S4#=1} = 0.24639760 (0.99000000)
           FUTURE STATE SYSTEM "NORMAL"
*** MISSION STEP: 2 (MISSION TIME: 124) ***
  *** Diagnosis ***
Propagate PLAN STREAM
2:ActionId#:1 0 0 0 0 0 0 0 0 :3:ActionId#:1 0 0 0 0 0 0 0 0 :4:ActionId#:1 0 0 0 0 0 0 0 0 :
5:ActionId#:1 0 0 0 0 0 0 0 0 :6:ActionId#:1 0 0 0 0 0 0 0 0 :
Propagate SENSORS STREAM
2:OpticalDepth#:1 0 :2:PowGenSA1#:1 0 :2:PowGenSA2#:1 0 :2:PowGenSA3#:1 0 :2:AngleSA1#:0 1 0
:2:AngleSA2#:0 1 0 :2:AngleSA3#:0 1 0 :2:BattCharge#:0 0 0 1 :2:Temp#:0 1 0 :2:Time#:1 0 :
NO FAILURE Pr{S1#=2} = 0.00000000 (0.99000000)      NO FAILURE Pr{S2#=2} = 0.00000000 (0.59000000)
NO FAILURE Pr{S3#=2} = 0.00000000 (0.99000000)      NO FAILURE Pr{S4#=2} = 0.00000000 (0.99000000)
NO ANOMALY Pr{S1#=1} = 0.00000000 (0.99000000)      NO ANOMALY Pr{S2#=1} = 0.00000000 (0.59000000)
NO ANOMALY Pr{S3#=1} = 0.00000000 (0.99000000)      NO ANOMALY Pr{S4#=1} = 0.00000000 (0.99000000)
        STATE SYSTEM "NORMAL"
  ## Prognosis ##
NO FAILURE Pr{S1#=2} = 0.40137030 (0.99000000)      NO FAILURE Pr{S2#=2} = 0.28233399 (0.59000000)
NO FAILURE Pr{S3#=2} = 0.00411488 (0.99000000)      NO FAILURE Pr{S4#=2} = 0.00981223 (0.99000000)
NO ANOMALY Pr{S1#=1} = 0.07227470 (0.99000000)      NO ANOMALY Pr{S2#=1} = 0.07607630 (0.59000000)
NO ANOMALY Pr{S3#=1} = 0.09682351 (0.99000000)      NO ANOMALY Pr{S4#=1} = 0.24128415 (0.99000000)
         FUTURE STATE SYSTEM "NORMAL"
*** MISSION STEP: 3 (MISSION TIME: 187) ***
  *** Diagnosis ***
Propagate PLAN STREAM
3:ActionId#:1 0 0 0 0 0 0 0 0 :4:ActionId#:1 0 0 0 0 0 0 0 0 :5:ActionId#:1 0 0 0 0 0 0 0 0
:6:ActionId#:1 0 0 0 0 0 0 0 0 :7:ActionId#:0 0 1 0 0 0 0 0 :
Propagate SENSORS STREAM
3:OpticalDepth#:1 0 :3:PowGenSA1#:1 0 :3:PowGenSA2#:0 1 :3:PowGenSA3#:0 1 :3:AngleSA1#:0 1 0
:3:AngleSA2#:0 1 0 :3:AngleSA3#:0 1 0 :3:BattCharge#:0 0 0 1 :3:Temp#:0 1 0 :3:Time#:1 0 :
NO FAILURE Pr{S1#=2} = 0.00000000 (0.99000000)     NO FAILURE Pr{S2#=2} = 0.00000000 (0.59000000)
NO FAILURE Pr{S3#=2} = 0.00000000 (0.99000000)     NO FAILURE Pr{S4#=2} = 0.00000000 (0.99000000)
Pr{S1#=1} = 1.00000000] >= 0.99000000
        STATE SYSTEM "ANOMALOUS" (1)
  ## Preventive Recovery ##
Policy to convert: P1
policy_observation_stream : 4:ActionId#:1 0 0 0 0 0 0 0 0 :5:ActionId#:1 0 0 0 0 0 0 0 0 :6:ActionId#:1 0 0 0 0 0 0 0 0
:7:ActionId#:1 0 0 0 0 0 0 0 0 :8:ActionId#:1 0 0 0 0 0 0 0 0 :9:ActionId#:1 0 0 0 0 0 0 0 0 :10:ActionId#:1 0 0 0 0 0 0 0 0
:11:ActionId#:1 0 0 0 0 0 0 0 0 :12:ActionId#:1 0 0 0 0 0 0 0 0 :13:ActionId#:1 0 0 0 0 0 0 0 0 :
Future inference 13
Utility Function= 0.4736
Policy to convert: P2
policy_observation_stream : 4:ActionId#:0 0 0 0 0 0 1 0 :5:ActionId#:0 0 0 0 0 0 1 0 :
Utility Function= 0.0622
Best policy for Preventive Recovery is P1
```

Figure 6. *ARPHA output.*

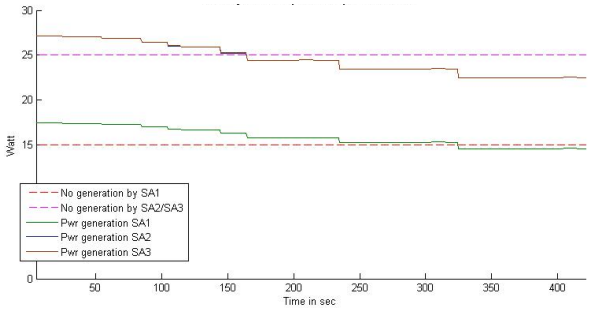

Figure 7. *Power generation by the solar arrays during the mission.*

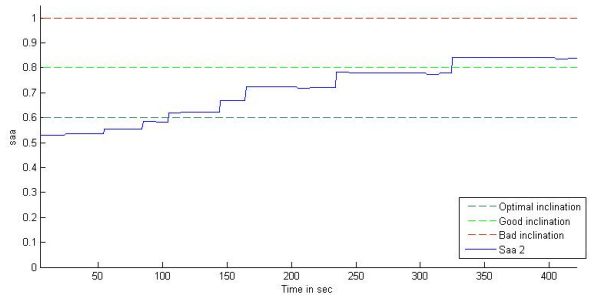

Figure 8. *Sun aspect angle during the mission.*

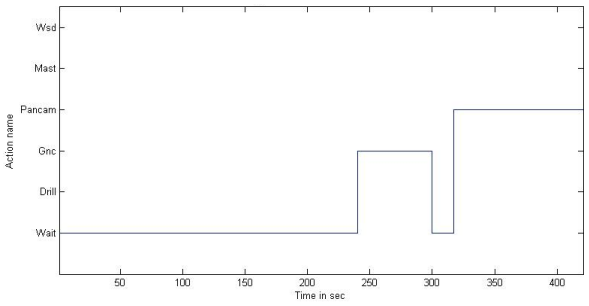

Figure 9. *Plan executed by the rover during the mission.*

battery, mission elapsed time, action under execution, plan under execution.

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. In Fig. 7 we show the profile generated by rover simulator for the power generated by SA1, SA2 and SA3. Fig. 8 shows the sun aspect angle related to the generation of the power generation profile in Fig. 7. Fig. 9 shows a graphical representation of the plan.

Fig. 6 shows the ARPHA output assuming such mission profiles: at each mission step, the current sensor data and plan are retrieved and converted into current or future observations for the variables in the onboard model. Such observations are expressed as the probability distribution of the possible variable values. For example, the "wait" action in the plan at the mission step 0, 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 plan action (or the recovery action). In particular, the value 0 corresponds to the "wait" action. An example about sensor data is the sensor pwrsa1 providing the value 17.39931 at the mission step 0 (see Fig. 7). This values 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 time step, in order to represent that the solar array SA1 is generating power in that step (the value 1 represents instead the absence of power generation).

In the ARPHA output in Fig. 6, we notice that at mission steps 0, 1, 2, the diagnosis (by model inference at the current time) detects that the current system state is normal. In other words, none of the scenarios S1, S2, S3, S4 is detected. In the same steps, the prognosis (by model inference at 5 mission steps in the future[4]) still detects the normal future state of the system. At mission time 3, the diagnosis detects the scenario S1, and in particular an anomaly (low level of power generation combined with a not optimal sun aspect angle). As a consequence, the preventive recovery is activated and the policies P1 and P2 are evaluated according to the utility function specified in Fig. 10.b. This function assigns a coefficient (utility value) to all the combinations of the variables *ActionId* and *Balance*. The computation of the utility function returns 0.4736 and 0.0622 for the policies P1 and P2 respectively. Therefore the policy P1 is suggested as the best one to be applied.

6 Conclusions

We have presented ARPHA, a formal architecture for on-board FDIR process for an autonomous spacecraft. ARPHA aims at keeping as much standard as possi-

[4]The number of time steps for the prognosis can be set by the user.

 DOI: 10.2420/AF05.2012.99

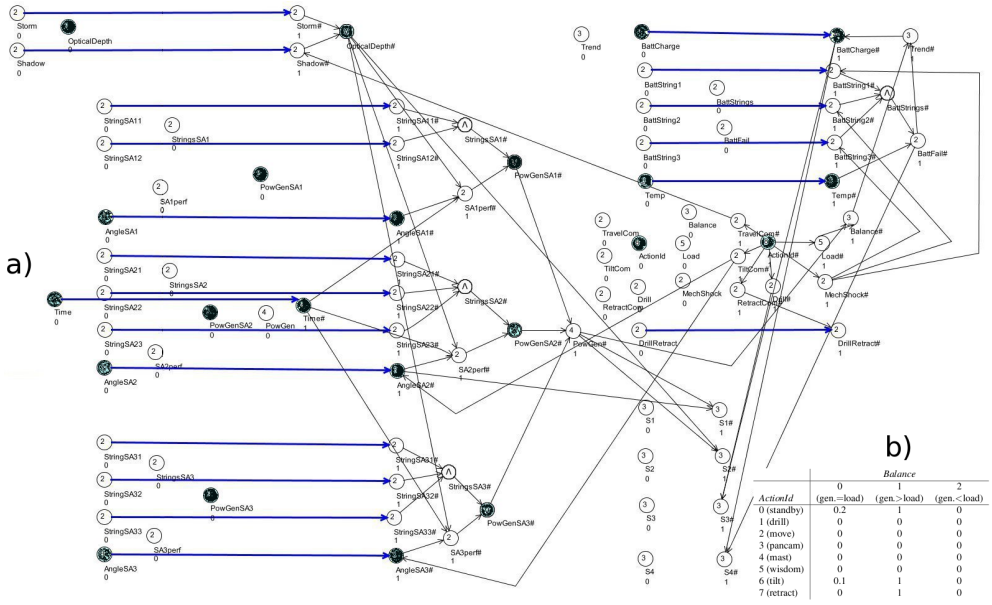

Figure 10. *a) DBN model of the case study. b) Utility function for P1 and P2.*

ble the fault analysis phase, by allowing reliability engineers to build their fault models using an intuitive extension of the DFT language (the EDFT language), being 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, partial observability, system-environment uncertain interactions. ARPHA transforms the EDFT model into an equivalent DDN to be used as the operational model for the FDIR analysis task. On-board analysis exploits JT inference, by transforming the DDN into a corresponding DBN from which to derive the JT structure to be actually used on-board for on-line diagnosis, recovery and prognosis. The formal on-board software architecture of ARPHA has then been presented through UML diagrams. The architecture is currently under validation on a set of case studies concerning the on-board FDIR process applied to a Mars rover. One of such case studies has been briefly presented, showing the innovative capabilities of ARPHA with respect to preventive recovery. ARPHA has been designed as part of the ESA/ESTEC study TEC-SWE/09259/YY called VERIFIM (Verification of Failure Impact by Model checking).

References

[1] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. UAI 1998*, pages 33–42, 1998.

[2] K. Buchacker. Modeling with extended fault trees. In *Proc. IEEE Int. Symp. on High Assurance System Engineering*, Albuquerque,NM, 2000. IEEE Press.

[3] J. Dugan, S. Bavuso, and M. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Trans. on Reliability*, 41:363–377, 1992.

[4] W. Glover, J. Cross, A. Lucas, C. Stecki, and J. Stecki. The use of PHM for autonomous unmanned systems. In *Proc. Conf. PHM Society*, Portland, OR, 2010.

[5] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[6] S. Montani, L. Portinale, A. Bobbio, and D. Codetta-Raiteri. Radyban: a tool for reliability analysis of dynamic fault trees through conversion into dynamic Bayesian networks. *Reliabil-*

ity Engineering and System Safety, 93(7):922–932, 2008.

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

[8] L. Portinale and D. Codetta-Raiteri. Using dynamic decision networks and extended fault trees for autonomous FDIR. In *Proc. Int. Conf. on Tools with Artificial Intelligence*, Boca Raton, FL, 2011.

[9] P. Robinson, M. Shirley, D. Fletcher, R. Alena, D. Duncavage, and C. Lee. Applying model-based reasoning to the FDIR of the command and data handling subsystem of the ISS. In *Proc. iSAIRAS 2003*, Nara, Japan, 2003.

[10] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach (3rd ed.)*. Prentice Hall, 2010.

[11] W. G. Schneeweiss. *The Fault Tree Method*. LiLoLe Verlag, 1999.

[12] M. Schwabacher, M. Feather, and L. Markosian. 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.

 DOI: 10.2420/AF05.2012.99