ARPHA: AN FDIR ARCHITECTURE FOR AUTONOMOUS SPACECRAFTS BASED ON DYNAMIC PROBABILISTIC GRAPHICAL MODELS.

Luigi Portinale and Daniele Codetta-Raiteri

Dipartimento di Informatica, Università del Piemonte Orientale,
Viale T. Michel 11, 15121 Alessandria, Italy
e-mail:{luigi.portinale, dcr}@di.unipmn.it

ABSTRACT

This paper introduces a formal architecture for on-board diagnosis, prognosis and recovery called ARPHA. ARPHA is designed as part of the ESA/ESTEC study called VERIFIM (Verification of Failure Impact by Model checking). 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 actions taking into account imminent failures. We propose to base the inference engine of ARPHA on Dynamic Decision Network (DDN), a class of Probabilistic Graphical Models suitable to reason about system evolution with control actions, over a finite time horizon. The DDN model needed by ARPHA is assumed to be derived from standard dependability modeling exploiting an extension of the Dynamic Fault Tree (DFT) language, called Extended DFT. We finally discuss the software architecture of ARPHA, where on-board FDIR is implemented.

Key words: Fault Diagnosis, Fault Recovery, Prognosis, Probabilistic Graphical Models.

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) 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 pre-compiled look-up tables), that may also be the execution of the saf-ing actions to put the spacecraft into a known safe configuration and transfers 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 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 [1] or to systems requiring heavy human intervention [2], some others by emphasizing the prognostic phase and relying on heuristics techniques to close the FDIR cycle [3]. 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 integrating standard dependability analysis with 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 model-

ing, 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 introduces the DDN (Dynamic Decision Network) model to be used for the actual FDIR analysis of ARPHA; finally, the design and the formal software architecture of ARPHA are then discussed in Sec. 4.

2. MODELING CAUSAL PROBABILISTIC KNOWLEDGE

Modeling probabilistic causal dependencies is one of the main capabilities of Probabilistic Graphical Models (PGM) [4] 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 (Anomaly Resolution and Prognostic Health management for Autonomy) 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 [4, 5, 6] 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) [7] 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 [7], 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) [8]. However, Fault Trees (FT) are limited to model systems with independent binary components (i.e. characterized by the "ok-faulty" dual behavioral modes, failing independently from other components in the system). For this reason, several extensions have been proposed, either to address specific stochastic dependencies as in Dynamic Fault Tree (DFT) [9] or to allow the modeling of "multi-state" components [10, 11], or both [12]. Concerning the off-board process of ARPHA (having the goal of producing a suitable DDN for on-board inference), we have proposed to extend the formalism of DFTs to another formal modeling language called Extended Dynamic Fault Tree (EDFT) [13]; 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 from the input EDFT model, and then using a suitable algorithm for on-line inference to perform the FDIR task. Details about the EDFT formalism and the modeling features can be found in [13].

In the next sections, we provide the details about the resulting DDN model and the components of the ARPHA architecture.

3. 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. 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:

- *Observable nodes*:
  - `Plan` nodes whose values are the possible actions the planner can execute: these nodes are assumed to be always set by the ABB;
  - a decision node `Recovery` whose value are the possible recovery and control actions the autonomous device can execute[1];

[1]For the sake of simplicity, we assume that all possible recoveries

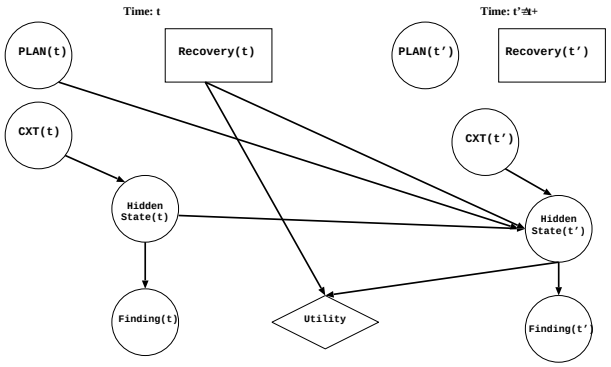

Figure 1. The DDN scheme for ARPHA FDIR.

  - a set of *Sensor Nodes* representing possible measurements from the devices 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.

- *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. 1 and an actual instantiation of this scheme is shown in Fig. 3. 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 actions (plan and recovery) and the current state[2]; the utility function to be optimized, in order to choose the best recovery action, depends on the chosen action and the system state determined by the action.

4. DESIGNING 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. 2 summarizes the basic scheme of the ARPHA on-board reasoning process, involving the interactions with the off-board processing phase.

4.1. The role of the 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 [14]), 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 plan nodes, assumed to be always observed as evidence) and of control actions/policies (that can be mapped into states of the 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. Fig. 3 shows a simplified version of a DDN obtained after a fault analysis and modeling phase, concerning the power management subsystem of an autonomous Mars rover, and in particular, some simplified version of the possible faults and behaviors that may influence the absence of power from rover's bat-

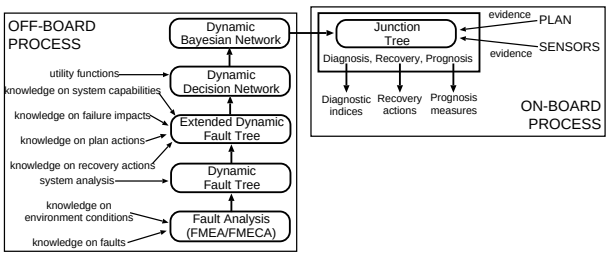

Figure 2. ARPHA on-board reasoning process plus off-board process.

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. In the following we will indicate as recovery action or policy the possible values or states of the decision node *Recovery*.

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

tery. The EDFT originating the DDN in Fig. 3 is available in [13]. Fig. 3 is a modified screenshot of the editor of the Radyban tool [14], highlighting the set of observable parameters; numbers inside the nodes show the node's cardinality, the number below a node is the time slice (0 for the so-called anterior layer and 1 for the so-called ulterior layer), while thick temporal arcs connect node X at slice 0 with its copy $X\#$ at slice 1.

The example captures the following knowledge about the problem. There is no power coming from the rover's battery (`NoPWRBatt`) when either the battery is permanently damaged (`BattDamaged`) or when it is completely discharged (`BattCharge=Flat` see below). Battery damages may occur in case of exposition to either over-temperature or under-temperature, or because of a mechanical shock. The latter has some prior probability of occurrence, that is increased if the rover is executing a `Drill` action; over-temperature and under-temperature are caused by the actual external temperature (which is monitored through a sensor) and by failures of the TCS component (Temperature Control System) that may "fail to keep warm" (`FKW`) or "fail to keep cold" (`FKC`), having in this way three possible states or behavioral modes: `OK, FKW, FKC`. Also Battery charge is discretized on 3 levels: `OK`, `Reduced` and `Flat`. Battery charge level is a sensored parameter (with a possible uncertain reading as any real sensor). Battery charging occurs through power supply from a solar array subsystem composed by a main solar array `SA1` and a warm spare solar array `SA2`. There is no power supply (`NoPWRSA` event true) when either the solar array subsystem is in shadow or when both solar arrays are faulty (`SA` is true). Both shadow and power supply are monitored parameters. The charge of the battery is affected by the operational mode under which the rover is working, namely `Mode=standard`, `Mode=energy_saving`, `Mode=halt`). The operational mode is a controllable parameter, so a recovery action can be executed by setting the suitable state of the node `Mode`. Discharge and charge rates between the battery levels depends on the operating mode (`Mode`) as well as on the presence of power from solar arrays.

In ARPHA, we decided to implement the DDN analysis by resorting to Junction Tree (JT) inference algorithms [4, 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. In particular, since a specific instantiation of the decision node will transform the DDN in a DBN, we implemented two different JT-based algorithms for DBNs as core inference procedure to be used: Murphy's 1.5JT algorithm [6] and Boyen-Koller (BK) algorithm [15]. The first is an exact inference method, while the second is a parametric algorithm that, depending on the input parameters, produces approximated inference with different degrees of accuracy (actually, 1.5JT can be considered as a special case of BK); the main reason for implementing approximated 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]. As mentioned, the implemented inference algorithms use a DBN as the underlying network model; this is because the DDN obtained during off-board analysis is evaluated by considering different setting of the control actions/policies (that are then entered as evidence in the corresponding variables), then transforming the DDN into a DBN. In this way, 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.

4.2. On-board process

The on-board process operates on a Junction Tree as actual operational model, receiving evidence from both sensors (for contextual as well as finding information) and the Autonomy Building Block (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. 2). 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 $d \in D$ given the evidence (from `Plan` and `Sensor Nodes`) up to time t;
- *Recovery* at time t: choice of the "best" action/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. 4 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.

[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.

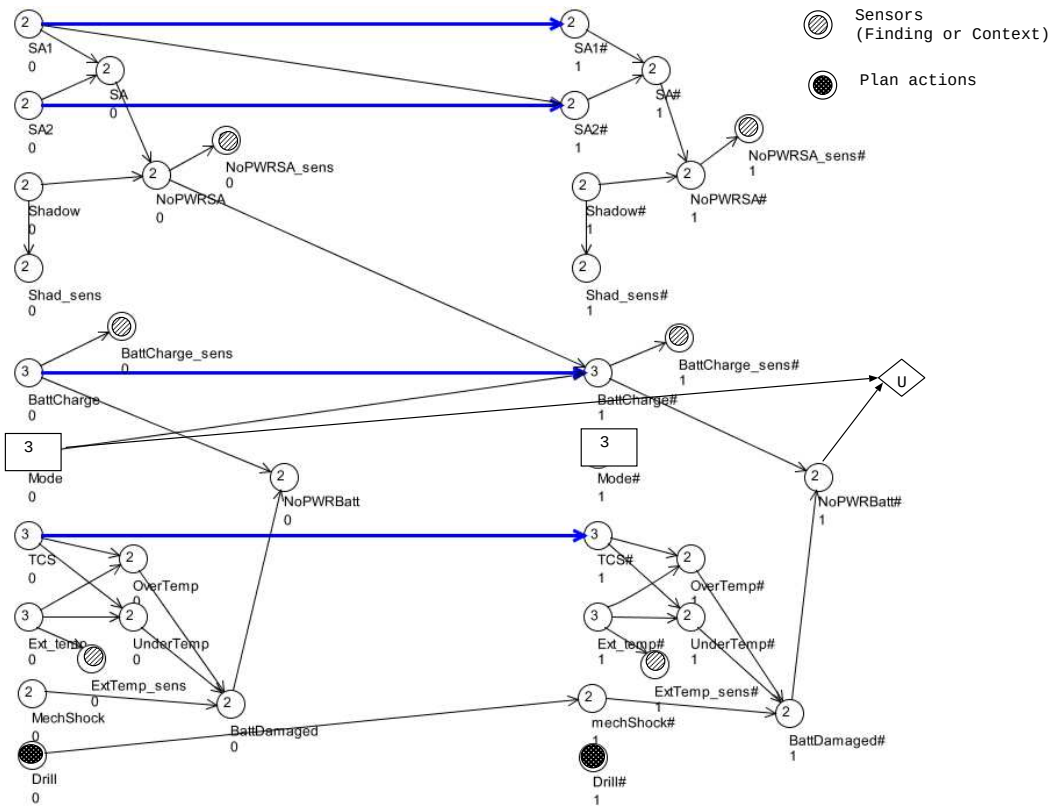

Figure 3. Example of a DDN for ARPHA.

- *Event Handler*: it represents the manager of events receiving from ARPHA the id of the action 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 action (i.e. for each possible state of the recovery node), the action itself is loaded (propagated) into the on-board JT; as mentioned in section 3, the decision node *Recovery* is in general designed in such a way that each value or state of the node actually represents a recovery policy (i.e. a set of atomic recovery actions). Setting this node actually means to set a specific value to the model's variables affected by the policy itself. So, the underlying model on which to perform inference can be assumed to be a DBN (from which to derive the JT as on-board model); the expected utility of each policy is then computed by setting in the JT the corresponding evidence and by propagating it. The action/policy with the maximum expected utility is then determined; such action is converted into a command to be executed by the actuator components, then the command is delivered to the Event Handler for the execution. For example in Fig. 3, the states of the decision node `Mode` are "atomic" policies (composed by a single action) each one setting a specific operational mode for the device. The decision node can then be simply mapped into a chance node having

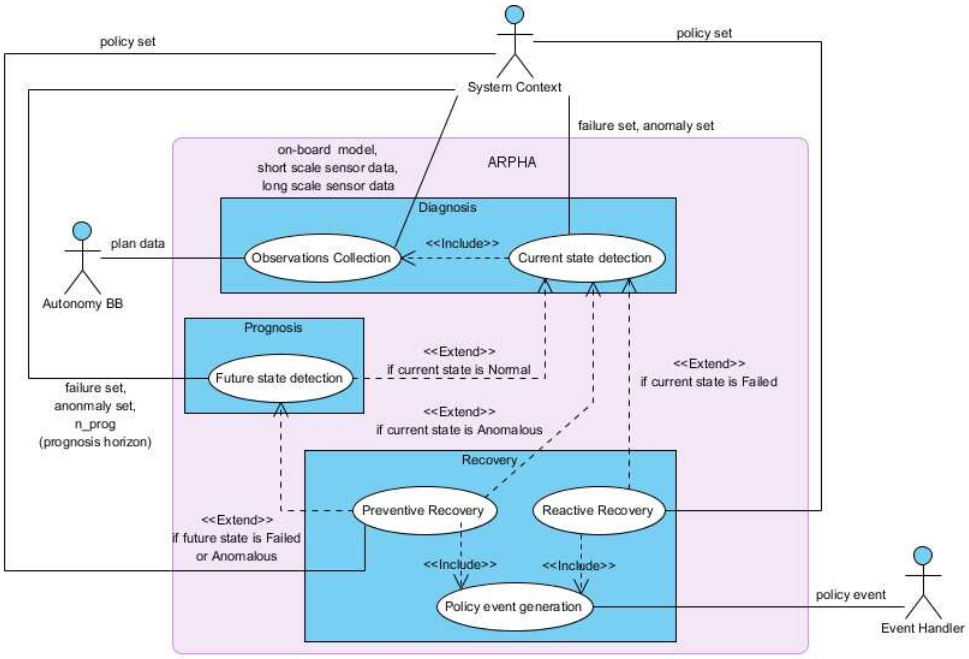

Figure 4. The UML use case diagram of ARPHA.

the operational modes as values/states; they are then considered as evidence in the resulting DBN each time a policy has to be evaluated. The Event Handler will then deliver, as command to be executed, the switching to the operational mode with the maximum expected utility.

- *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 no recovery (i.e. the recovery node set at the current state) and 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 action 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. 5. The results of the execution of each use case are stored in a log file.

The software architecture of ARPHA is composed by the following components represented by the UML class diagram in Fig. 6:

- *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 observations 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 action.
- *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

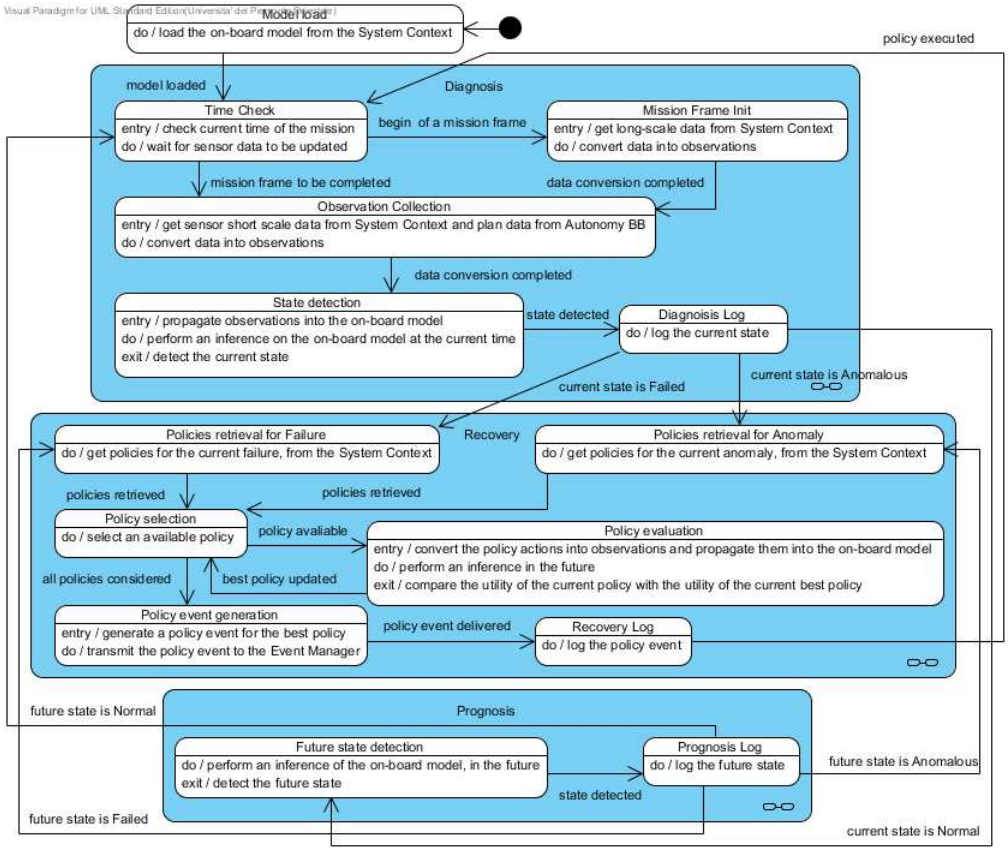

Figure 5. The UML state-chart diagram of ARPHA.

inference algorithm (with the special case of 1.5JT) 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 *Logger* records all the probabilistic computations performed by the *JT_Handler* and can then provide such logs to Ground when requested. The interactions among such components in each use case have also been designed in form of UML sequence diagrams which are reported in [13].

5. 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 possible 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 Junction Tree inference, by compiling the DDN into the JT structure to be actually used on-board; FDIR is then implemented by resorting to standard JT propagation as the core procedure for on-line diagnosis, recovery and prognosis. The formal 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.

ACKNOWLEDGEMENTS

This work has been funded by European Space Agency (ESA/ESTEC) under study TEC-SWE/09259/YY. The study is a joint effort with Thales/Alenia, Torino (Italy). We want to thank Andrea Guiotto of Thales/Alenia for having contributed with his work to the content of the present paper.

REFERENCES

[1] M. Schwabacher, M. Feather, and L. Markosian. Verification and validation of advanced fault detection, isolation and recovery for a NASA space sys-

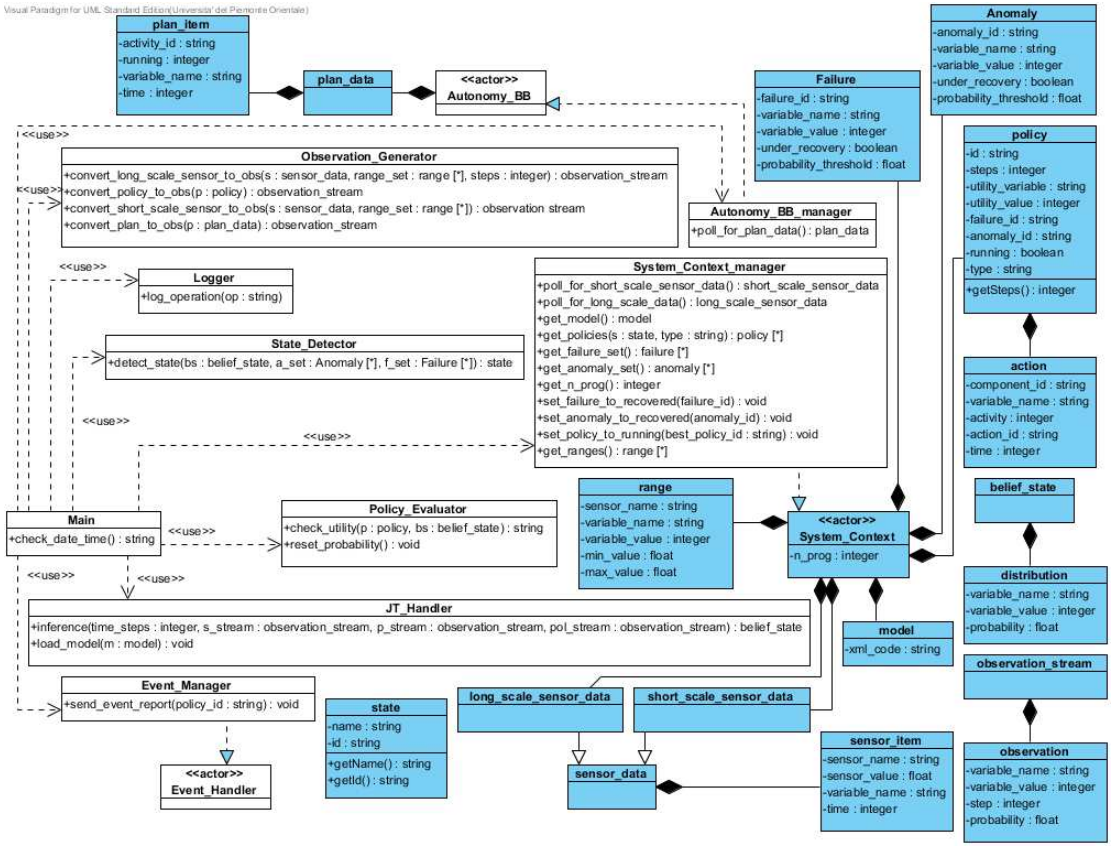

Figure 6. The UML class diagram of ARPHA.

tem. In *Proc. Int. Symp. on Software Reliability Engineering*, Seattle, WA, 2008.

[2] 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.

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

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

[5] F.V. Jensen and T.D. Nielsen. *Bayesian Networks and Decision Graphs (2nd ed.)*. Springer, 2007.

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

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

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

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

[10] Y. Kai. Multistate fault tree analysis. *Reliability Engineering and System Safety*, 28(1):1–7, 1990.

[11] X. Zang, H. Sun, D. Wang, and K.S. Trivedi. A BDD-based algorithm for analysis of mulitstate systems with multistate components. *IEEE Transactions on Computers*, 52(12):1608–1618, 2003.

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

[13] D. Codetta-Raiteri and L. Portinale. ARPHA: an FDIR architecture for Autonomous Spacecrafts based on Dynamic Probabilistic Graphical Models. Technical Report TR-INF-2010-12-04-UNIPMN, Dip. di Informatica, Univ. del Piemonte Orientale, http://www.di.unipmn.it/?page=pubblicazioni&pubid=131, December 2010.

[14] 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. *Reliability Engineering and System Safety*, 93(7):922–932, 2008.

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