Dipartimento di Informatica
Università del Piemonte Orientale "A. Avogadro"
Viale Teresa Michel 11, 15121 Alessandria
http://www.di.unipmn.it

ARPHA: an FDIR architecture for Autonomous Spacecrafts based on Dynamic Probabilistic Graphical Models

D. Codetta Raiteri, L. Portinale (daniele.codetta_raiteri@mfn.unipmn.it, luigi.portinale@mfn.unipmn.it)

TECHNICAL REPORT TR-INF-2010-12-04-UNIPMN
(December 2010)

The University of Piemonte Orientale Department of Computer Science Research Technical Reports are available via WWW at URL `http://www.di.unipmn.it/`. Plain-text abstracts organized by year are available in the directory

Recent Titles from the TR-INF-UNIPMN Technical Report Series

2010-03 *ICCBR 2010 Workshop Proceedings*, C. Marling, June 2010.

2010-02 *Verifying Business Process Compliance by Reasoning about Actions*, D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. Pozzato, D. Theseider Dupré, May 2010.

2010-01 *A Case-based Approach to Business Process Monitoring*, G. Leonardi, S. Montani, March 2010.

2009-09 *Supporting Human Interaction and Human Resources Coordination in Distributed Clinical Guidelines*, A. Bottrighi, G. Molino, S. Montani, P. Terenziani, M. Torchio, December 2009.

2009-08 *Simulating the communication of commands and signals in a distribution grid*, D. Codetta Raiteri, R. Nai, December 2009.

2009-07 *A temporal relational data model for proposals and evaluations of updates*, L. Anselma, A. Bottrighi, S. Montani, P. Terenziani, September 2009.

2009-06 *Performance analysis of partially symmetric SWNs: efficiency characterization through some case studies*, S. Baarir, M. Beccuti, C. Dutheillet, G. Franceschinis, S. Haddad, July 2009.

2009-05 *SAN models of communication scenarios inside the Electrical Power System*, D. Codetta Raiteri, R. Nai, July 2009.

2009-04 *On-line Product Configuration using Fuzzy Retrieval and J2EE Technology*, M. Galandrino, L. Portinale, May 2009.

2009-03 *A GSPN Semantics for Continuous Time Bayesian Networks with Immediate Nodes*, D. Codetta Raiteri, L. Portinale, March 2009.

2009-02 *The TAAROA Project Specification*, C. Anglano, M. Canonico, M. Guazzone, M. Zola, February 2009.

2009-01 *Knowledge-Free Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids*, C. Anglano, M. Canonico, February 2009.

2008-09 *Case-based management of exceptions to business processes: an approach exploiting prototypes*, S. Montani, December 2008.

2008-08 *The ShareGrid Portal: an easy way to submit jobs on computational Grids*, C. Anglano, M. Canonico, M. Guazzone, October 2008.

2008-07 *BuzzChecker: Exploiting the Web to Better Understand Society*, M. Furini, S. Montangero, July 2008.

2008-06 *Low-Memory Adaptive Prefix Coding*, T. Gagie, Y. Nekrich, July 2008.

Contents

ARPHA: an FDIR architecture for Autonomous Spacecrafts based on Dynamic Probabilistic Graphical Models

Daniele Codetta-Raiteri, Luigi Portinale
Dipartimento di Informatica, Università del Piemonte Orientale
Viale T. Michel 11, 15121 Alessandria, Italy
e-mail: {dcr, luigi.portinale}@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 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 show how the model needed by ARPHA can be built through a standard fault analysis phase, finally producing an extended version of a fault tree called EDFT; we discuss how EDFT can be adopted as a formal language to represent the needed FDIR knowledge, that can be compiled into a corresponding Dynamic Decision Network to be used for the analysis. We also discuss the software architecture we are implementing following this approach, where on-board FDIR can be implemented by exploiting on-line inference based on the junction tree approach typical of probabilistic graphical models.

Keywords: Fault Trees, 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, Isolation 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 (that may also be the execution of the safing 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 addressed these problems, some by restricting attention to manned systems [13] or to systems requiring heavy human intervention [10], some others by emphasizing the prognostic phase and relying to 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 should be 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 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 introduces the EDFT formalism to be used for fault analysis, while in Sec. 4 the model to be used for the actual FDIR analysis is discussed; the design and the formal software architecture of ARPHA are then discussed in Sec. 5.

2 Modeling Causal Probabilistic Knowledge

Modeling of probabilistic causal dependencies is one of the main capabilities of Probabilistic Graphical Models (PGM) [7] 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 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 [7, 9] 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) [11] 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 this reason, on-line inference [11] is preferred in the ARPHA architecture. This allow for the choice of a locally 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.

However, dealing directly with the DDN formalism can be a problem for a reliability engineer, usually more familiar with other formalisms and techniques supporting classical FDIR task. Among them Fault Tree Analysis (FTA) [12] is definitely one of the most popular one. 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) [3] or to allow the modeling of "multi-state" components [6, 14], or both [1]. As observed in [1], modeling the system to be analyzed using a set of Boolean variables is often preferable than directly resorting to multi-state variables[1]. In the ARPHA architecture, in order to avoid the burden of introducing a totally new and unfamiliar formalism to the traditional fault analysis phase, we propose a methodology where an extended version of the basic formalism of DFTs is used as a formal modeling language; 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. We call this formalism Extended Dynamic Fault Tree (EDFT) and we propose to use it as a formal notation for reliability engineering during the system modeling phases. 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. We aim at exploiting the power of the EDFT language to express all the knowledge we need for the on-board FDIR engine, while not committing the modeler to learn a totally new modeling language. The ARPHA approach to the analysis of the model is then to compile a DDN from the input EDFT model, using on-line inference

[1]Reliability engineers are often familiar with the use of Boolean gates for modeling a faulty behavior; introducing multi-state variables would require to replace Boolean gates with specific functional gates at the modeling level, producing a relevant impact on the methodology usually adopted to build the model.

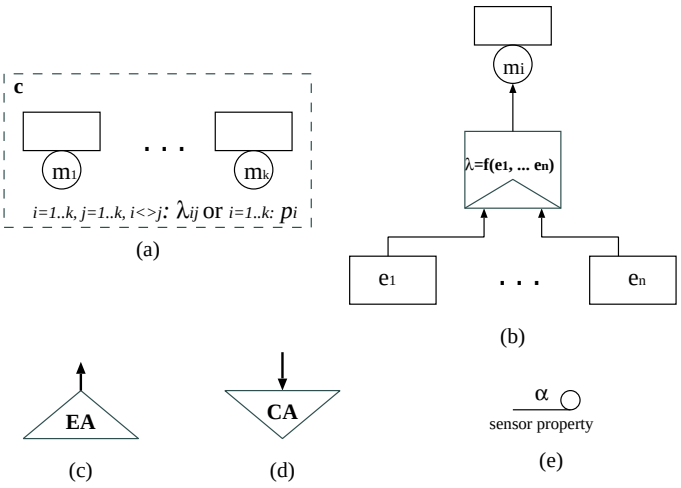

Figure 1: EDFT events and gates.

to perform the FDIR task. In the next sections, we provide the details about the EDFT language and the DDN model that can result from a model built through the formalism.

3 The EDFT Formalism

The EDFT language is an extension to the DFT language as defined in [3] with the following additional constructs[2] (see Fig. 1):

- `Component Box (C-Box)` (Fig. 1(a)): a set of mutually exclusive basic events $m_1, \ldots m_k$ (called *states*) each one associated with either a set of exponentially distributed transition rates $\lambda_{i,j}$ from m_i to m_j ($i \neq j$, $1 \leq i, j \leq k$) or a fixed probability $p_i = Pr\{m_i = \texttt{true}\}$.

- `Stochastic Dependency (SDEP) Gate` (Fig. 1(b)): a gate with events $e_1, \ldots e_n$ as inputs, a basic event m_i belonging to a C-Box c as output[3], a property $\lambda = \lambda_{i,j}$ if $\lambda_{i,j}$ is a transition rate from m_i to m_j (with $m_j \in c$) or $p = p_i$ if p_i is the probability of occurrence of m_i, and a function $f(e_1, \ldots e_m)$. The behavior of the gate is the following: given the configuration $e_1, \ldots e_n$, parameter λ (or p) is set to $f(e_1, \ldots e_m)$; we assume that Boolean value `true` is mapped to 1 and Boolean value `false` is mapped to 0.

- `External Action Event (EA)` (Fig. 1(c)): a special Boolean basic event representing the occurrence of a specific external action having influence on the behavior of the modeled system. EA events have no quantification (i.e. point

[2]We assume the reader familiar with the basic notions concerning FT and DFT.
[3]Of course m_i can also be the special case of a standard Boolean basic event.

probability or rate), since they represent events that are always known to have occurred or not. EA events are assumed to be mutually exclusive (i.e. only one can be set true at a given time instant).

- `Control Action (CA)` (Fig. 1(d)): a special Boolean basic event representing the occurrence of a specific control action issued by the system. CA events have no quantification (i.e. point probability or rate), since they represent events that must be determined to occur or not. CA events are assumed to be mutually exclusive (i.e. only one can be set true at a given time instant).

- `Sensor` (Fig. 1(e)): a property attached to each event or C-Box and set to a value $\alpha \in [0, 1]$;

The C-Box construct is aimed at modeling a system component c having multiple behavioral modes: basic event m_i is true iff component c is in mode m_i. Exactly one m_i is true at a given instant, while all the others are false. In case $k = 2$, a C-Box can be compactly represented as a standard basic event. In such a case, given m the name of the basic event, we can simply denote as λ_m the transition rate from $m =$ `false` to $m =$ `true` (i.e. the failure rate) and as μ_m the viceversa (i.e. the repair rate).

SDEP gates model stochastic dependencies among different events, in particular among system components and between system components and environment. In particular, SDEP gates are aimed at modeling conditional changes in the transition rates (or the probability of occurrence) of a system component mode; in fact, it is trivial to verify that SDEP generalizes every dynamic gate of a DFT, but the Priority AND (PAND) gate (see Fig. 2).

EA events are used in order to model actions that may have influence on the behavior of the modeled system, and that are set externally to the control part of the modeled system. On the other hand, if the system has a control part, some actions can be chosen by the system controller and set to occur (CA events). The difference is that, while EA are "observed events", CA are events that must be determined by the control part of the system. We make the assumption that only one EA is set to occur at a given time instant and that only one CA can be determined at a given time point (see in the following).

The Sensor property of events is aimed at modeling the possibility of gathering observations (evidence) on particular events through sensors; the α value is intended to model the probability of reading the exact state of the C-Box or event to which α is connected[4]; for example, if a C-Box c has states $m_1, \ldots m_k$, then $\alpha = Pr\{\texttt{sensor\_of\_c} = m_i | \texttt{c} = m_i\}$; since $\overline{m_i} = \bigvee_{j \neq i} m_j$, we assume that for each $j \neq i$, $Pr\{\texttt{sensor\_of\_c} = m_j | \texttt{c} = m_i\} = \frac{1-\alpha}{k-1}$ (i.e. if the sensor is wrong, there is an equal probability of reading one of the wrong values). Of course, a sensor property with $\alpha = 1$, means that the corresponding event is directly observable (with no uncertainty), while $\alpha = 0$ means that the corresponding event is "hidden" to any observation; in the latter case the property can be omitted.

[4]This value can be obtained by considering both the *accuracy* of the corresponding sensor (defined as the "ability of a measurement to match the actual value of the quantity being measured") and the level of discretization of the monitored parameter.

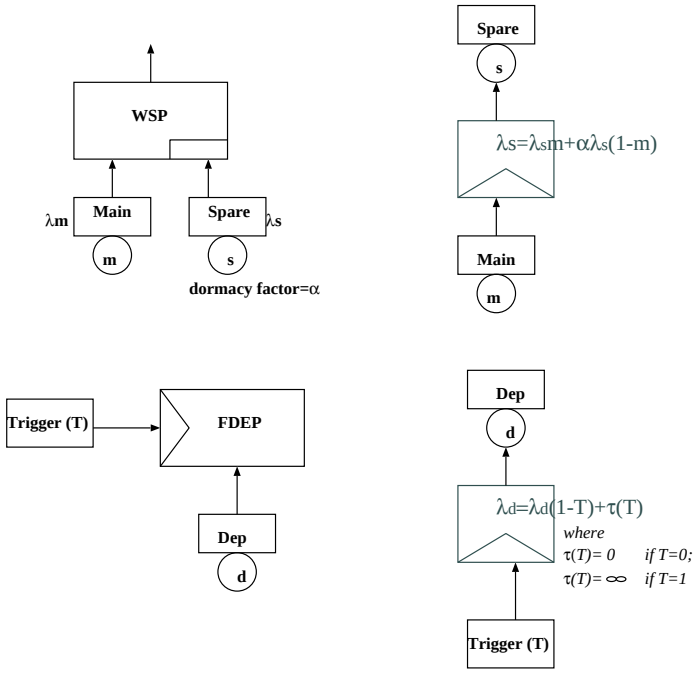

Figure 2: The Warm Spare gate (WSP) and the Functional Dependency gate (FDEP) [3] in form of SDEP gate.

3.1 Example of EDFT Modeling

In order to show the capabilities of the EDFT language, we consider an example which is part of a more complex model which has been developed (and which is still under development) as part of a study called VERIFIM[5]. The example concerns part of 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 battery. The EDFT of Fig. 3 captures the following knowledge about the problem. There is no power coming from the rover's battery when either the battery is permanently damaged or when it is completely discharged (flat). 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 (of a 20% factor) if the rover is executing a "Drilling" action; over-temperature and under-temperature are caused by the actual external temperature (which is monitored through a sensor with a correct reading 95% of the time) and by failures of the TCS component (Temperature Control System) that may "fail to keep warm" (FKW) or "fail to keep cold" (FKC). Battery charge is discretized on 3

[5]VERIFIM is a study conducted by Thales/Alenia and University of Piemonte Orientale, under the funding of ESA/ESTEC, TEC-SWE/09259/YY.

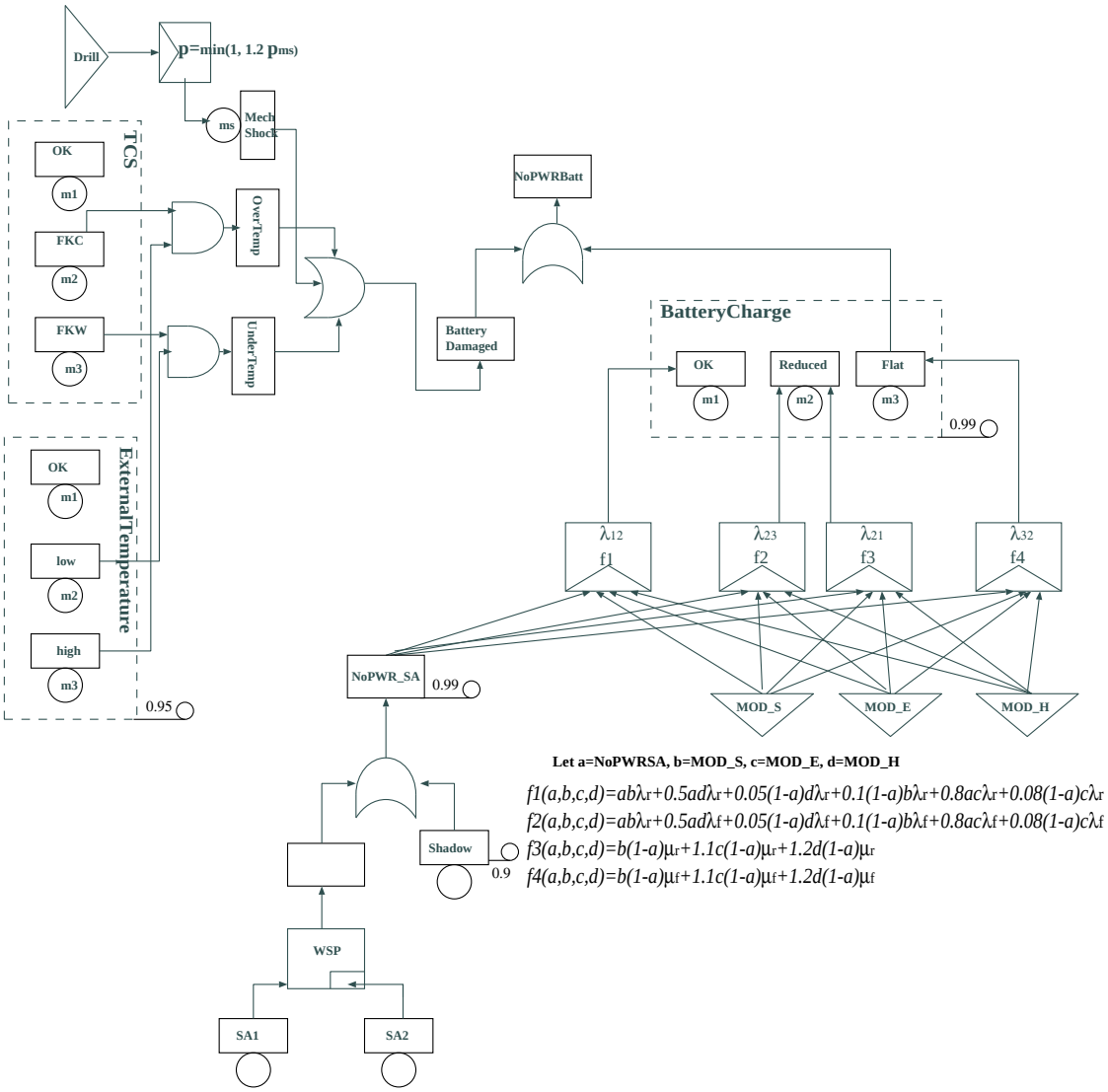

Figure 3: . EDFT for a Mars rover's battery power management.

levels: OK (mode m_1), Reduced (mode m_2) and Flat (mode m_3). It is sensored with a correct reading of 99%. 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 (NoPWR_SA event true) when either the solar array subsystem is in shadow or when both solar arrays are faulty. 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 *standard* (MOD_S), *energy saving* (MOD_E) or *halt* (MOD_H) mode. Basic discharge rates λ_r (from OK to Reduced) and λ_f (from Reduced to Flat) are supposed to be defined in standard mode, with no power supply from SAs; in this situation no recharge is possible, so we have $\lambda_{12} = \lambda_r; \lambda_{23} = \lambda_f; \lambda_{21} = \lambda_{32} = 0$. On the contrary recharge rates μ_r from m_2 to m_1 (Reduced to OK) and μ_f from m_3 to m_2 (Flat to Reduced) refer to powered up battery in standard mode.

Discharge rates are reduced by 20% and by 50% if the operational modes are MOD_E and MOD_H respectively; they are reduced further by 90% in case of power supply form SAs. Recharge rates are increased by 10% and by 20% in MOD_E and MOD_H respectively. All these dependencies are represented by the SDEP gates in Fig. 3 having the modes of Battery Charge as output. Functions f_1, f_2 model the reduction of discharge rates; for example $f_1(a, b, c, d) = 0.5\lambda_r$ if $a = \texttt{NoPWRSA} = 1, b = \texttt{MOD\_S} = 0, c = \texttt{MOD\_E} = 0, d = \texttt{MOD\_H} = 1$, i.e. the rate from OK to Reduced is reduced by 50% when the rover is in "halt mode" with no power supply from SAs. Functions f_3, f_4 model the increment of the repair rates; for example $f_3(a, b, c, d) = 1.1\mu_r$ if $a = 0, b = 0, c = 1, d = 0$, i.e. the rate from Reduced to OK is increased by 10% when the rover is in "energy saving" mode and powered by SAs.

4 A DDN Model 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 assume 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 setting and executing a given plan. We assume the following characterization of DDN nodes:

- *Observable nodes*: a node Plan whose values are the possible actions the planner can execute: this node is assumed to be always set; a decision node Recovery whose value are the possible recovery and control actions the autonomous device can execute; 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*

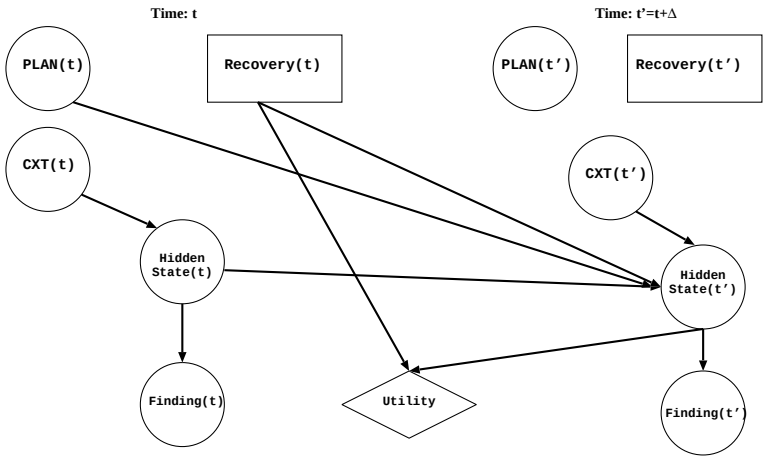

Figure 4: The DDN scheme for ARPHA FDIR.

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. 4; an actual instantiation of this scheme is shown in Fig. 6. The scheme encodes the following general assumptions: 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[6]; 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.

In the next sections we will discuss how a specific instance of the DDN scheme of Fig. 4 can be obtained in the ARPHA architecture.

5 Designing ARPHA

The ARPHA architecture puts emphasis on the on-board software capabilities; however, an off-board processing phase is necessary, in order to provide it with the inputs and the operational model that it needs (Fig. 5).

5.1 Off-board process

The off-board process starts with a fault analysis phase 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

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

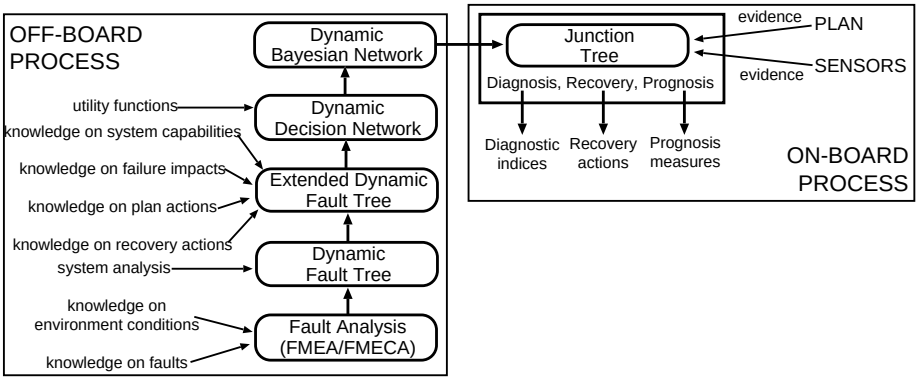

Figure 5: ARPHA on-board reasoning process plus off-board process.

with knowledge about more specific system capabilities and failures, with particular attention to the identification of multi-state components and of stochastic dependencies not captured at the DFT language level. The aim is to generate an EDFT representing all the needed knowledge about failure impacts. For example, events of the DFT representing different modes of the same components are identified and "clustered" in a C-Box, while specific stochastic dependencies among different parts of the modeled system are represented through SDEP gates. 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 [8]), with the addition of the compilation of the SDEP gates (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 the plan node, assumed to be always observed as evidence) and of control actions (that can be mapped into states of the decision nodes). 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. 6 shows the DDN obtained from the EDFT of Fig. 3 The RADYBAN tool [8] can be used in this phase.

In ARPHA, the DDN analysis is actually performed by exploiting Junction Tree (JT) inference. So, another role of the off-board process is the generation of the JT from the DDN. In particular, we decided to adopt Murphy's 1.5JT algorithm for DBN [9] as the core inference procedure. We transform the DDN obtained during off-board analysis, into a corresponding DBN by considering different setting of the control actions. In this way, since the inference procedures will be performed on board, the JT will be the operational model undergoing analysis by the on-board process of ARPHA, with diagnosis, recovery, and prognosis purposes.

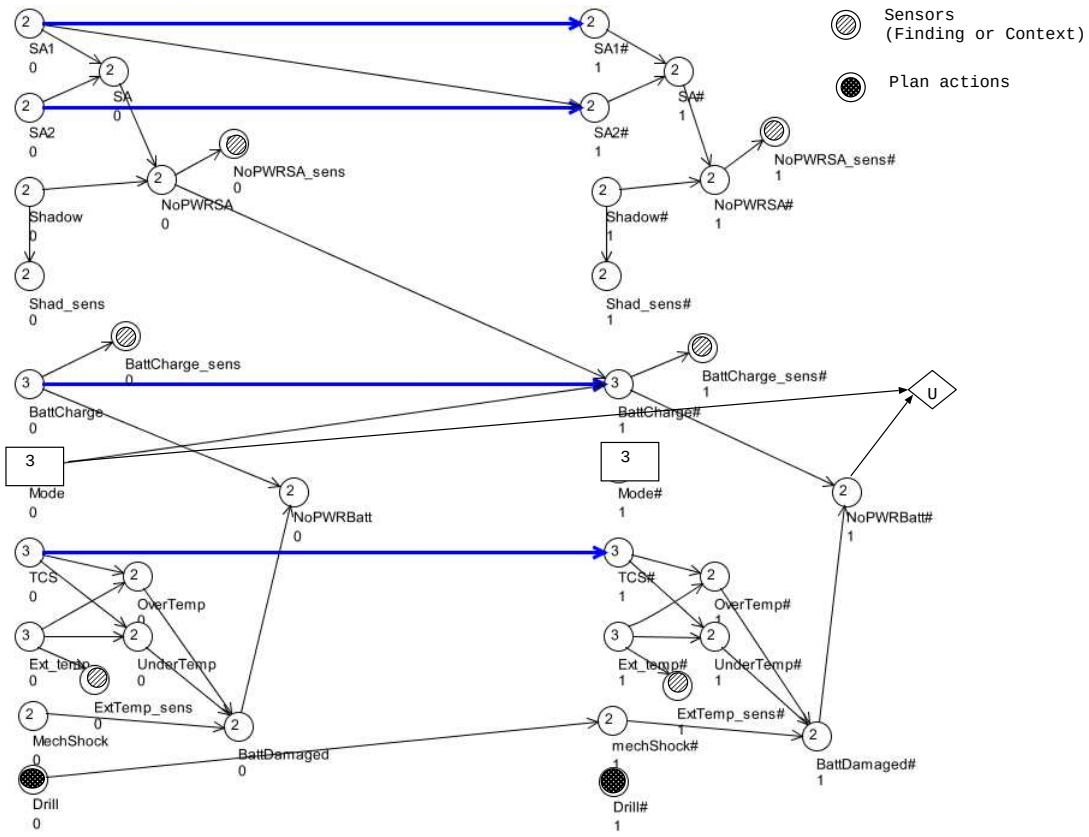

Figure 6: DDN obtained from the EDFT of Fig. 3.

5.2 On-board process

The on-board process resort to JT as actual operational model, receiving evidence from both sensors (for contextual as well as finding information) and an 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 prognosis indices (see Fig. 5). 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 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;

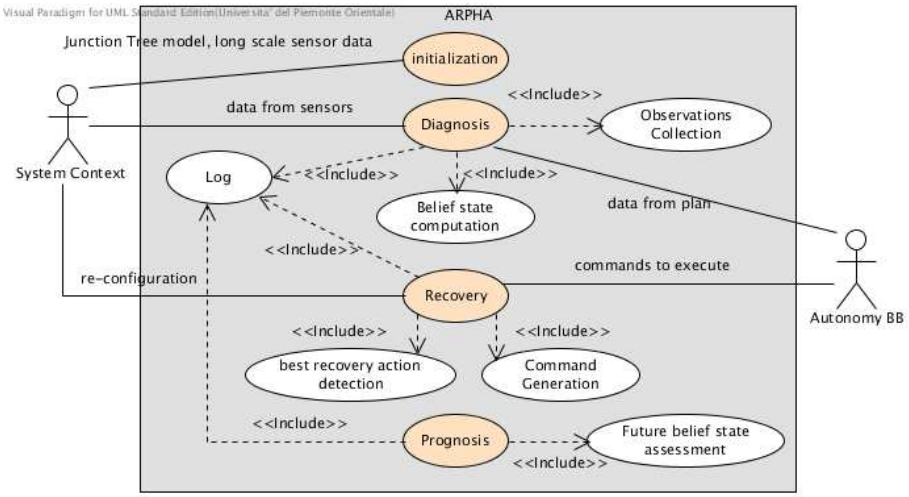

Figure 7: The UML Use case diagram of ARPHA.

- *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. $MF = [t_0, t_0 + \Delta, \ldots, t_f - \Delta, t_f]$.

It is worth noting that, among the possible recovery actions there is also the `no_recovery` action, meaning that no explicit recovery is needed. Furthermore, if prognosis is required for $t' > t + \Delta$, it is assumed that `no_recovery` action will be selected from $t+\Delta$ to t' (i.e. the algorithm can predict the future state of the system given the current best action and given that other explicit recovery actions will not be performed).

The UML use case diagram in Fig. 7 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.

ARPHA cyclically performs the following sequence of use cases:

- *Initialization*: it periodically retrieves data necessary for on-board reasoning. More specifically, ARPHA periodically checks the current mission time: if the first day of 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, still from the System Context. In particular, long scale sensor data are converted into observations

(evidence) for the on-board model; then, observations are propagated into the on-board model.

- *Diagnosis*: sensor data and plan data are retrieved from the System Context and the Autonomy BB respectively. Then, both kinds of data are converted in form of observations concerning the variables of the on-board model; such observations are used to update the JT on-board model and inference is executed by 1.5JT propagation [9]. Inspection of the probabilities of the diagnostic variables can provide the diagnosis at the current mission time.

- *Recovery*: after having incorporated the current evidence in the diagnostic phase, for each available recovery action, the action itself is propagated into the JT on-board model, and the expected utility of the action is computed. The action (possibly the `no_recovery` action) with the maximum expected utility is then determined; such action is converted into a command to be executed by the actuator components managed by the Autonomy BB. So, the command is delivered to the Autonomy BB for the execution.

- *Prognosis*: the time horizon t' for prognosis is determined and 1.5JT inference is performed with a time step of Δ until t', by considering `no_recovery` action and plan information at each time step as evidence.

The operations performed inside each use case are represented by the UML statechart diagram in Fig. 8. The results of the execution of each use case are stored in a log file.

5.3 ARPHA architecture

The architecture of ARPHA is composed by the following components represented by the UML class diagram in Fig. 9:

- *Main*: it implements the main program capabilities and manages the other components;

- *Observation_Generator*: it retrieves sensor data and plan data from the System_Context_Manager and the Autonomy_BB_Manager respectively. It converts both kinds of data into observations to be propagated into the on-board model;

- *Command_Generator*: it implements the conversion of recovery actions detected by the Recovery phase, into commands executable by the Autonomy BB;

- *JT_Configurator*: it implements propagation of observations and actions into the on-board model;

- *JT_Analyzer*: it computes the expected utility and gives the current or future belief state;

- *Logger*: it implements the logger capabilities;

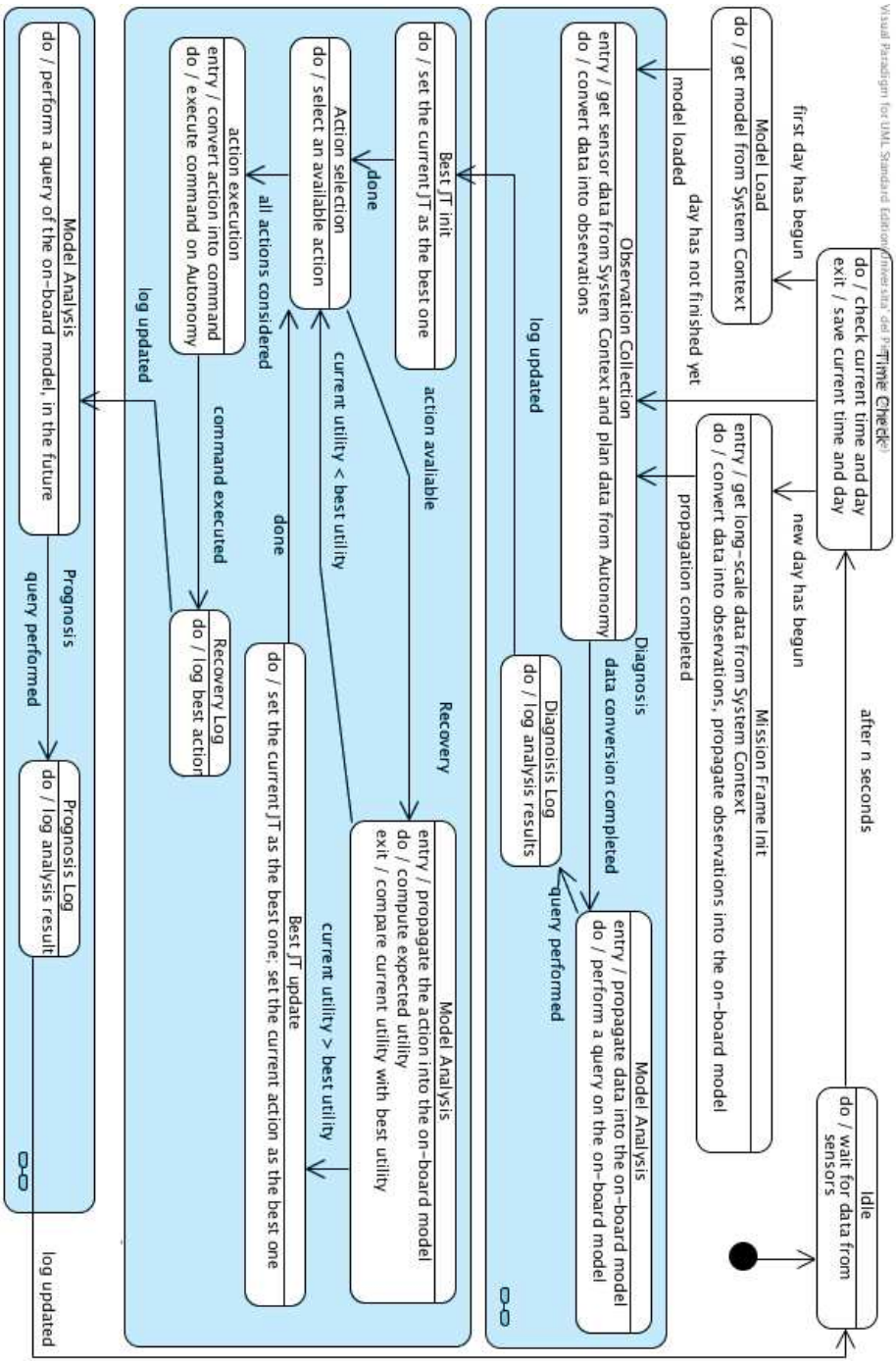

Figure 8: The UML State-chart diagram of ARPHA.

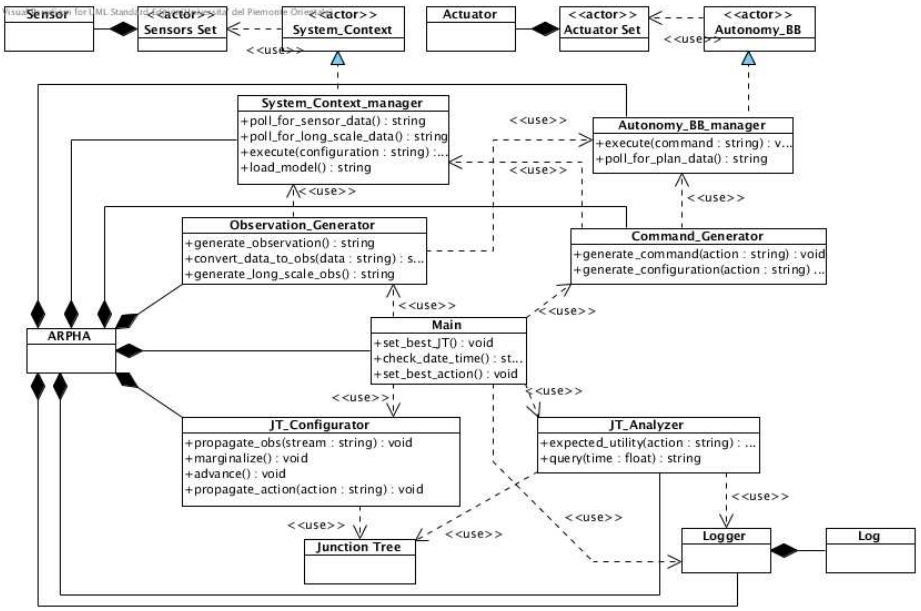

Figure 9: The UML class diagram of ARPHA.

- *System_Context_Manager*: it implements functions dedicated to manage data contained in System Context;
- *Autonomy_BB_Manager*: it implements functions dedicated to interface the Autonomy BB.

The UML sequence diagram in Fig. 10 represents the cyclic performance of the sequence composed by the Initialization, Diagnosis, Recovery, and Prognosis use case. The components involved in each use case and their interactions in order to realize the use case, are shown still in form of UML sequence diagram, in Fig. 11 (Initialization), Fig. 12 (Diagnosis), Fig. 13 (Recovery) and Fig. 14 (Prognosis). The Main component participates to each use case and coordinates the other components.

6 Conclusions

We have presented the ARPHA 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

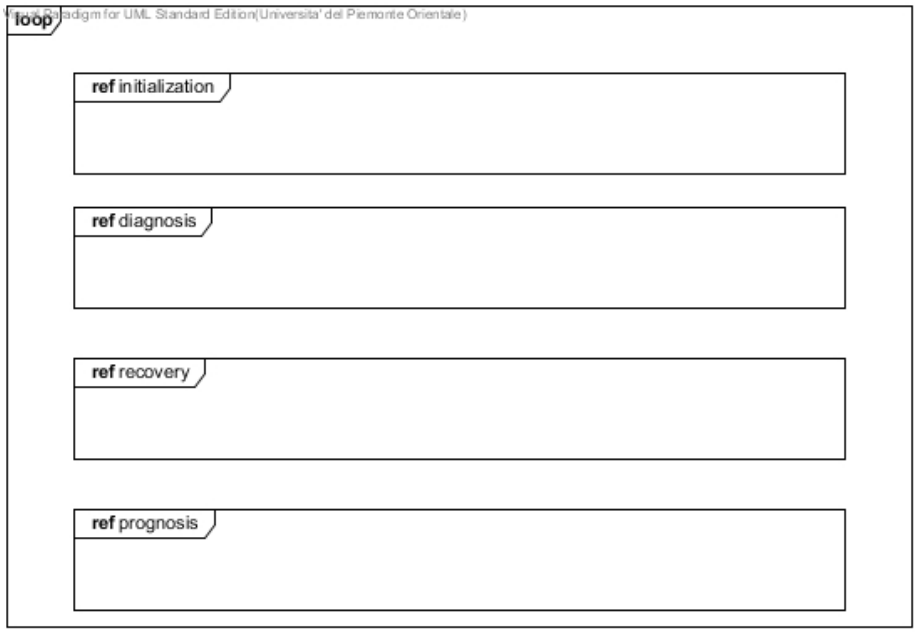

Figure 10: The main UML sequence diagram of ARPHA.

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.

References

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

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

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

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

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

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

[7] Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)

[8] Montani, S., Portinale, L., Bobbio, A., Codetta-Raiteri, D.: 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)

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

[10] 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)

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

[12] Schneeweiss, W.G.: The Fault Tree Method. LiLoLe Verlag (1999)

[13] 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)

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

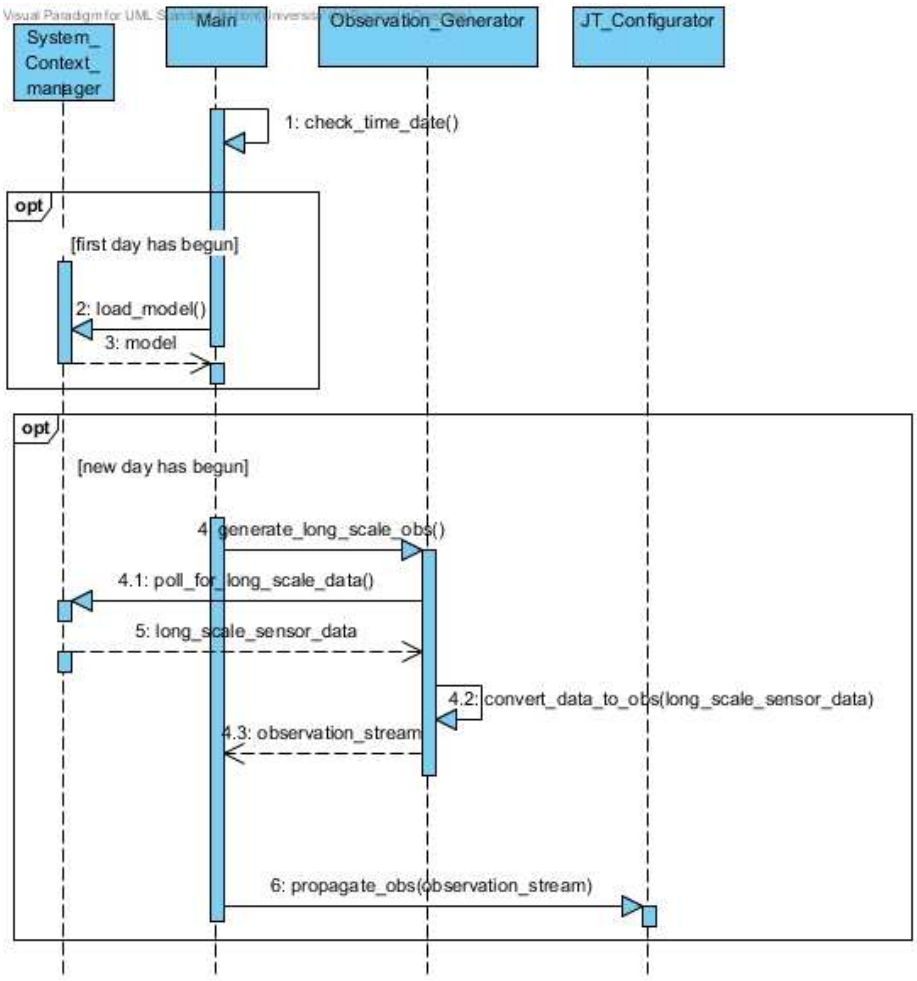

Figure 11: The UML sequence diagram of the Initialization use case.

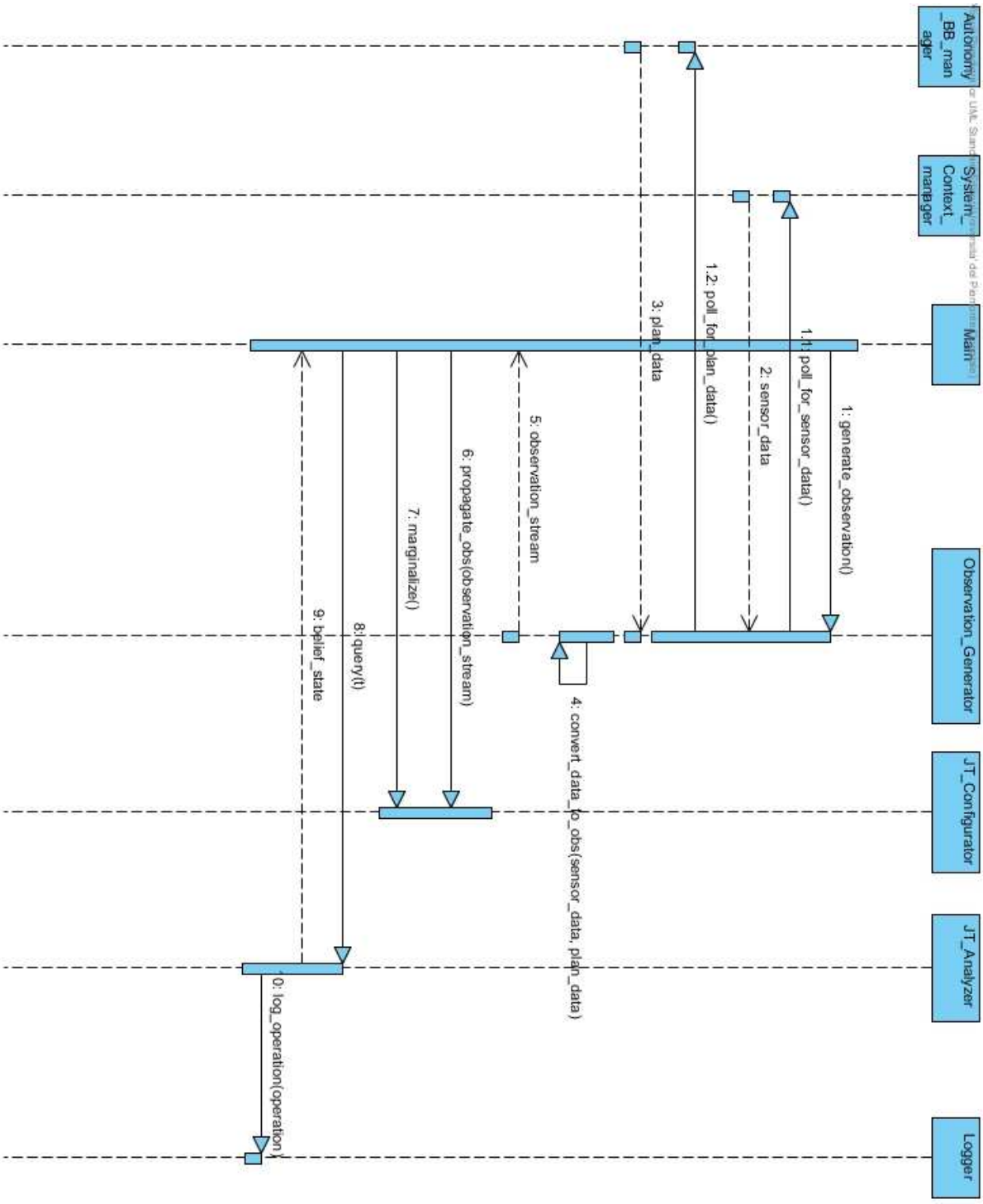

Figure 12: The UML sequence diagram of the Diagnosis use case.

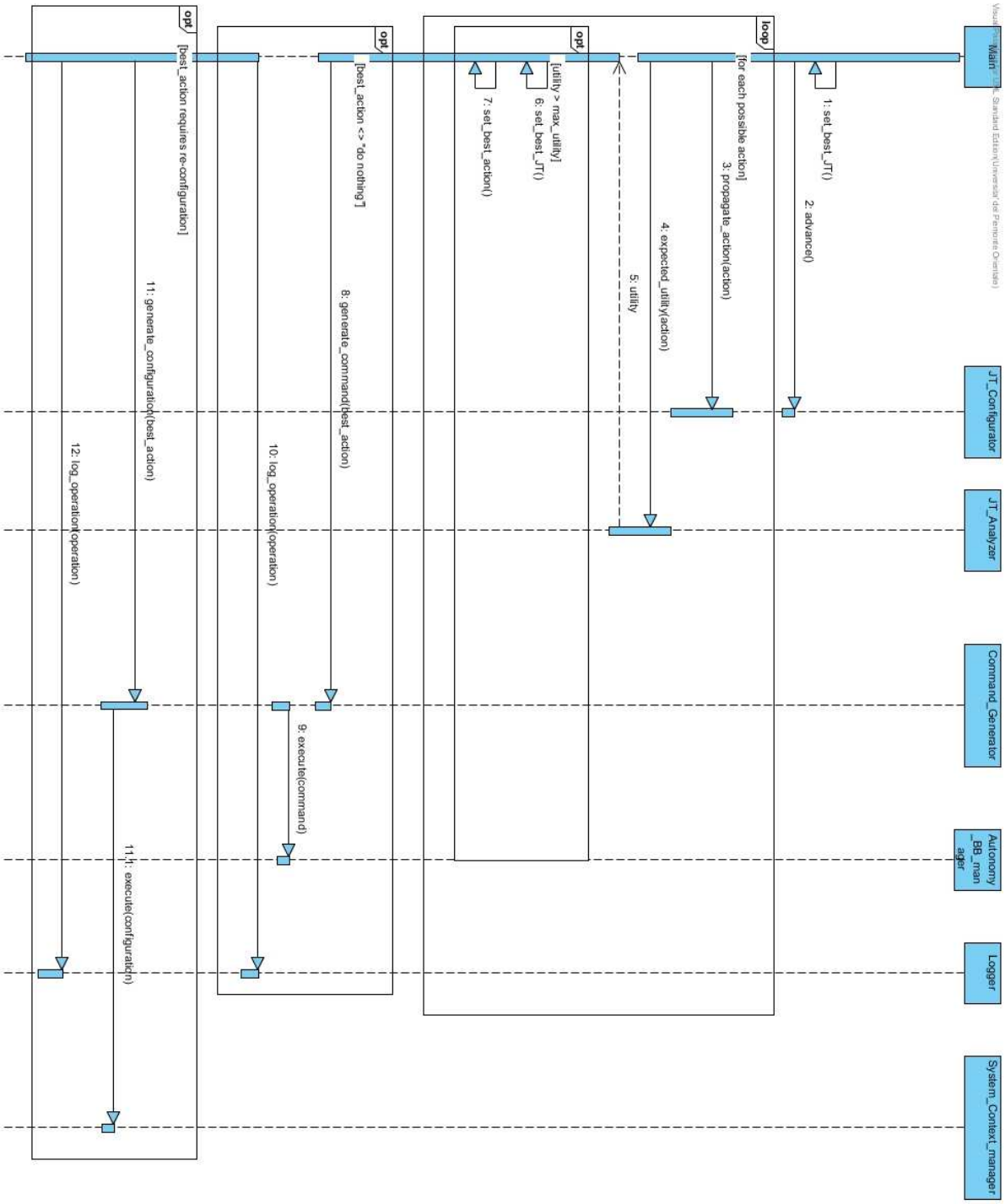

Figure 13: The UML sequence diagram of the Recovery use case.

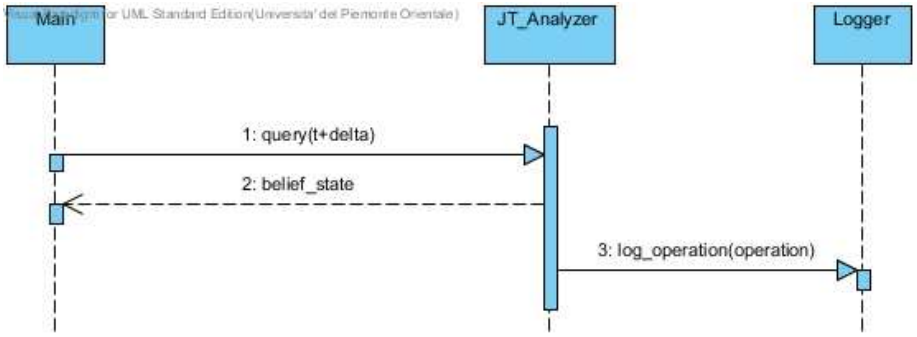

Figure 14: The UML sequence diagram of the Prognosis use case.