THE UNIVERSITY OF HULL

Distributed On-line Safety Monitor Based on Safety Assessment Model and Multi-agent System

being a Thesis submitted for the Degree of

Doctor of Philosophy

in the University of Hull

by

Amer Abdaladeem Dheedan

May 2012

To my mother,,,,,

And

To the memory of my father,,,,,

I dedicate this thesis.

Abstract

On-line safety monitoring, i.e. the tasks of fault detection and diagnosis, alarm annunciation, and fault controlling, is essential in the operational phase of critical systems. Over the last 30 years, considerable work in this area has resulted in approaches that exploit models of the normal operational behaviour and failure of a system. Typically, these models incorporate on-line knowledge of the monitored system and enable qualitative and quantitative reasoning about the symptoms, causes and possible effects of faults.

Recently, monitors that exploit knowledge derived from the application of off-line safety assessment techniques have been proposed. The motivation for that work has been the observation that, in current practice, vast amounts of knowledge derived from off-line safety assessments cease to be useful following the certification and deployment of a system. The concept is potentially very useful. However, the monitors that have been proposed so far are limited in their potential because they are monolithic and centralised, and therefore, have limited applicability in systems that have a distributed nature and incorporate large numbers of components that interact collaboratively in dynamic cooperative structures. On the other hand, recent work on multi-agent systems shows that the distributed reasoning paradigm could cope with the nature of such systems.

This thesis proposes a distributed on-line safety monitor which combines the benefits of using knowledge derived from off-line safety assessments with the benefits of the distributed reasoning of the multi-agent system. The monitor consists of a multi-agent system incorporating a number of Belief-Desire-Intention (BDI) agents which operate on a distributed monitoring model that contains reference knowledge derived from off-line safety assessments. Guided by the monitoring model, agents are hierarchically deployed to observe the operational conditions across various levels of the hierarchy of the monitored system and work collaboratively to integrate and deliver safety monitoring tasks. These tasks include detection of parameter deviations, diagnosis of underlying causes, alarm annunciation and application of fault corrective measures. In order to avoid alarm avalanches and latent misleading alarms, the monitor optimises alarm annunciation by suppressing unimportant and false alarms, filtering spurious sensory measurements and incorporating helpful alarm information that is announced at the correct time.

The thesis discusses the relevant literature, describes the structure and algorithms of the proposed monitor, and through experiments, it shows the benefits of the monitor which range from increasing the composability, extensibility and flexibility of on-line safety monitoring to ultimately developing an effective and cost-effective monitor. The approach is evaluated in two case studies and in the light of the results the thesis discusses and concludes both limitations and relative merits compared to earlier safety monitoring concepts.

Acknowledgments

To me the PhD sounded like a mission whose achievement would prove my own capabilities. As the mission is at its end, I admit that without support from many people the mission could not have been achieved.

First and foremost, I am sincerely and heartily grateful to my supervisor Professor Yiannis Papadopoulos. He not only gave me the opportunity to conduct this mission, he further supported me with his valuable guidance, constant patience and encouragement along the way. Without him the achievement could never have come this far.

Thanks to Dr. Darryl Davis and Dr. Chandra Kambhampati for their insightful advice which contributed to improve the overall project.

I also owe Dr. Hassan Al-Qadhi, the president of Delmon University in the Kingdom of Bahrain, a great debt of gratitude for encouraging and supporting me during my study years.

My great sisters, Iqbal, Intidhar and Kawthar, have overwhelmed me with their affection. They were constantly beside me and have boosted me morally and supported me in my weakest and loneliest moments. Likewise, my brothers; Samer, Dheedan, Badran and Omran; no words are adequate me to thank them, but I will be forever thankful for their support.

I am also deeply indebted to Haider Al-Badri and Anwar Al-Badri. When I first came to the UK they hosted me and they voluntarily took the responsibility to ask about my study and my situation and supported me to the best of their capacity. I have been really lucky to be their friend and I owe my earnest thanks to them for all they have done.

Throughout my study years I have met fellows whose friendship is always in a source of pride and appreciation. Among them are some who have finished the process: Dr. Martin Walker, Dr. David Parker, Dr. Ian Wolforth and Dr. Septavera Sharvia, and other who are still in the process: Nabil Abu Hashesh, NaMian Zhi Bao and Ernest Edifor. Special thanks to my fellow and friend Shawulu Nggada whose truthful experience was constantly accessible to me to extend my knowledge.

I should also extend my gratitude to people who work behind the scenes to maintain the smooth running of the department of computer science. Among those people are the administrator; Amanda Millson, and the secretaries; Helen El-Sharkawy, Colleen Nicholson, Joan Hopper and Jo Clappison. Equal gratitude is due to: Mark Bell, Mike Bielby, David Glover, Adam Hird and Simon Grey, for their provision of support services.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of this mission.

Contents

Chapter One
Introduction

Chapter Two
Background

Chapter Three
Distributed On-line Safety Monitor: an Overview and Distributed Monitoring Model

List of Figures

List of Tables

Chapter One

Introduction

1.1 Field of Research and Problem Context

Safety critical systems are a class of engineered systems the failure of which potentially causes hazards for people and the environment; for example, nuclear power plants, chemical engineering processes, transportation systems, and spacecraft. The safety of such systems must thus be delivered at a high level of assurance, demonstrated by minimum likelihood of disastrous failure scenarios (Grunske and Strabe, 2003).

With the aim of developing the safest possible critical systems, it has become common practice for a rigorous off-line safety assessment process to be applied during the development life cycle of such systems. Safety assessment processes, in particular model-based safety assessment, have been widely and deeply researched and hence a range of different automated assessment techniques have emerged (Joshi *et al.*, 2006). Invariably, these techniques are based on a process of examining potential hazards and iteratively improving the design of the proposed system by removing, containing and mitigating causes of hazards in the architecture of the system, and retrofitting and affixing whenever necessary fault-tolerant means and/or highly reliable components (Du *et al.*, 2010). The goal of the safety assessment process is to produce a verifiably safe design for the system and evidence that the system meets its safety requirements. In addition to their role in the improvement of a system, the results of safety assessment are also used to form the system safety case, an official document produced at the end of the process to certify the safe deployment of the critical system.

From the point of view of this thesis, it is important to note that together the design models and safety assessment model produced in the course of off-line safety assessment contain substantial knowledge about the normal and abnormal operational behaviour of the monitored system. This knowledge is often produced in electronic form with the aid of computerised techniques and tools. Surprisingly, however, despite the effort and cost put into off-line safety assessments and the availability of results, in the current industrial practice this knowledge mostly ceases to be useful after certification. This is clearly at odds with the practical need to ensure safety during operation via safety monitoring - *an umbrella term used in this thesis to describe collectively real-time detection, diagnosis, alarm annunciation and fault controlling*.

Despite this apparent gap between off-line assessment and on-line monitoring, on-line safety monitoring tasks have been extensively researched and a variety of solutions developed over the years. Detection, diagnosis and alarm annunciation typically help in raising the situational

awareness of human operators in the presence of complex faults. Fault controlling tasks support operators by automating some of their fault controlling responsibilities.

In their early appearance, on-line safety monitoring techniques relied on simple monitoring instruments, such as analogous gauges and alarm lights to communicate the operational conditions of the monitored system, and fuses or circuit breakers that responded to abnormal deviation from operational parameters. With such instruments, system operators were mainly responsible for analysing plant conditions and deciding on applying appropriate fault controlling actions. These simple safety monitoring systems are challenged by the increasing complexity of technology and systems. Operators of such systems were unable, particularly in emergency conditions, to analyse and understand the vast amount of communicated measurements and alarms raised by the system. This in turn increased the likelihood of inappropriate responses to faults. To address this problem, in the late 1970s and early 1980s, a large amount of effort was devoted to developing on-line monitoring instruments that could support the role of the operators. Accordingly, more advanced and effective monitoring instruments, such as hardwired controllers and informative alarm annunciation panels, were introduced (Kim, 1992).

Despite these developments, serious operational hazards and failures of safety critical systems continued to be recorded. The accident of the Three Mile Island Unit 2 (TMI-2) nuclear power plant in 1979 is a prominent case of such failure (U.S.NRC, 2008). In this case, a core meltdown in Unit 2 resulted in the release of approximately 2.5 million curies of radioactive gases, and approximately 15 curies of iodine-131. The accident began with failures in the non-nuclear secondary system, followed by a stuck-open pilot-operated relief valve in the primary system, which allowed large amounts of nuclear reactor coolant to escape. Investigation showed that mechanical failures were compounded by the initial failure of plant operators to recognise the situation as a loss-of-coolant accident. The accident has been attributed to two main factors; an ineffective alarm annunciation and the lack of automated fault controlling (Chambers, 2005). While the former overwhelmed the operators with an avalanche of alarms[1] without highlighting the vital alarms or pointing out the underlying causes in the early stages of the disturbance, the latter additionally demanded and relied on the operators to remember prompt controlling actions during emergency and confusing conditions. Due to the surrounding confusion, operators decided incorrectly and performed inappropriate corrective actions that exacerbated the situation and resulted in that disaster.

Learning from the TMI-2 lesson, in the early 1980s development efforts were directed towards developing more advanced computer-based monitors. Accordingly, computerised alarm

[1] Alarm avalanche is a term describes the case in which the number of the released alarms on a single fault is larger than the average that could be processed efficiently and effectively by one operator (Aizpurua *et al.*, 2009).

annunciation and conditions display schemes as well as automated fault controllers were developed (Billings, 1991; Kim, 1992). Hence, on-line monitors started to appear as systems installed in plant control rooms and flight decks of aircraft. Since then, such monitors have demonstrated their merits and contributed significantly to improving the on-line safety of the monitored processes.

However, despite these improvements, still there have been numerous instances of accidents that could have been averted with better safety monitoring. The explosion and fire at the Texaco Milford Haven refinery in 1994, for instance, was attributed to the poor presentation of the alarms as well as inadequate operator training for dealing with a stressful and sustained plant upset (HSE, 2000). The Kegworth Air disaster occurred in 1989 because of (a) ineffective alarm annunciation – delay in alerting the crew pilot of the occurrence of the fault and its underlying causes; (b) lack of automated fault controlling (Trimble, 1990). Recently, monitoring problems contributed to a fatal accident to Air France flight AF447, in which an Airbus A330 crashed in the Atlantic on 1st of June 2009 and all 228 people on board were killed. The technical investigation partly attributed the accident to misleading alarm annunciation and the absence of clear guidance on the emergency conditions, which fell beyond the skills and training of the pilot and co-pilot (BEA, 2011).

Such accidents have motivated and still motivate considerable work on safety monitoring, including the work reported in this thesis.

1.2 Research Motivation and Scope of Research

A large part of the research on safety monitoring has looked into the development of model-based monitors which can employ knowledge about the normal operational behaviour and failure of a system contained in a system model. In the context of this work, models such as state-machines (Papadopoulos, 2003; Eo *et al.*, 2000 and 2001), goal trees (Modarres and Cheon, 1999; Larsson 1994), signed direct graph (Lu and Wang, 2007; Dong *et al.*, 2010) and fault trees (Papadopoulos, 2002; Peng *et al.*, 2007; Felkel *et al.*, 1978) have been exploited and demonstrated their benefits as reference knowledge for system monitoring. Typically, these models incorporate deep knowledge of the monitored system and enable qualitative and quantitative (often probabilistic) reasoning about behavioural transitions, symptoms, causes and possible effects of faults.

Recently, a monolithic safety monitor (Papadopoulos, 2003) that exploits knowledge derived from the application of a semi-automated off-line safety assessment method and tool, called Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) (Papadopoulos *et al.*, 2001), has been proposed. That knowledge is composed of two elements:

- A hierarchy of state-machines describing the behaviour of the system, effectively capturing the normal and abnormal behavioural transitions of the system and its sub-systems.
- A number of fault trees, which effectively represent diagnostic models that relate the symptoms of failure to their immediate and ultimate underlying causes.

A variant of this safety monitor which maintains fault trees in the model but replaces state machines with a control chart of the monitored process has been proposed by Peng *et al*. (2007).

The motivation for the above work has been the observation that, in current industrial practice, vast amounts of knowledge derived in off-line safety assessments cease to be useful following the certification and deployment of a system. A key contribution of this work is that it brings this knowledge forward to the operational phase of a system and usefully exploits it for the purposes of on-line safety monitoring. The concept is potentially very useful. However, one of the difficulties with the monitors proposed in Papadopoulos (2003) and Peng *et al*. (2007) is that they are based on a monolithic concept in which all monitoring of a plant is delegated to a single object or device. This does not align well with the distributed nature of most modern systems. Systems are typically implemented as a set of sub-systems which exist in a complex cooperative structure and coordinate to accomplish system functions. Systems are also typically large and complex and show dynamic behaviour that includes complex mode and state transitions. As a result, such systems need a distributed mechanism for safety monitoring; first it is essential to minimise the time of on-line failure detection, diagnosis and hazard control; second, a distributed monitoring scheme can help focus and rationalise the monitoring process and cope with complexity.

Recent work on multi-agent systems shows that the distributed reasoning paradigm could cope with the nature of such systems. For example Mendes *et al.* (2009) and Ren *et al.* (2006) exploited multi-agent systems to increase the capacity of a diagnostic scheme of large-scale, dynamic systems. Multi-agent systems have also demonstrated prompt responses in detecting faults and diagnosing the underlying causes of the failures of complex distributed chemical processes (Ng and Srinivan, 2010). These experiments show that multi-agent systems have the potential to address some of the issues arising in otherwise potentially very useful model-based safety monitors, in particular those monitors that exploit models from safety assessment.

This thesis specifically looks into the synthesis of these two strands of work on model-based monitoring using safety assessment models and multi-agent systems. The thesis develops a *distributed on-line safety monitor*, which combines the benefits of using knowledge derived from off-line safety assessments with the benefits of the distributed reasoning of multi-agent systems.

The monitor consists of a number of agents, i.e. *multi-agent system*, and a *distributed monitoring model* that contains reference knowledge derived from off-line safety assessments. Guided by the knowledge contained in the monitoring model and real-time observations of the monitored system, monitoring agents are hierarchically deployed and work collaboratively to integrate and deliver safety monitoring tasks, both locally at the sub-system levels and globally overseeing the overall behaviour of the system.

By exploiting their portions of the monitoring model, agents reason on the operational parameters of the monitored system, to detect and assess the effects of deviations, diagnose the underlying causes of the detected deviations and automatically apply corresponding fault controlling measures. Moreover, in order to avoid alarm avalanches and latent alarms[1] that may mislead the system operators (Hwang *et al.*, 2008), agents are also able to optimise alarm annunciation by (a) suppressing unimportant and false alarms; (b) filtering and validating sensory measurements and diagnosing and controlling sensory failures; (c) providing helpful alarm information, such as assessment of the operational conditions after the occurrence of the fault, guidance on controlling the occurred fault, prognosis of the future effects of the occurred fault and diagnostics of the underlying causes of failures.

1.3 Research Hypothesis

The thesis argues that a novel distributed on-line safety monitor that operates on a model containing knowledge derived from safety assessments could:

a- deliver a wide range of effective monitoring tasks which extend from fault detection and diagnosis to effective alarm annunciation and fault controlling.
b- address the limitations associated with earlier safety monitors and deliver improvements in the effectiveness, timeliness, correctness, flexibility and scalability of on-line safety monitoring tasks.
c- have a generic architecture that would make it applicable in a variety of safety-critical systems and contexts, such as those used in numerous transport industries or industrial processes.

1.4 Research Objectives

To investigate the validity of the above research hypothesis, the thesis systematically researches, develops and evaluates a distributed on-line safety monitor in three major steps, which form the key objectives of the thesis:

[1] Latent alarm is a term describes alarm whose release is delayed until identifying the underlying causes of the disturbance or until the appearance of the disturbance as a deviation in the intended functionality of the system.

1.4.1 Development of a Distributed Monitoring Model

The intention for the proposed monitor is to operate on a monitoring model derived from safety assessment. As safety assessments yield an immense amount of knowledge, it seems vital to study this knowledge and select coherent and consistent models, which can effectively inform the monitoring agents to deliver safety monitoring tasks. In other words, a suitable monitoring model will be studied, decided, formalised, and exploited to provide the agents with executable reference knowledge.

The thesis develops two monitoring models that are derived via application of two state-of-the-art model-based safety assessment techniques: Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) and the Architectural Analysis and Design Language (AADL). One difficulty with the results of safety assessments is that they are not directly suitable for the purposes of safety monitoring; for example, they lack precise and executable semantics that could enable real-time evaluation and confirmation of events described during the assessment. Thus, part of this work is to develop and clarify linguistic constructs that enable precise specification of events, validation of measurements, long term monitoring, diagnosis of causes, and encoding of corrective measures.

1.4.2 Development of a Multi-agent System that Operates on the Distributed Monitoring Model

This objective requires investigation of the various options available for the design of intelligent agents as distributed reasoning objects that can operate on the proposed monitoring model. Belief-Desire-Intention (BDI) agents are examined for their suitability as monitoring agents. The appropriate deployment, design and implementation of the agents as well as collaboration protocols among them are also investigated to develop the entire multi-agent system. This step is also concerned with the design of effective monitoring algorithms that agents apply in order to reason on the monitoring model and observations of the monitored system and deliver the safety monitoring tasks.

1.4.3 Application in Two Case Studies and Evaluation of the Developed Monitor

This step involves implementation of the monitor and application of the approach in two case studies: an aircraft fuel system and an aircraft brake system. The aim is to evaluate the approach, in particular the effectiveness, timeliness, flexibility and scalability of the monitor, and discuss its relative performance in comparison to earlier relevant safety monitors.

1.5 Thesis Contribution

The main contribution of this research is the development of a novel distributed on-line safety monitor. To our knowledge this is the first monitor of its type. The monitor can use knowledge derived from safety assessment models and deploys the distributed reasoning of a multi-agent system composed of BDI agents. The monitor delivers a wide range of monitoring tasks which span from fault detection and diagnosis to effective alarm annunciation and fault controlling. Although there is some early work in the area of safety monitoring, no such work matches those characteristics. The most similar monitor is that developed by Papadopoulos (2000, 2003). The monitor exploited the safety assessment model to derive the monitoring knowledge and deliver a similar range of monitoring tasks. However, this monitor only operates on HiP-HOPS models as opposed to the monitor developed in this thesis, which also operates on AADL models. This earlier work is also based on a monolithic and centralised reasoning scheme, which is fundamentally different from the distributed monitoring approach developed in this thesis.

In the Background chapter, a number of multi-agent monitors are reported. Despite their exploitation of multi-agent systems and distributed reasoning, those monitors were developed from different deployment approaches and monitoring knowledge. The monitors also deliver some and not the whole range of the monitoring tasks that are deliverable by the monitor developed in this thesis.

Within the context of this research, a number of secondary contributions are also made. They can be listed as follows:

- The establishment of a formalised description of the monitoring model as an executable specification. This grammar is presented as E-BNF to formalise the monitoring knowledge.
- The establishment of an approach to distribute the monitoring model without violating the consistency and integrity of the monitoring knowledge.
- The establishment of a hierarchical approach to deploy monitoring agents according to the architectural view of the monitored system. This has supported the extensibility and hence the applicability of the developed monitor to different large-scale systems.
- The establishment of monitoring protocols that supported the integrity of the reasoning over the overseen conditions and the monitoring models of the monitored system and its sub-systems.
- The establishment of a number of monitoring algorithms that tracked the behaviour of the system and its components and diagnosed faults by relating the verified failure events to their underlying causes.
- The establishment of techniques that can filter out spurious sensory measurement and validate sensory measurements, detect, diagnose and control faulty sensors. These techniques supported, to a certain extent, the correctness of the deliverable monitoring tasks.

1.6 Thesis Structure

The rest of the thesis is arranged in seven chapters as follows:

Chapter Two: Background

In this chapter the relevant literature is examined in the light of the three monitoring tasks: fault detection and diagnosis, alarm annunciation and fault controlling. Both monolithic and multi-agent monitors that deliver some of these tasks are examined. A critical review of the literature which focuses on the benefits and shortcomings of earlier work provides the background and motivation for the approach developed in this thesis.

Chapter Three: Distributed On-line Safety Monitor: Overview and Distributed Monitoring Model

This chapter provides an overview of the distributed safety monitor developed in this thesis, giving definitions of the basic terms and sketching the position, role, architecture and constituents of the monitor. On developing the distributed monitoring model, the chapter discusses *what* types of knowledge should be encoded in that model and *how* that knowledge could be derived from safety assessment, formalised and presented to serve on-line reasoning. The formal grammar of the monitoring model is presented in E-BNF notation. The chapter explains the formalisation of the monitoring expressions, whose real-time evaluation verifies the normality and abnormality of operational conditions. Methods for filtering and validation sensory measurements and detecting, diagnosing and controlling faulty sensors are also discussed.

Chapter Four: Distributed On-line Safety Monitor: Multi-agent System

This chapter focuses on the multi-agent system. It discusses the deployment of monitoring agents, the main concepts and basic reasoning principles of BDI agents and justifies the particular selection of this paradigm to be distributed reasoning objects. This is followed by a definition of the collaboration protocols among the monitoring agents. The reasoning processes and roles of those agents are then described. The notion of programming the monitoring agents as plan libraries is introduced and those libraries are designed and programmed. Finally, the logical problem of the omniscience of the monitoring agents is discussed along with the interpretation process of the Jason programming language - Jason being an extended version of the AgentSpeak programming language used in this thesis to implement the multi-agent system.

Chapter Five: Distributed On-line Safety Monitor Based on HiP-HOPS and a Case Study of an Aircraft Fuel System

This chapter presents the first case study of this thesis, which was performed on a simulated model of an aircraft fuel system. It presents the fuel system and the deployment of the agents to monitor the system. It also develops a HiP-HOPS model of the fuel system from which the distributed monitoring model is derived. Thereafter, the feasibility and effectiveness of the monitor are demonstrated experimentally through the case study; a number of faults are injected to simulate abnormal conditions of the fuel system and test the ability of the monitor to deliver the intended range of monitoring tasks; fault detection and diagnosis, alarm annunciation and fault controlling.

Chapter Six: Distributed On-line Safety Monitor Based on AADL and a Case Study of an Aircraft Brake System

This chapter presents the second case study, on a simulated aircraft brake system. The chapter presents the aircraft brake system and the deployment of the agents for this new instance of the safety monitor. It shows that the generic architecture of the monitor can be instantiated to accommodate the requirements of a second and different system. The chapter develops an AADL model and error model of that system from which the distributed monitoring model is derived. It shows that the design of agents is generic enough to operate on a different monitoring model derived from a new set of safety assessment performed using a different methodology (AADL). The chapter then reports simulated fault scenarios, whereby the fault detection and diagnosis, alarm annunciation and fault controlling performed by the monitor are tested. The effectiveness of the approach is once more demonstrated through this second case study.

Chapter Seven: Evaluation

This chapter evaluates the achievement of this thesis. Firstly, it compares what has been achieved against the objectives set out in section 1.4 of the present chapter. Secondly, it evaluates the approach and effectiveness of the monitor via discussion of the application to the two case studies presented in Chapter Five and Chapter Six.

Direct experimental comparison with earlier monitors was impossible. However, the last section of Chapter Seven presents a theoretical comparison that contrasts the relative advantages and disadvantages of the monitor with those of a number of early developed monitors, which are reported in the literature.

Chapter Eight: Conclusion

This chapter summarises the research and draws conclusions. It also includes a section on future work in which limitations are identified, along with the recommended further research to address them.

Appendix A:

This appendix lists and explains abbreviations that appear throughout this thesis.

1.7 Note on Publications

The key aspects of this research have been published in three conference papers (Dheedan and Papadopoulos, 2010, 2011) and (Dheedan *et al.*, 2012).

Chapter Two

Background

2.1 Introduction

This chapter reviews the relevant literature on on-line safety monitoring. Researchers have approached the monitoring problem from different perspectives. In the work of Venkatasubramanian *et al.* (2003a, b and c) and Ng and Srinivan (2010), for example, the problem is approached as a task of fault detection and diagnosis. In the work of Hwang *et al.* (2008) and Hogg *et al.* (1995), it is approached as a process of announcing effective alarms that could help the operators in maintaining the operational safety of the monitored system. In other work, fault controlling is also discussed, e.g. see Jiang (2005) and Zhang and Jiang (2008). Finally, in several approaches safety monitoring is defined more broadly as a process that could deliver a range of tasks encompassing fault detection and diagnosis, alarm annunciation and fault controlling; see, for example, monitors developed by Papadopoulos (2003), Peng *et al.* (2007) and Lopez and Sarigul-Klijn (2010).

This chapter reviews and classifies extant approaches with respect to their capacity for fault detection and diagnosis, alarm annunciation, and fault controlling, as these are the typical tasks of on-line safety monitors. The chapter also reviews monitors that deliver all or some of those tasks and classifies the monitors architecturally into monolithic and multi-agent monitors.

2.2 Fault Detection and Diagnosis

Fault detection can be defined as the identification of abnormal events during system operation, whereas fault diagnosis can be defined as the process in which the underlying causes of the detected abnormal events are identified (Ma and Jiang, 2011; Johnson, 1996). Historically, the task of fault detection and diagnosis has demonstrated a spectrum of operational merits ranging from preventing abnormal events becoming serious hazardous failures to improving the availability and productivity of processes (Bhagwat *el al.*, 2003; Hu *et al.*, 2003).

Approaches to this task are, typically, developed from knowledge about the behaviour of the monitored system, which is encoded to inform on-line reasoning on the monitored conditions. According to the encoded knowledge, the approaches are classified into two main classes: model-based and data-based[1] approaches. These classes are classified further into qualitative

[1] Data-based approaches may also be called model-free based approaches (Ma and Jiang, 2011), process history based approaches (Venkatasubramanian *et al*, 2003a) or non-model-based approaches (Sayda, 2011).

and quantitative techniques. Figure 2-1 extends the classifications of Venkatasubramanian *et al.* (2003a) and Zhang and Jiang (2008), to capture a general typology of these approaches. The figure also lists a number of commonly-used detection and diagnosis techniques.

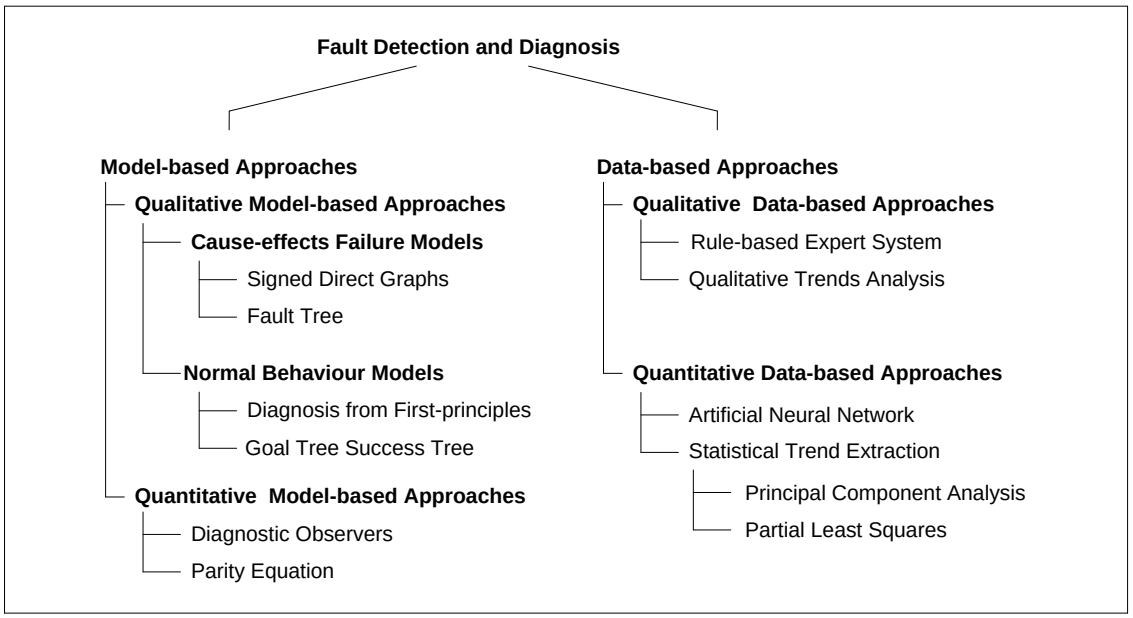


Figure 2-1: A Classification of Fault Detection and Diagnosis Approaches.

In addition to this classification, a comparative study by Venkatasubramanian *et al.* (2003c) has shown that each of the classified approaches and techniques has its own relative strengths and weaknesses and there is no single approach or technique that possesses all the desired features. The study has also shown that approaches and techniques can complement each other synergistically to enhance fault detection and diagnosis.

2.2.1 Model-Based Approaches

The distinctive characteristic of the model-based approaches is that their encoded knowledge is derived from off-line models of the monitored systems and a number of safety analysis techniques. Knowledge about the normal behaviour and operational structures is derived from design models, such as Data Flow Diagrams (DFD), Functional Flow Block Diagrams (FFBD), or more recently from models defined in the Unified Modelling Language (UML). Knowledge about abnormal behaviour (behaviour in conditions of failure) is typically derived from analysing the effects of potential failure scenarios on the normal functionality of the system. Such analysis is achieved via application of safety analysis techniques, such as HAZard and OPerability study (HAZOP), Functional Failure Analysis (FFA), and Failure Mode and Effect Analysis (FMEA) (Papadopoulos, 2003). The derived knowledge is then encoded in executable models to inform the on-line reasoning.

Design models and analysis techniques allow the derivation of qualitative and quantitative knowledge, which incorporates fundamental understanding of the structure, functionality and components of the monitored system. Accordingly, model-based approaches are commonly referred to as deep knowledge approaches (Milne, 1987).

2.2.1.1 Qualitative Model-Based Approaches

Qualitative model-based fault detection and diagnosis approaches can be classified into cause-effect failure models and models of normal behaviour (see Figure 2.1).

1. Cause-effect Failure Models

Cause-effect failure models typically record faults, error propagation paths and symptoms. Signed direct graph and fault trees are among the prominent examples of these models.

- Signed Direct Graph

In the qualitative modelling of the signed direct graph the relationships among monitored parameters are recorded in terms of deviations and consequent effects. The graph consists of a number of nodes and directed arcs. Every node records a possible deviation of one parameter and directed arcs link nodes to other dependent nodes (Lu and Wang, 2007).

Arcs might also be annotated with either a minus "-" or plus "+" sign. The minus signifies an inverse (negative) correlation between the linked nodes; a change in one parameter causes a change in the other parameter in the opposite direction. The plus sign signifies a positive correlation between linked nodes; a change in one parameter causes a change in the other parameter in the same direction (Ahn *et al.*, 2008; Kelly and Bartlett, 2006).

Figure 2-2 shows a simple example of a signed direct graph. It shows that parameter P3 is in negative and positive correlation with respective parameters P1 and P2. Parameter P5 is also in negative and positive correlation with respective parameters P3 and P4.

During the monitoring time, the deviant increase or decrease in the value of parameter P5 results in the verification of a fault. In such a case, a diagnostic algorithm investigates the correlations and values of the parameters on which P5 depends. If P5 is increased abnormally and P4 is less than or at its intended value then the investigation is initiated to P3. The decreasing in the value of P3 verifies the propagation path and initiates further investigation for the correlations and values of P1 and P2.

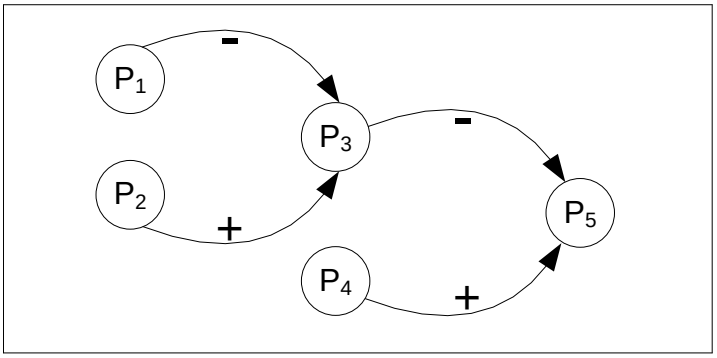


Figure 2-2: A Schematic Illustration of a Signed Direct Graph.

In the current monitoring practice, it has become normal to combine the model-based and data-based approach to generate effective fault detection and diagnosis techniques. The technique of Maurya *et al.* (2006) is developed from the combination of signed direct graph and qualitative trend analyses, which is a data-based approach, to detect and diagnose faults of complicated chemical processes. Qualitative trend analysis is employed to improve the accuracy in tracking the propagating paths and ultimately achieve effective diagnosis. Similarly, in the work of Dong *et al.* (2010) such a combination has also been exploited to detect and diagnose faults of industrial chemical processes.

- Fault Trees

The fault tree is a popular model used to represent the logical relationships between the causes and effects of failure in a system. Fault trees have precise semantics and they are widely used for safety and reliability analysis of systems. A fault tree can be seen as a graph that consists of nodes representing events connected with logical gates. The graph has a single initiating node at the top (also known as the top event), intermediate nodes and terminal nodes at the bottom (also known as basic events).

The top event node and intermediate event nodes represent effects of failure and they are linked to children nodes via a logical gate, an AND or OR gate. Basic events represent the root causes of failure in the system. A set of basic events whose occurrences result in the occurrence of the top event is called a cut-set (Vesely *et al.*, 1981). Unlike the signed direct graph in which the effect and its possible causes are predominantly linked by an OR gate, in the fault tree effects and possible causes can also be linked with AND gates.

In a diagnostic scenario, when the occurrence of the top event is verified, the diagnostic algorithm traverses the fault tree from that event and looks to verify symptoms at progressively lower levels tracking the propagation path and until the basic event nodes that enclose the underlying root causes are reached. When AND gates are encountered in the course of the

traverse, the diagnostic algorithm verifies that a conjunction of the inputs to the gate has occurred. On the contrary, encountering OR gates requires further investigation to verify a disjunction of events at the inputs (Papadopoulos, 2002).

Large fault trees are difficult to construct manually; the process is laborious and error prone. The possibilities of automating that process have therefore been extensively researched. In the work of Lapp and Powers (1977), an algorithm that could automatically synthesise fault trees from signed direct graphs fault tree has been developed.

More recently, Papadopoulos *et al.* (2001) have developed a tool called Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) to automatically synthesise fault trees from architectural model of systems which are annotated with local failure logic of components. This logic shows for each component how local output failures of the component are caused by internal failures and deviations of component inputs. The top events of fault trees are deviations of parameters in the outputs of a system and the fault trees are synthesised by traversing the model and interpreting the local failure logic encountered in the course of the traversal. These fault trees have also been exploited by Papadopoulos (2002) as a diagnostic model. For the purposes of diagnosis, a real-time algorithm that combines between blind-depth-first and heuristic traverses has been developed to traverse the fault tree and relate the top event to its underlying causes.

2. Normal Behaviour Models

In this type of qualitative model-based approach, the monitoring model encodes knowledge that is only about the normal behaviour of the monitored system. Fault detection and diagnosis reasoning is established through the real-time instantiation of modelled parameters with sensory measurements. Difference between the modelled and observed parameters suggests the presence of a fault. The underlying causes of the detected faults are diagnosed via further comparison between modelled behaviour and observed symptoms. Diagnosis from first principles and the Goal Tree Success Tree are among prominent examples of these models.

- Diagnosis from first principles

The theoretical foundation of diagnosis from first principles was proposed by Reiter (1987). In this technique the underlying causes of the detected faults are diagnosed by (a) identifying the minimal set of the suspicious components that could contribute to that fault; (b) examining formulae that describe the functionality of those components. The detection algorithm instantiates such formulae with sensory measurements and checks their consistency with expected results. The verification of inconsistency signifies fault detection and accordingly a

diagnostic process is launched. The diagnostic algorithm, in turn, identifies the minimal set of the suspicious components, instantiates their formulae with real-time sensory measurements and again checks their consistency.

Figure 2-3 depicts the model of a multiplier and adder system. Assuming that a fault in one of the components of the system causes a result of F=10 instead of F=12, looking at the structure of the system, components that may contribute to the detected deviation are the following minimal set: ADD-1, MULT-1, MULT-2.

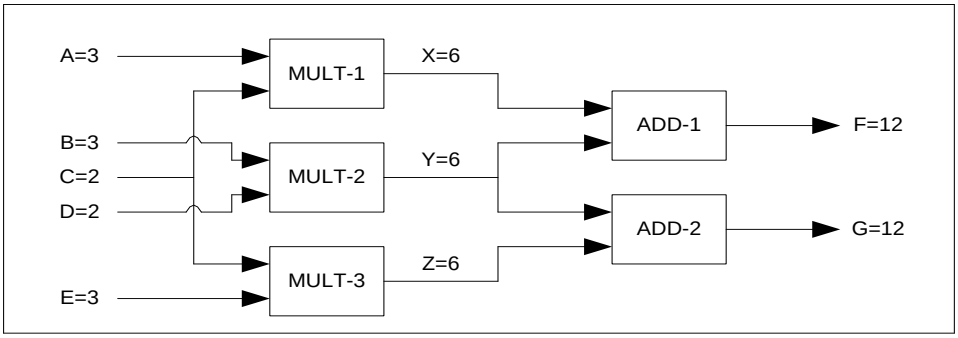


Figure 2-3: A Multiplier and Adder System (Davis, 1984).

In order to diagnose precisely the underlying cause of that deviation, the formulae of each of those components are examined and the resulting inconsistency is tracked according to the structural abstraction. The logical formulae of the component MULT-1 are as follows:

$$X = A * C;$$
$$A = X / C;$$
$$C = X / A;$$

By instantiating the above formulae and similarly those of the other components in the minimal set with the real-time sensory measurements and examining their consistency, the underlying cause can be diagnosed. The algorithm instantiates and checks the consistency starting from the first component which receives the system's inputs and proceeds progressively towards the last component, which delivers the deviant function.

According to the structure of the system the first examined component would be either MULT-1 or MULT-2. If the examination of either of those components shows consistent results, then that indicates that it is not a faulty component and accordingly the diagnosis process moves to another component. If, on the contrary, the examination shows inconsistent results, then that indicates that the component is faulty and accordingly it would be diagnosed as an underlying cause. If, moreover, the examination shows that all the components in the minimal set are consistent with their normal functionality, then that indicates none of the components are faulty and the underlying cause lies in the inputs of the system.

The monitoring virtue of this technique is that in the case of multiple faults the minimal set could help in diagnosing the underlying cause effectively. Assuming that a fault causes results of F=10 and G=10 instead of F=12 and G=12, respectively, while the minimal set of the former function is ADD-1, MULT-1, MULT-2, the minimal set of the latter function is ADD-2, MULT-2, MULT-3. Achieving intersection on the two sets would result in MULT-2, which is the underlying cause of the detected multiple faults (Davis and Hamscher, 1992).

However, this technique has limited applicability since it requires a precise matching between the structural and functional abstractions to identify and examine the consistency of the suspicious components. The theoretical foundation of this technique does not offer a capacity to formulate dynamic behaviours and monitor dynamic systems, whose components are engaged in several structures to deliver different functionality, e.g. phased-mission systems[1]. The applicability of this technique has been rather confined to monitoring electrical circuits.

- Goal Tree Success Tree (GTST)

Systems whose behaviour is dynamic and constructed from a considerable number of components may call for a bigger capacity model rather than the limited abstraction offered by diagnosis from first principles. The functional and structural view of such systems is better described as a hierarchical abstraction (Modarres and Cheon, 1999).

A Goal Tree Success Tree (GTST) is a model that can hierarchically abstract thorough knowledge about the functional and structural implementations of the monitored system. Figure 2-4 shows the way in which a system is modelled as a hierarchy of structural and functional decompositions and converted into GTST.

GTST can be viewed as composed of two constituents: goal tree and success tree. In the goal tree the functionality of the system is presented as the top objective, which is divided further into lower-level goals. Similarly, the goals are divided into lower-level sub-goals. The division proceeds hierarchically from the top level until it reaches the level at which sub-goals (functions) cannot be divided further. Note that the functional decomposition also reflects the structural decomposition of the system to sub-systems and components, so sub-goals represent functions of sub-systems. In the success tree the focus is on the potentially observable/verifiable conditions that assure the achievement of the sub-goals (Kim and Modarres, 1987).

[1] A typical example of phased-mission systems is an aircraft; it delivers its mission through a number of phases: pre-flying, taxiing, take-off, climbing, cruising, approaching, and landing. Over every phase the phased-mission system engages its components in a certain topology and delivers a specific functionality.

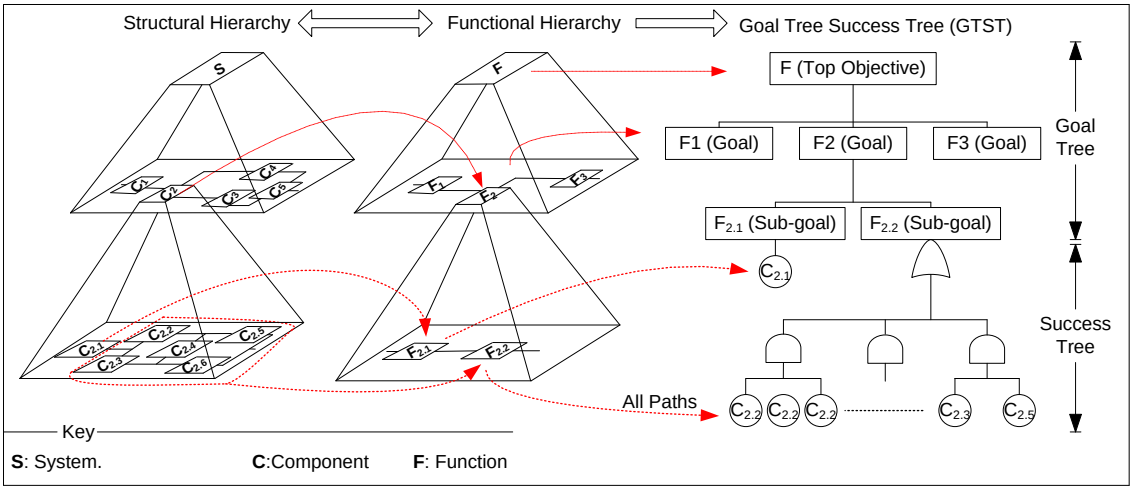


Figure 2-4: Structural and Functional Hierarchy and GTST (Modarres and Cheon, 1999).

Typically, faults of the components of the monitored system result in the delivery of an abnormal top objective (system functionality); this represents detection of a fault and accordingly a diagnostic process is launched. A diagnostic algorithm traverses the GTST top-down and whenever it faces parameters (goals or sub-goals) they are instantiated with real-time sensory measurements and evaluated to track the error propagation path and ultimately diagnose the underlying causes (basic components).

In the work of Modarres and Cheon (1999), GTST has been joined with master logic diagram (MLD), which is a diagram used to identify hazardous events of the critical process[1], to develop a diagnostic model for complex systems. In the work of Chung *et al.* (1989) the capability of the GTST as a diagnostic model is improved by replacing the success tree with operating knowledge represented by if-then rules. A depth-first traversal is used to traverse the model from symptoms towards underlying causes. The technique is accordingly called GOal TRee Expert System (GOTRES) and has been applied to the detection and diagnosis of faults in nuclear power plants.

FAX, developed by Chen and Modarres (1992), is another fault detection and diagnosis technique that exploits GTSTs. In this technique, traversal of the GTST is facilitated by an inference algorithm that incorporates Bayesian theorem and two traverse strategies: depth-first traverse and adaptive traverse. The main reason behind incorporating Bayesian theorem is to deal with uncertainty, which may result from sensory failures or limited sensory measurements. FAX has been successfully applied to complicated chemical processes.

Another variant of the GTST is the Multi-level Flow Model (MFM). MFMs contain goals, functions and components (Larsson, 2002). Figure 2-5 shows the way in which the components contribute to achieving functions and how the collaboration among those functions could

[1] More information about MLD can be found in Papazoglou and Aneziris (2003).

contribute to achieving goals which in turn contribute to achieving the main goal of the system. As the faults are detected at the top goal, a backward chaining algorithm traverses the MFM from the top level goal towards diagnosing the underlying cause at the components level (Larsson, 1996).

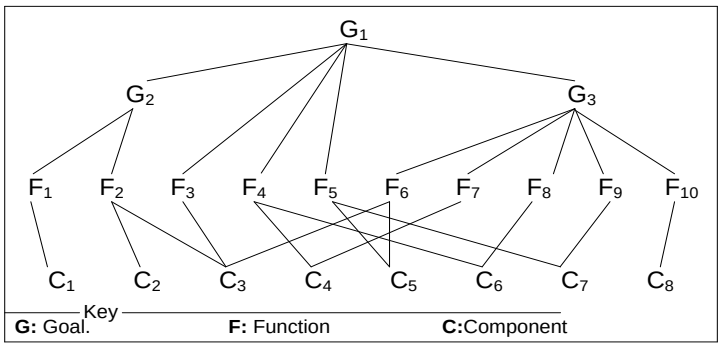


Figure 2-5: Component, Functions and Goals Modelling in the MFM (Larsson, 1994).

2.2.1.2 Quantitative Model-based Approaches

In the quantitative model-based approaches, the normal and abnormal behaviour of the monitored system is modelled in the form of mathematical relations. Normal behaviour is modelled as a number of relations, where each relation represents a parameter and its intended value (result of the relation) is modelled also. In detecting and diagnosing faults, the approaches depend mainly on an assumption that every fault results in a certain abnormal value of the relevant parameter. Accordingly, the abnormal behaviour of the system is modelled as the differences (residuals) between the possible abnormal values and intended values of the parameters. Every residual represents a unique signature of a fault and thus the faulty components can be diagnosed.

Based on this notion, two different techniques have been proposed: diagnostic observers (Beard, 1971) and parity space (Willsky, 1976).

Diagnostic Observers

To develop the diagnostic observers technique, relations, intended parametric values and residuals are modelled. During the monitoring time relations are instantiated with sensory measurements, their values are calculated and the differences between the calculated and the intended values also calculated. If the resulting difference equals or lies close to zero, then that signifies the normality of the conditions. On the contrary, if it is not equal to zero, then that signifies a parameter deviation and accordingly a fault is detected. Hence, a diagnostic process in which that difference is matched with the residuals is launched. The process, in turn, results

in a unique matched residual according to which the underlying causes are diagnosed. In order to deliver effective fault detection and diagnosis, issues such as uncertainty and disturbance should be taken into account while modelling the residuals (Zolghadri *et al.*, 1999).

Diagnostic observers have been widely exploited to detect and diagnose faults of dynamic systems, i.e. phased-mission systems. In the work of Narasimhan *et al.* (2008), the diagnostic observer is exploited to detect faults, diagnose the underlying causes and isolate the faulty components of chemical engineering systems. Lyapunov-like analysis is used to analyse and model the dynamic behaviour of the monitored system and also to achieve a clear distinction between the normal and abnormal behaviour and modelling effective residuals. In this technique, sets of functional relations and residuals are modelled for every phase and according to the given phase the corresponding set is activated to inform the on-line fault detection and diagnosis reasoning.

Parity Space:

Parity space (or parity equations) depends also on prior generated residuals and in this respect it is alike to the diagnostic observer approach. The difference, however, is that in parity space, in addition to modelling the parameters, the entire functionality of the monitored system is also modelled as a parity (equations) model. The parity space technique requires two types of residuals. The first is for potential deviations of the parity model by which the party space of the system is checked. The second is for the potential deviations of the parameters by which the underlying causes are diagnosed. In real time, the variables of the parity model are instantiated with real-time sensory measurements, the equations are calculated and accordingly the ultimate result of the model is compared with the intended value to decide their parity. If a disparity is verified then that signifies fault detection and thus a diagnostic process is launched. In this process, a diagnostic algorithm calculates the difference between the calculated and intended values and matches it with the modelled residuals (of the parity model). According to the matched residuals a number of suspicious parameters are identified. In a further diagnostic step, the differences between the values of the suspicious parameters and their intended values are calculated. Any non-zero difference is moreover matched with the modelled residuals and thus the fault signature is identified and underlying causes are diagnosed.

Parity space has been widely exploited in detecting and diagnosing sensory failures (Chen and You, 2008). In the work of Abdelghani and Friswell (2001) the parity space has been exploited to develop a generic technique capable of detecting and diagnosing the sensory failures. Similarly, Borner *et al.* (2002) also exploited parity space to detect and diagnose sensory failures of the suspension and hydraulic brake systems of vehicles. Recently, El-Mezyani *et*

al. (2010) exploited parity space to develop a cost-effective technique to detect and diagnose sensory failures of power systems.

2.2.2 Data-based Approaches

As shown in Figure 2-1, the data-based approach is another of classes of fault detection and diagnosis techniques. Unlike model-based approaches, data-based approaches do not derive monitoring knowledge from off-line models of the monitored system. They rather encode knowledge derived from experimental observations and the on-line context of the monitored system.

Techniques that are developed from data-based approaches may appear similar to those developed from model-based approaches, as they also incorporate normal and abnormal behaviour and rely on monitoring the parameters of the monitored system to detect faults and on already modelled residuals to diagnose the underlying causes.

In the data-based approaches, knowledge about the normal behaviour of the monitored system is modelled from an empirical analysis of real-time observations (training data). In order to model knowledge about the abnormal behaviour, FMEA is applied to identify the potential faults of the basic components of the monitored system. Those faults are then injected experimentally and the resulting symptoms and ultimate effects on the functionality of the monitored system are modelled (Ma and Jiang, 2011). The derived knowledge is then encoded in executable models to inform the on-line reasoning.

As data-based approaches depend on knowledge derived from the on-line context of the monitored system, they are largely referred to as shallow knowledge approaches. This is also because they do not incorporate knowledge about the architecture, operational structure or relationships among the deliverable functions of the monitored system (Petti *et al.*, 2004).

2.2.2.1 Qualitative Data-based Approaches

As shown in Figure 2-1, qualitative data-based approaches include rule-based expert systems and qualitative trained analysis as prominent among the developed techniques.

- Rule-based Expert System

The earliest emergence of the rule based system was perhaps in the medical field. Such systems have been used widely in diagnosing diseases from the associated symptoms. The typical development of fault detection and diagnosis expert systems is achieved through four steps: (a) experimentally obtaining knowledge about possible faults, their associated symptoms and

underlying causes; (b) encoding that knowledge as a knowledge base; (c) developing an inference engine that could relate those faults to their underlying causes by examining the associated symptoms; (d) developing an interactive interface among the symptoms, inference engine, knowledge base and the monitored system (and/or system operators) (Giarratano and Riley, 2005).

Among the earliest monitoring expert systems is the REACTOR, developed by Nelson (1982). REACTOR was developed originally for detecting, diagnosing and controlling the faults of nuclear reactors. It consists mainly of a knowledge base and procedural inference engine that is able to reason backwards and forwards on the knowledge base. The inference engine reasons forwards from the known facts towards inferring detection and diagnosis conclusions. If those facts are insufficient then backward reasoning is launched. Backward reasoning, in turn, achieves further query of the sensors and/or the operators to fill the gap in the knowledge base.

REACTOR's knowledge base consists of function-oriented knowledge and event-oriented knowledge. While the former is about the delivered functionality of the monitored plant in normal conditions, the latter is about the potential behaviour of the plant in the presence of faults. Event-oriented knowledge is always obtained from historical experience with the occurrences of actual faults during the operating time of the plant. The knowledge is constructed as if-then rules that would be evaluated by the inference engine in real time to detect a fault and diagnose its underlying cause.

In the work of Ramesh *et al.* (1988 and 1989) a hierarchical organisation was introduced to the knowledge base of an expert system. The intention was to increase the efficiency of the inference engine and establish prompt detection and diagnosis of the faults of large-scale systems. In another work, Rich *et al.* (1989) also developed a rule-based technique that could achieve prompt detection and diagnosis responses. The technique was based on a knowledge base that classifies potential faults into three classes: operator misleading, monitoring instrument faults and system malfunction.

- Qualitative Trends Analysis

Qualitative trends analysis is another technique of qualitative data-based approaches[1] in which measured signals are represented as a number of basic shapes, such as decreasing, increasing, constant and so on. Shapes are called primitives, a combination of which comprises a trend. Figure 2-6 shows seven primitives: A(0,0), B(+,+), C (+,0), D(+,-), E(-,+), F(-,0), G(-,-). The first and second signs of every primitive represent the first and second derivative, respectively.

[1] In some work, qualitative trend analysis has been classified as a semi-quantitative data-based technique due to its dependency on calculated residuals to detect and diagnose faults (Maurya *et al.*, 2005).

It is also possible to specify more primitives, e.g. sharp decrease or increase, or even use some of the seven primitives to represent intended sensory signals (Maurya *et al.*, 2005).

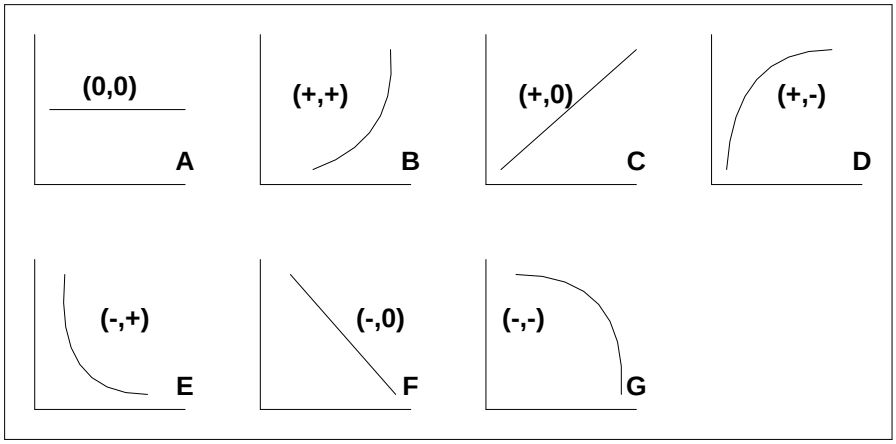


Figure 2-6: Seven Example Primitives of the Qualitative Trends Analysis (Maurya *et al.*, 2005).

Normal sensory signal (trend) is represented as a number of the primitives. During the monitoring time the real-time signals are compared with the modelled trends. If they matched then that signifies normal conditions. If, on the contrary, there is a discrepancy then that signifies the detection of a fault and accordingly a diagnostic algorithm calculates the residuals and matches them with already modelled residuals to diagnose the underlying causes.

In the work of Maurya *et al.* (2007), qualitative trends analysis is combined with signed direct graph to develop a technique for fault detection and diagnosis of chemical processes. In that technique, the signed direct graph represents a level that records the potential set of faults, while the qualitative trends analysis represents another higher level that holds the actual trends. This technique has resulted in the delivery of accurate, reliable and fast-response fault detection and diagnosis.

In more recent work, Maurya *et al.* (2010) developed another technique to detect and diagnose faults of more complicated and critical chemical processes. The technique incorporates a special algorithm for on-line extraction[1] of qualitative trends and a framework in which those trends are presented as an updatable knowledge base for real-time monitoring reasoning. In addition to matching the trends and updating the knowledge base, the monitoring algorithm is also able to estimate the fault occurrence time and thus an accurate detection of the faults can be achieved.

[1] To eliminate momentary or disturbance behaviour that may occur due to mode transitions or interfering noise and come up with a good quality of trends, the on-line extraction algorithm incorporates Kalman filter, which is a quantitative model-based technique. More information about Kalman filter can be found in Maurya *et al.* (2010).

2.2.2.2 Quantitative Data-based Approaches

As shown in Figure 2-1, quantitative data-based fault detection and diagnosis approaches can generally be classified into non-statistical and statistical approaches, which can be exemplified by artificial neural network and statistical trend analysis, respectively.

- Artificial Neural Network

An artificial neural network is a set of computational nodes (or units) connected to each other by a number of links. Every link is associated with a numerical value called a weight. Weights are the means upon which the inputs and the outputs of a node are processed and delivered, respectively. The weights are stored as long-term means in the neural network. Weights may, however, be updated if the neural network learns new criteria that should be applied for future processing.

Every unit may have one or a number of input links coming from other nodes in the neural network. A unit may also have one or a number of output links go to other units in the network. Some units might be connected to the external environment for receiving inputs from or sending outputs to that environment. In the neural network each node achieves local processing for its own inputs according to the weights that are relevant to those inputs. There is no global control of the local process, but there is synchronous control to update all the nodes consistently in a fixed sequence (Russell and Norvig, 2010).

Figure 2-7 illustrates the local computation of a unit. The computation is achieved on the signals received by the inputs (a_j) to produce a new activation level (a_i) which in turn is sent out through the outputs. The computation consists of two processes; the first is a linear computation process achieved by the input function and the second a nonlinear process achieved by the activation function.

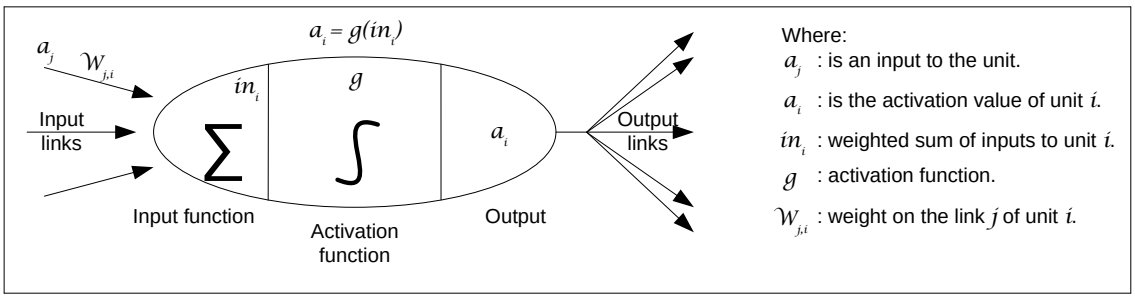


Figure 2-7: A Unit of a Neural Network (Russell and Norvig, 1995)

According to Russell and Norvig (1995), the total weighted input is calculated as the summation of the input activations multiplied by their relevant weights and as follows:

$$in_i = \sum_j W_{j,i}\, a_j$$

The activation value is calculated as follows:

$$a_i = g(in_i)$$

To build a neural network there are different (a) activation functions, such as step basis, redial basis, sigmoidal basis and feedforward; (b) learning strategies, such as supervised and unsupervised. The topology of neural networks and the related computation functions and learning strategies have been extensively researched with the aim of developing effective fault detection and diagnosis techniques.

In the work of Rusinov *et al.* (2009) a neural network is exploited to develop fault detection and diagnosis of chemical processes. In this approach, expert analysis techniques, such as projection of fuzzy values of diagnostic parameters, principal components space and HAZOP, are exploited to achieve effective estimation of parameter deviations and identifications of weights. Moreover, in order to develop an effective network that can cope with the complicated architecture of the monitored process, a number of neural networks are designed and arranged in a hierarchy. The networks are unsupervised and they exploit feedforward function to reason and detect deviations of the parameters and also to diagnose the underlying causes of those deviations. Practically, at the highest hierarchical level a network detects and identifies deviations and at the lower level a set of neural networks relate the deviations to their underlying causes.

- Statistical Trend Extraction

In normal conditions, the real-time behaviour of the monitored system should show probabilistic distribution that matches the intended distribution. Random or abnormal changes in that behaviour occur only due to faults. Accordingly, by finding the difference (residual) between the intended distribution and the abnormal distribution, faults can be detected, their signature can also be identified and thus the underlying causes are diagnosed. Distribution differences should already be trained experimentally and according to the potential faults.

The computational implementation of the statistical trained extraction depends mainly on the mean and standard deviation values of the monitored parameters. If the values of the means and standard deviations of monitored parameters are consistent with those of the normal behaviour then that verifies normal operation of the monitored system. On the contrary, an inconsistency signifies the occurrence of a deviation in a parameter value and according to the differences

between the modelled and real-time values the underlying causes are diagnosed (Venkatasubramanian *et al.*, 2003c).

Based on this notion a number of fault detection and diagnosis techniques have been developed. The majority of those techniques depend on statistical methods, such as principal component analysis and partial least squares (MacGregor *et al.*, 1995; Wise and Gallagher, 1996).

In the work of Kaistha and Upadhyaya (2001), principal component analysis is exploited to analyse faults of nuclear power plants and diagnose their underlying causes. In more recent work, Miller (2006) has developed another technique in which the principal component analysis model is exploited to differentiate between the normal and abnormal conditions of chemical processes.

In a different fault detection and diagnosis technique, partial least squares has been exploited to reduce the number of parameters of the monitored process, through combining them in the least number of relations, and thus improve the quality of the statistical reasoning. In the work of Wold *et al.* (1984), the partial least squares technique is exploited to support the effective statistical fault detection and diagnosis of a chemical process.

2.3 Alarm Annunciation

Fault detection and diagnosis is a prerequisite for successful safety monitoring. Optimising the presentation of alarms is an equally important task. Despite the highly automated nature of contemporary safety-critical systems, human operators are still required in controlling and responding to faults. Typically, an alarm is the key means by which the occurrence of faults is brought to the attention of the operators so that correct controlling and directive decisions can be taken (Hwang *et al.*, 2008). Figure 2-8 depicts the relation among the monitored system, alarm, controlling and the operators.

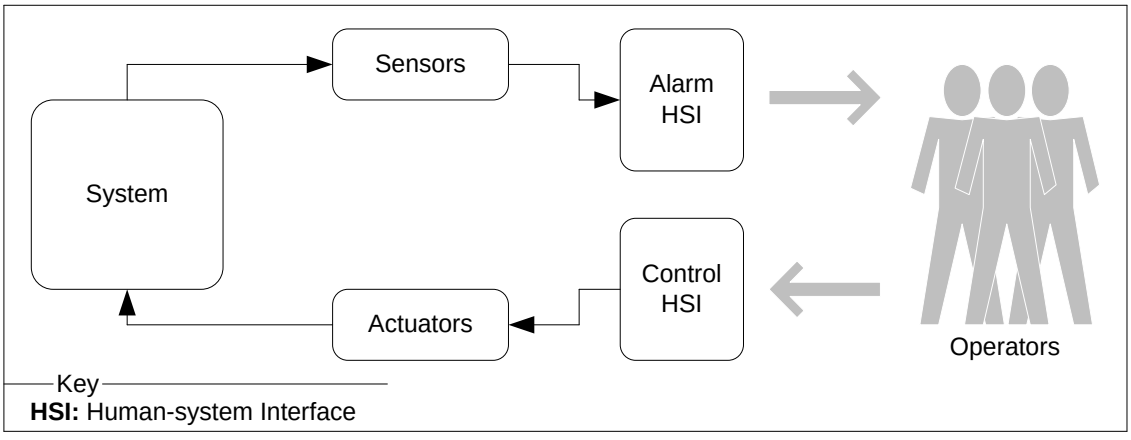


Figure 2-8: An Abstracted View of System, Alarm and Operators (Huang, 2007).

Given that alarms provide the means of informing operators about faults, it is ironic that alarm function itself contributes to monitoring problems that may lead to disasters. The industrial disaster of the Three Mile Island nuclear reactor in 1979 (Suh *et al.*, 2008), the explosion of the Texaco Milford Haven refinery in 1994 (HSE, 2000), and the avionics disaster of the Kegworth Air disaster 1989 (Trimble, 1990) all represent cases where ineffective alarm techniques confused the operators with an alarm avalanche and led them to take incorrect action which exacerbated emergency conditions.

A variety of alarm problems have been experienced. In highly dynamic systems, behavioural transitions may confuse the alarm annunciation technique and result in releasing false alarms that do not reflect the actual conditions of the monitored system (Lee *et al.*, 2010). Faults of the alarming instruments, such as sensors and instrumentation circuits, may also expose operators to false and misleading alarms (Hines and Davis, 2005). False and misleading alarms are a nuisance that can distract operators and cause unnecessary or inappropriate actions that may reduce the safety of the system.

As systems become more complex, the ability of the operators to understand promptly the potential safety consequences of low level faults becomes increasingly challenged (Lupton *et al.*, 1992). There is therefore a broader issue of effective presentation and organisation of alarms, and this is becoming urgent with the increasing complexity of modern critical systems.

Jang *et al.* (2008) and Brown *et al.* (2000) have identified three processes that should be considered during the development of an alarm annunciation technique: alarm definition, alarm processing, and alarm prioritisation. Figure 2-9 summarises those processes along with methods used to realise them.

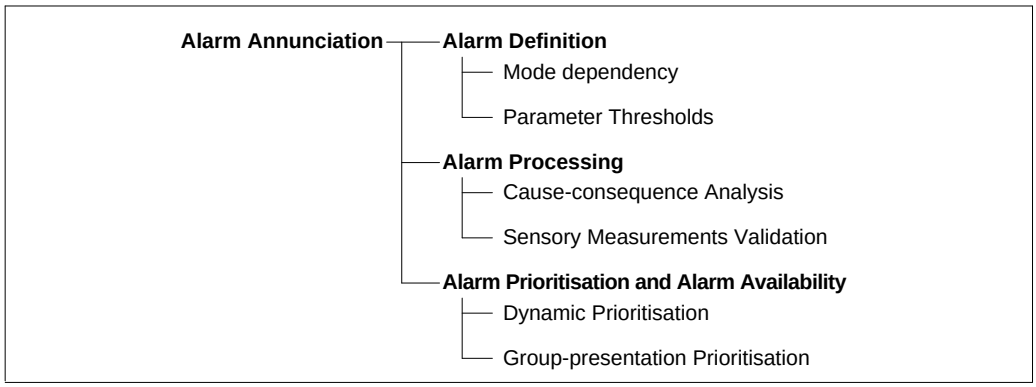


Figure 2-9: Strategic Processes of Designing an Alarm Scheme.

2.3.1 Alarm Definition

Alarm definition involves the definition of the mode dependency and parameter thresholds of the monitored system. Mode dependency is an explicit distinction between different operational structure and deliverable functionality during the normal operation of a system. Each mode is triggered by a certain event, i.e. change the set-point, and the deliverable functionality of a mode can be monitored through a specific set of parameters.

Parameter threshold is a value set against the real-time measurements of the deliverable functionality. Whenever a threshold is violated by the real-time functionality, the occurrence of a deviation is verified.

2.3.1.1 Mode Dependency: Specifying the Parameters and the Set-points

The interpretation of events is typically sensitive to the system context, often given by the modes of the system. An event which is considered normal in one mode can represent a symptom of failure in another mode. Figure 2-10 shows an example of a simple batch process, which is a heating sub-system. The sub-system is a part of a chemical plant and it delivers its functionality cyclically over four modes: idle, filling, heating and emptying. Modes are illustrated in Figure 2-10 as an operational sequence in GRAFCET notation (Oulton, 1992).

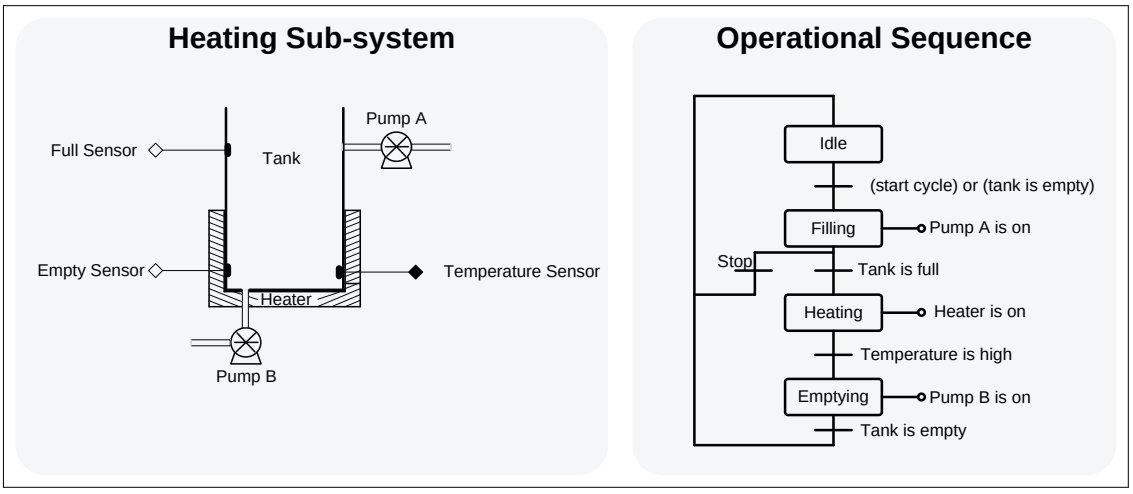


Figure 2-10: A Sub-system and its Operational Sequence (Papadopoulos and McDermid, 2001).

In the idle mode, the sub-system is idle and all the components are off. During the filling mode, pump "A" is on and filling the tank with a chemical liquid. During the heating mode, pump "A" is switched off and the heater is switched on. Finally during the emptying mode, pump "A" and the heater are switched off and pump "B" is switched on to empty the tank. The operation is repeated until the stop signal arrives (Papadopoulos and McDermid, 2001).

In the context of the four modes, events that are signified by the sensors of the sub-system are interpreted differently. In the idle mode, the occurrence of the event "tank is empty" is normal and the operators do not need to be alarmed. However, in the heating mode the occurrence of this event signals abnormal conditions and the operators should be alarmed accordingly. In the emptying mode, this event should occur to trigger the normal transition to the idle mode, whereas its absence signifies the occurrence of abnormal conditions on which the operators should be alarmed.

Hwang *et al.* (2008) have developed an advanced alarm technique that is able to interpret the occurrence of different events of nuclear power plants and inform the operators accordingly to make them aware of the conditions. The operators are always provided with pre-alarm (early alarm) as (a) alert on forthcoming normal transitions so they would be aware of possible false alarms that may be released subsequently; (b) prognoses on the disturbance at an early stage and before it escalates to a real hazard, so they could take the proper actions. This technique exploits the control chart and the trends of the active parameters. The control chart typically records the set-points of the monitored process and informs the alarm reasoning about the mode transitions of the monitored process, so the corresponding parameters are activated and monitored. Trends of the active parameters are used to monitor the behaviour of the parameters and verify the occurrence of the actual disturbance on which alarm is released.

2.3.1.2 Parameter Thresholds

To detect disturbances in their early stages, the thresholds of parameters that define normality and/or abnormality should be specified effectively by achieving trade-off among three factors:

- The need to filter spurious sensory measurements.
- The need to provide operators with important alarms but not every symptom of failure.
- The need to respect temporal constraints and ensure the timeliness of alarms.

Thresholds should not be specified with a too high level of sensitivity. Such a situation might result in the release of false alarms due to different types of noise, e.g. ionisation radiation and electromagnetic interference or potential sensory bias. Thresholds should also not be too relaxed, which could result in not detecting deviations and depriving operators of knowledge of the actual conditions. In between sensitivity and indifference, thresholds should be specified to achieve effective timeliness in releasing an alarm (Brown *et al.*, 2000).

Davey *et al.* (1995) developed a technique that can achieve trade-off among the three factors and ultimately improve alarm annunciation for the Canadian Deuterium Uranium (CANDU). In this technique, alarm messages are generated early via condition monitoring and calculating the

rates and margins of parameters. Alarms are released whenever the rates are close to the margins, thereby affording the operators sufficient time to make the correct judgement by predicting future conditions from the early alarm messages. Despite the effective timeliness demonstrated by this technique, the huge number of early produced alarm messages may overwhelm and confuse the operators.

To address this issue, another alarm technique was developed by Augustine *et al.* (2008), with the aim of improving the awareness of the operators of cement industrial processes. It involves a knowledge base from which alarm messages are composed and presented to the operators. Messages integrate a range of alarm information, such as identifying the deviation that causes the alarm and recommendations and guidance for corrective actions that could effectively help the operators in taking the corresponding fault controlling actions.

Similarly, Jang *et al.* (2008) have developed an Advanced Alarm System (AAS) that consists of a Visual Display Unite (VDU) and Active Database System (ADS). The main role of the AAS is to provide the operators with sufficient feedback and alarm under normal and abnormal conditions, respectively. AAS depends mainly on extracting the relevant feedback and alarm information from the ADS. The idea of AAS, in turn, gave rise to the development of the System-integrated Modular Advanced Reactor-alarm System (SMART-AS), which is widely used in the control room of nuclear power plants (Jang *et al.*, 2008).

Figure 2-11 shows an example of the guidance message provided to the system operators in association with the released alarms. The message advises and guides the operators to carry out further checks before taking action. According to the outcome, the operators might decide to ignore the alarm message or to go ahead with the required action.

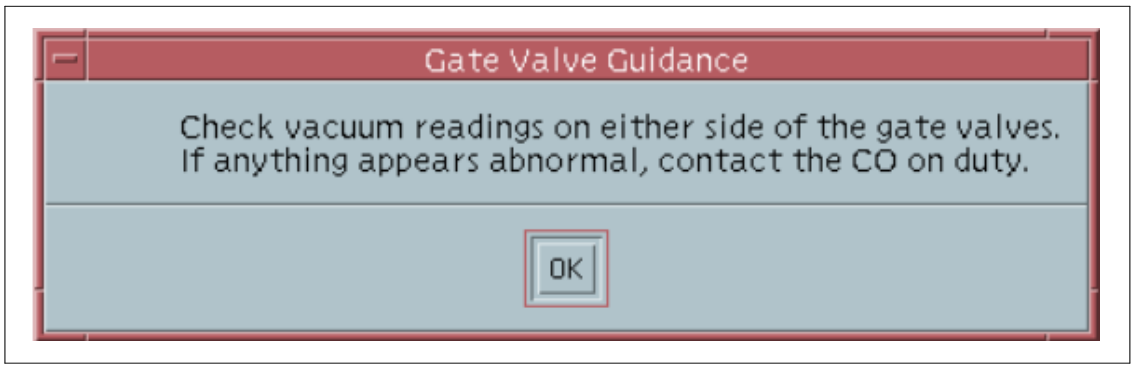


Figure 2-11: Guidance Text Produced by an Advanced Alarm System (AAS) (Anderson, 2007).

2.3.2 Alarm Processing

In alarm processing, the technique should be able to achieve accurate distinction among genuine, consequent and false alarms. As genuine alarms represent the occurrence of the

disturbances, they should be presented to operators in a meaningful way. To avoid alarm avalanches, a cascade of related alarms caused by propagation of faults and false alarms should be filtered out.

2.3.2.1 Cause-consequence Analysis

In the intelligent alarm processing system, a functional graph is exploited to reduce the number alarms and provide effective feedback for two different types of operators: the maintenance operators who are concerned with the underlying cause of the detected fault and the piloting operators who are concerned with keeping the system operative (Ghariani *et al.*, 2002).

The functional graph of Figure 2-12 illustrates an example of the relationship between the original and consequent alarm. In that graph, square nodes represent monitored functions (parameters). This implies that on the fault of F11 a genuine alarm is released and shortly thereafter another alarm is released on the consequent fault of F3. From the functional graph, it can be seen that the later alarm belongs to the same propagation path, that is, F11, F6 and F3. Thus, the alarm that is released on the fault of F11 is genuine, while the other one is consequent.

To provide the piloting operators with reduced alarm information the consequent alarm inhibits the causal alarm; pilot operators are provided with alarm on F3 only. On the contrary, to provide the maintenance operators with reduced alarm information, only the causal alarm on F11 is provided (Ghariani *et al.*, 2002).

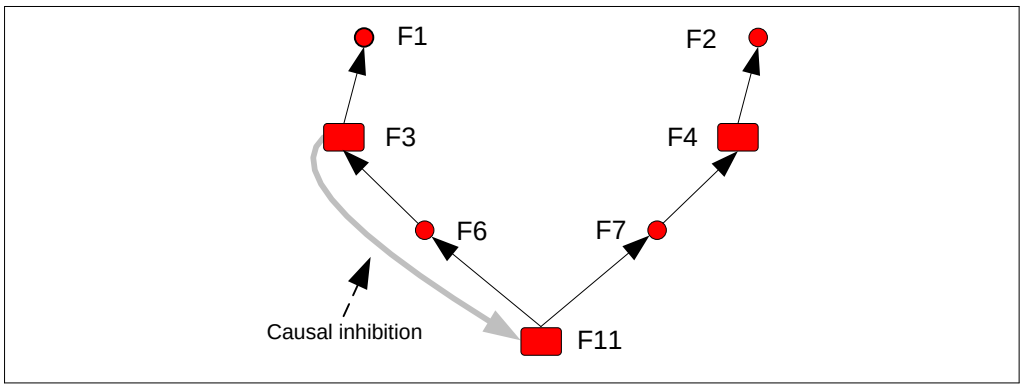


Figure 2-12: Functional Graph for Alarm Reducing (Ghariani *et al.*, 2002).

It is debatable whether this strategy would always work. The technique might work when both the piloting and maintenance operators work together in the same control room. In other context, it might have limited effectiveness. In modern aircraft, for instance, the flight deck crew typically comprises two or three pilots with no flight engineer, as the role of the engineer has been eliminated. Inhibiting causal alarms and releasing only consequent alarms could deprive pilots of good situational awareness and hinder timely diagnosis and control of failures.

In a different technique developed by Lee *et al.* (2010), a cause-consequence model of the monitored system is exploited to reduce the number of released alarms. In that model every potential alarm is annotated and recognised as precursor, causal or consequential. In the presence of a disturbance a number of corresponding alarms would be released. According to the annotation of each alarm, operators treat consequent alarms as unimportant, whereas precursor and causal alarms are treated as important. Figure 2-13 shows an example of a cause-consequence model.

When the feed-water pump "FWBP-P04" fails, the following alarms would be raised: (a) a causal alarm "FWBP-P04 Lube-Oil-Press-LoLo (P2)"; (b) precursor alarm "FWBP-P04 Trip (P2-P3)"; (c) two consequential alarms "RCPS-ACTu(P1-P2)" and "MFWP-P01 Trip (P2-P3)". In this case, the important alarm would be the causal one "FWBP-P04 Lube-Oil-Press-LoLo (P2)", while others (as annotated) are consequent or precursor alarms (Lee *et al.*, 2010).

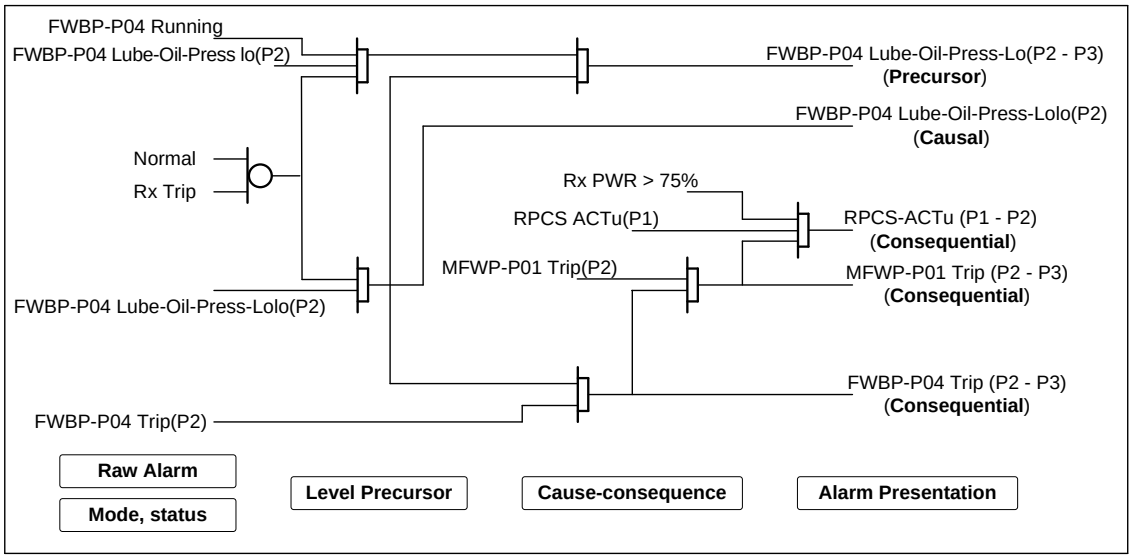


Figure 2-13: An Example of Different Alarms in a Cause-consequence Model (Lee *et al.*, 2010).

Despite the clear classification and distinction among the different types of alarms, this technique requires operators to achieve meticulous scanning before deciding on the corrective actions. This might impose extra workload that could potentially overwhelm and confuse the operators, particularly during emergency conditions.

2.3.2.2 Sensory Measurements Validation

Similar to the other system components, sensors have their own failures in which they exhibit abnormal behaviour. Practically, sensory failures result in the delivery of spurious sensory measurements to either or both automated controller and the alarm technique of the monitored system. In addition to affecting the automated controlling process of the monitored system,

spurious measurements could conceivably precipitate the release of false alarms that could confuse and mislead the operators. Thus, it is crucial to the effectiveness of the alarm technique to consider sensory failures.

Sensory failures can be classified into two types. The first is coarse failures in which the faulty sensor delivers measurements that lie outside the range of expected measurements. Such faults can be easily detected by setting up a validation mechanism according to a declared range. The second type is subtle failures in which the faulty sensor delivers spurious measurements that do not violate the expected range, e.g. sticking at a certain measurement and delivering non-linear drift or biased measurements. Such failures are accordingly hard to detect (Papadopoulos and McDermid, 2001).

To validate sensory measurements, detect and diagnose faults of sensors, Yu and Su (2006) developed a technique in which sensors of the monitored nuclear power plant are decomposed into a number of sub-groups. An important issue in the decomposing process is that each sensor is assigned to at least two sub-groups and no sensor should be assigned more than one time to any sub-group. Within every sub-group and based on the relations among the sensors, the consistency among the measurement is judged by Multivariate State Estimation Technique (MSET) and Sequential Probability Ratio Test (SPRT) [1].

If the measurements of a sub-group are consistent then that indicates all the measurements are valid; there is no faulty sensor in the sub-group. On the contrary, an inconsistency indicates that the sub-group is not valid and there is at least one faulty sensor. As every sensor is a member of more than one sub-group, the inconsistency should appear in the other sub-groups that incorporate the faulty sensor and should not appear in those which do not incorporate it. Thus, the faulty sensor can be identified, as the common member among the inconsistent sub-groups. Based on the sub-grouping notion Yu and Su (2006) have developed two validation techniques; the Sub-group Consistency check (SCC) to detect single sensory failures and Sub-group Voting (SV) to detect multiple sensory failures. Figure 2-14 illustrates the sub-grouping of the SCC technique. In that illustration there are nine sensors decomposed into five sub-groups.

During the validation process MSET may make three different types of judgements on the measurements of a sub-group, as follows:

- Inconsistent sub-group, when one of the incorporated sensors delivers a spurious measurement.

[1] More information about MSET and SPRT can be found respectively in Cheng and Pecht (2007) and Chang (2004). More information on how they interact to the interest of this technique can be found in Yn and Su (2006).

- Consistent sub-group, when no spurious measurement is delivered by the incorporated sensors.
- Undetermined sub-group, when a measurement of one of the incorporated sensors is undelivered to the MEST.

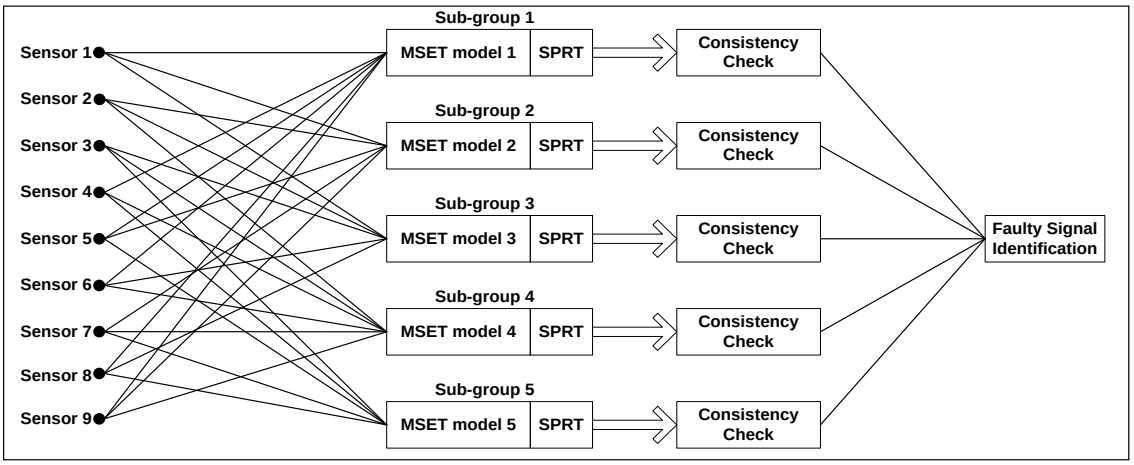


Figure 2-14: SCC Technique for Validating Sensory Measurements (Yu and Su, 2006).

With regard to detection of multiple sensory faults, Figure 2-15 illustrates the Sub-grouping Voting (SV) technique. Similar to the SCC, SV uses MSET to judge the sub-groups as consistent, inconsistent or undetermined. After a sub-group has been judged as inconsistent, faulty sensors are identified by a voting mechanism. This mechanism differentiates from SCC, where the faulty sensors are diagnosed through sub-group membership (inside the same sub-group).

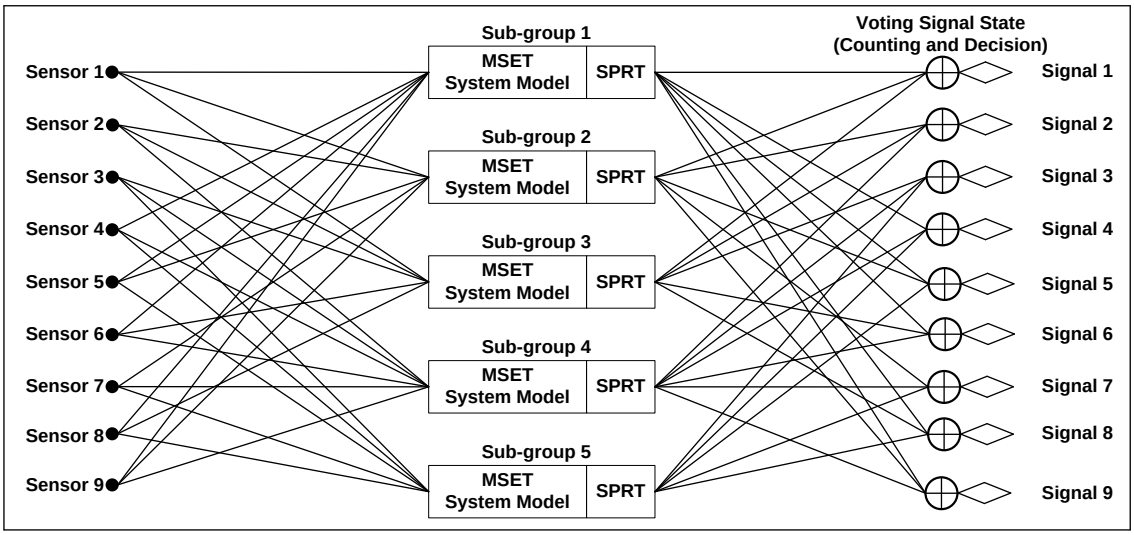


Figure 2-15: SV Technique for Validating Sensory Measurements (Yu and Su, 2006).

An observation of the effectiveness of sub-grouping validation reveals that the approach in both techniques (SCC and SV) delivers effective validation results when all the sub-groups are judged either consistent or inconsistent. However, if there are undelivered sensory measurements the entire sub-group would consequently be judged as undetermined. The presence of one or more undetermined sub-groups may result in failure to identify the faulty sensor and hence the validity of the sensory measurements cannot be judged. Moreover, Yu and Su (2006) have pointed out that, as the SV technique requires different types of sub-groups, its practical applicability is accordingly limited.

Baraldi *et al.* (2010a) developed a similar technique (but without sub-groups) to validate sensory measurements, and to detect and diagnose the sensory faults of nuclear power plants. In this technique, detective and diagnostic decisions are made by the Sequential Probability Ratio Test (SPRT). Knowledge to inform the SPRT reasoning is aggregated by Local Fusion (LF) strategy from principal component analysis of the local behaviour of the monitored sensors. More similar approaches with relative differences can be found also in Baraldi *et al.* (2009) and Baraldi *et al.* (2010b).

In earlier work, Kim *et al.* (1990, 1988) developed a Sensory Failure Diagnosis Tree (SFDT) to validate sensory measurement and diagnose the faulty sensors. The tree is originally built from a number of sensory validation criteria (SVCs) that probe to achieve the required validation and diagnosis. To illustrate the idea of this technique, let us consider the physical illustration of a flow line and its SFDT shown by Figure 2-16.

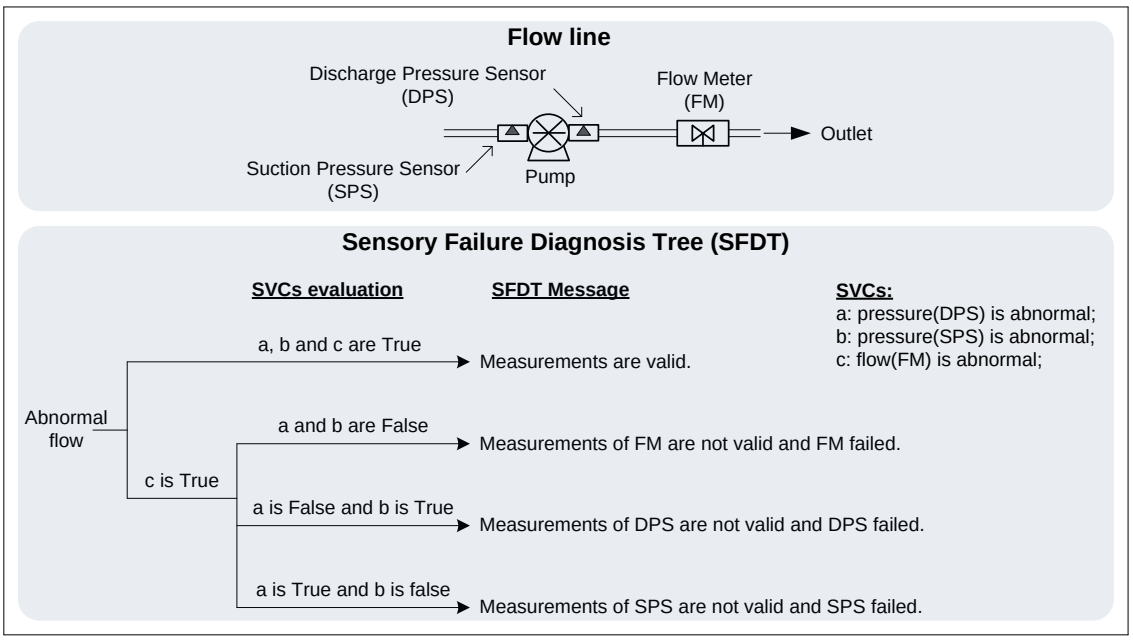


Figure 2-16: A Flow Line and its Sensory Failure Diagnosis Tree (SFDT).

The flow line incorporates three sensors: Suction Pressure Sensor (SPS), Discharge Pressure Sensor (DPS) and Flow Meter (FM) sensor. In order to validate the measurements of those sensors and diagnose the faulty among them, SVCs are formed in three criteria (as shown by SVCs column in Figure 2-16). After identifying the logic among those criteria, SFDT is constructed. During the monitoring time, criteria are evaluated according to their appearance in the SFDT and corresponding alarm messages are selected and announced to the operators.

The above discussed techniques depend on analytical redundancy among the variant sensors, in other words, the techniques depend on the relationships among the sensors of the monitored process. Dating back to the 1970s, measurement validation and faulty sensors diagnosis were dependent on hardware redundancy techniques. Typically, techniques were implemented by deploying a number of redundant sensors to measure the functionality of the monitored system. By taking those measurements into a voting process, biased and spurious measurements are filtered out and true measurements validated (Clark, 1978).

Although hardware redundancy techniques offer adequate robustness, their applicability is limited since they tend to increase cost, weight and volume. On the contrary, analytical redundancy techniques have proved cost-effective and also do not entail increased weight or volume and so they are widely applied.

2.3.3 Alarm Prioritisation and Alarm Availability

Alarm prioritisation can be defined as the process in which alarms are given annunciation priorities according to their relative importance. Alarm availability can be defined as the process in which an alarm is selected according to its priority to be presented to the system operators at the correct time. In the case in which two or more alarms are valid to alert the operators on the occurrence of a fault, each of them is given a different priority so that with the highest priority is announced and emphasised to attract the attention of the operators. The other, however, should be de-emphasised or delayed until the proper time (Brown *et al.*, 2000).

In prioritising the alarms of the nuclear power plants, it is important to consider the following four factors Hickling (1994):

- Safety consequences.
- Urgency.
- Relevance to the task in hand.
- Productivity consequences.

In the same work, Hickling (1994) emphasised that the safety consequences is the most important factor, so highly threatening alarms should have the highest priority. In a similar

work, O'Hara *et al.* (2002) emphasised that urgency in association with the safety consequences are the most important factors, so alarms that imply these factors should be given the highest priority.

Relevant literature has identified two different strategies typically taken to prioritise alarms in the control rooms of plants and cockpits. The first is dynamic prioritisation, which depends on presenting alarms with different format according to their priority. The second strategy is group-presentation prioritisation, which depends on presenting alarms according to the hierarchical architecture of the monitored process and the importance of the relevant functions.

2.3.3.1 Dynamic Prioritisation

Dynamic prioritisation is generally implemented by three different strategies. Firstly, presenting alarms in different colour codes (Pirus, 1996; Lee *et al.*, 2010). The colour spectrum ranging from red and amber to magenta is the most common for this purpose, with red consistently signifying the highest priority alarms and amber and magenta lower priority alarms. Colours coding may extend to signify the time intervals during which the operators should respond to the alarm. The time that is required to respond to a red alarm is, for instance, between 5 and 15 minutes[1] (Brown *et al.*, 2000).

In the second strategy the differentiation among alarms' importance is achieved by presenting them according to their associated severity. Alarms are prioritised from the highest to the lowest priority as catastrophic, critical, marginal and insignificant. Similar to the above, the response time could also be implied with such priorities (O'Hara *et al.*, 1997).

In the third strategy, alarms might be prioritised by announcing the high priority alarms and suppressing or hiding the less priority. In some cases, lower priority alarms should be facilitated as optional-access alarms that can support the operators with further information on the conditions (Brown *et al.*, 2000).

It is also common to combine among the above three strategies to develop effective alarm techniques. Figure 2-17 shows an example of a prioritised alarm message of the Engine Indication and Crew Alerting System (EICAS) of the Boeing 747-441.

[1] Actions that should be taken within less than 5 minutes are usually taken automatically (Brown *et al*, 2000).

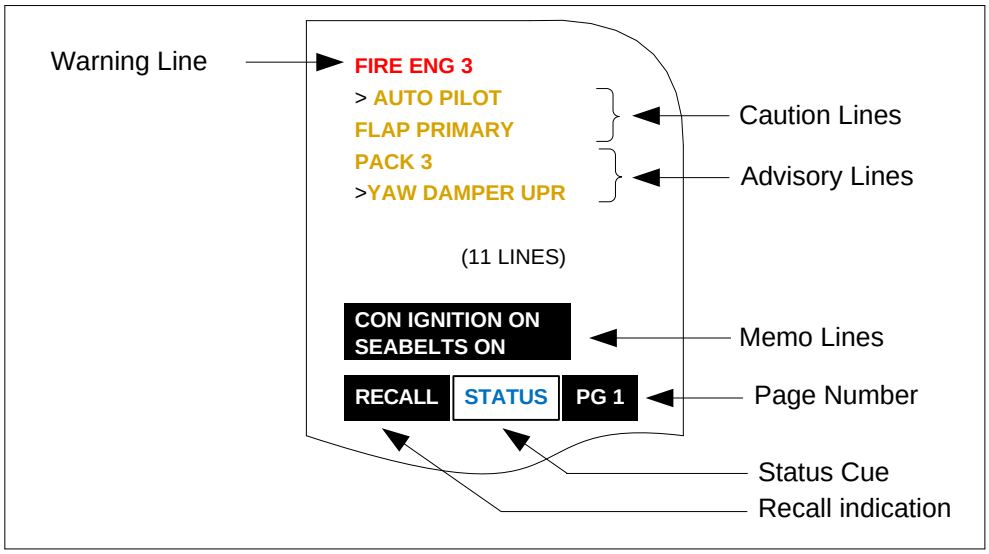


Figure 2-17: An alarm message of EICAS (Boeing, 2000).

The title of the warning message (warning line) appears in red due to its high priority and similarly the rest of the lines appear in different colours according to their priorities. The status cue signifies whether there are more warning messages and the page number signifies the number of hidden (not presented) pages that belong to the presented message.

2.3.3.2 Group-presentation Prioritisation

Taking advantage of the facility of computer-generating text, alarms are organised in groups and presented through screen (CRT or LCD) windows according to the hierarchical architecture of the monitored process and the importance of the relevant functionality. The main concern here is with presenting a small number of important alarms and facilitating access to further information and less important alarms (Roth and O'Hara, 2002).

Figure 2-18 shows an example of an alarm message window (alarm handler) presented according to group prioritisation. The window is divided into two main parts; an alarm tree structure shown in the left half and alarm group contents display shown in the right half. Additionally, at the top of the window there are two bars, the title bar and menu bar, and the lower part of the window shows a message area (Anderson, 2007).

The alarm tree structure is displayed as a hierarchy of grouped alarms. The contents of the selected alarms are displayed in more detail. The role of the menu bar is to give the required options by which operators can access information about other functions. The message area shows information about the presented function; whether it is a local or a global function and whether it is currently active or passive. A local function is one that is provided by a sub-system, while a global function is one that is provided collaboratively among a number of sub-

systems. This area facilitates also silencing alarms' sound or suppressing the illuminated alarms' lights (Anderson, 2007).

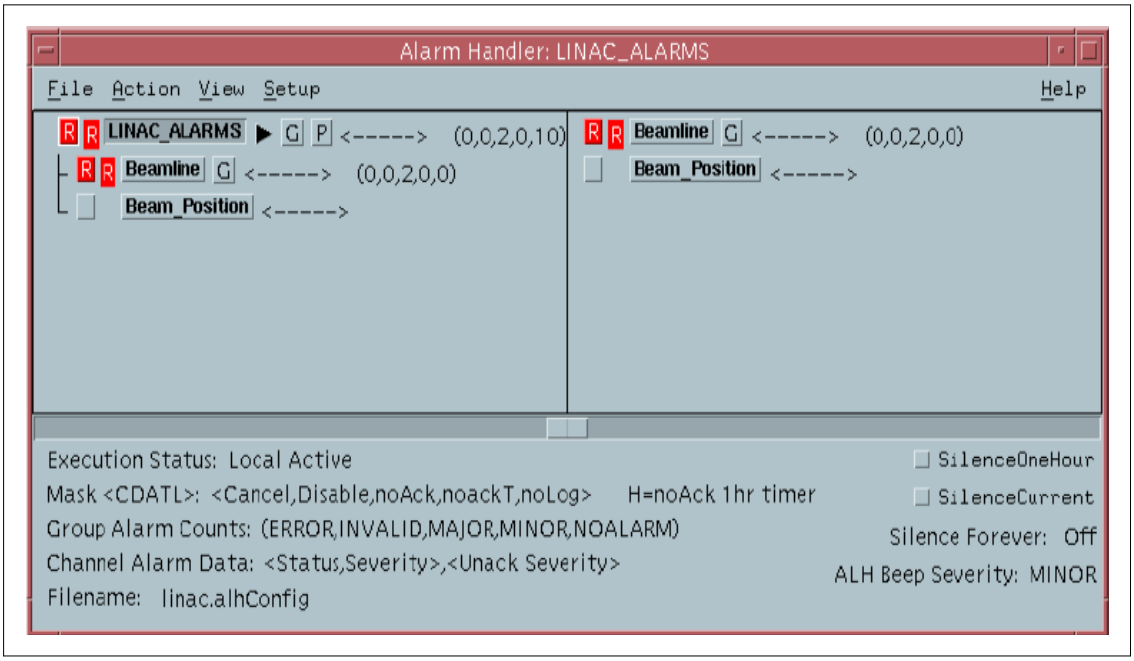


Figure 2-18: An Alarm Handler Window (Anderson, 2007).

2.4 Fault Controlling

Thus far, different techniques of fault detection and diagnosis and alarm annunciation have been discussed. In this section, fault controlling is discussed as the third and final safety task.

Faults of critical systems often result in disastrous failures if they are left unattended or controlled incorrectly. The industrial field has witnessed a number of accidents in which blame was ascribed to the human operators who either acted incorrectly or inadvertently failed to take the necessary actions. Automated fault controlling techniques, therefore, have been extensively researched to support the operators and avoid such scenarios (Jiang, 2005).

Fault controlling is always considered in alignment with the domain of the complex controlling process and there are two different approaches to achieve real-time fault controlling. Manual interference of the system's operators is one of those approaches. In this approach, in addition to the need of an advanced alarm scheme, the operators should be fully trained and provided with the required guidance on controlling faults (Patton, 1997; Seo and Kim, 1996).

The other fault controlling approach is achieved automatically by a computerised controller. Such a controller is most commonly called a Fault-Tolerant Control System[1] (FTCS) (Srichander and Walker, 1993; Lopez and Sarigul-Klijn, 2010). FTCS, in turn, can be classified

[1] In some literature, though, it is also called a reliable control system (Veillette *et al*, 1992).

into Active Fault-Tolerant Controlling (AFTC) and Passive Fault-Tolerant Controlling (PFTC) (Zhang and Jiang, 2008). Figure 2-19 shows a general classification of fault control approaches.

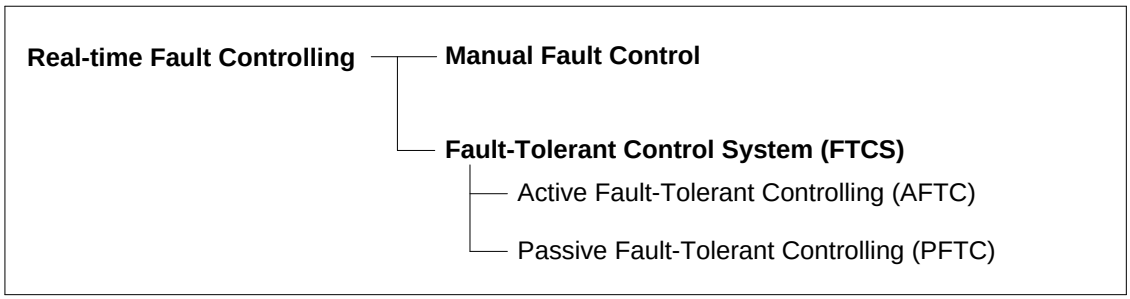


Figure 2-19: A Classification of the Fault Control and Correction Approaches.

2.4.1 Manual Fault Controlling

The manual fault controlling that is conducted by the crew pilots of aircraft is one of the prominent examples of un-automated fault controlling. Pilots are provided with a quick reference handbook as guidance on controlling potential failures. Based on the alarm and the feedback that are provided by the monitoring instrument of the flight deck, the pilots are trained to initially relate the abnormal condition to the corresponding control (remedial) procedure. They are also trained to apply the correct procedure with vigilant monitoring of evolutionary conditions (Holder, 2003).

For instance, the loss of electrical power (Alternating Current) of the left and right buses of a Boeing 767 aircraft would result in illuminating the "AC BUS OFF" light and a message being presented by the Engine Indication and Crew Alerting System (EICAS), as shown in Figure 2-20. In response to this failure, the crew pilots would open the quick reference handbook to select the corresponding checklist and control procedure.

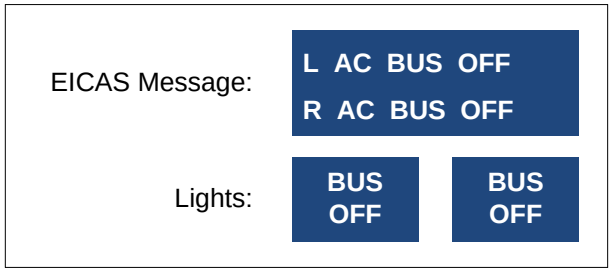
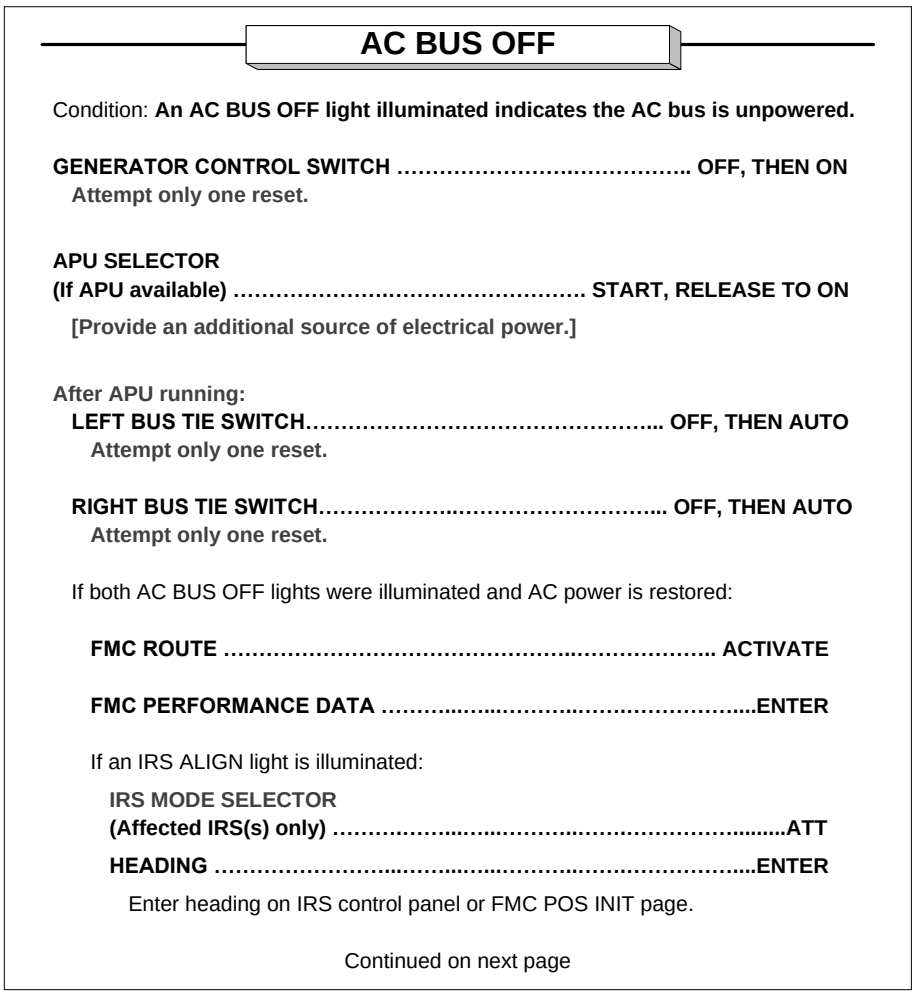


Figure 2-20: EICAS Message and Alert Light (Holder, 2003).

Figure 2-21 shows the corresponding checklist page of the reference handbook. To make sure that the selected procedure is able control the failure, the pilots apply the required actions and at the same time monitor the conditions through the EICAS and the alarm lights. One of the lines

in that page shows that if the Auxiliary Power Unit (APU) is available "If APU available" it should be started. After starting the APU, two actions should be taken; the first is "LEFT BUS TIE SWITCH, OFF, THEN AUTO" and the second is "RIGHT BUS TIE SWITCH, OFF, THEN AUTO". If, following these measures, the "AC BUS OFF" lights are still illuminated then further actions will need to be taken.



AC BUS OFF

Condition: **An AC BUS OFF light illuminated indicates the AC bus is unpowered.**

GENERATOR CONTROL SWITCH …….. OFF, THEN ON
Attempt only one reset.

APU SELECTOR
(If APU available) …….. START, RELEASE TO ON
[Provide an additional source of electrical power.]

After APU running:
LEFT BUS TIE SWITCH……... OFF, THEN AUTO
Attempt only one reset.

RIGHT BUS TIE SWITCH……... OFF, THEN AUTO
Attempt only one reset.

If both AC BUS OFF lights were illuminated and AC power is restored:

FMC ROUTE ……. ACTIVATE

FMC PERFORMANCE DATA ……...ENTER

If an IRS ALIGN light is illuminated:

IRS MODE SELECTOR
(Affected IRS(s) only) ……...ATT
HEADING ……...ENTER

Enter heading on IRS control panel or FMC POS INIT page.

Continued on next page

Figure 2-21: A Page of the Quick Reference Handbook of Boeing 767 (Holder, 2003).

Taking advantage of features that become available through electronic alarm annunciation, several techniques have been proposed to automate the notion of the quick reference handbook and reduce the workload of pilots and ultimately achieve effective fault control. Hill (1993) has defined three questions, which can be arisen from the occurrence of multiple dependent failures, to be addressed prior to the development of such a technique:

- The first question is how to provide the operators with control procedures that do not contradict each other in the case of the synchronous occurrence of multiple failures. For example, the two conflicting procedures that cure the faults of high fuel temperature and

engine stall of an aircraft. While the former requires increasing the thrust of the linked engine to aid cooling, the latter requires reducing the thrust.

- The second question is how to raise the operators' awareness of the combined results of applying more than one corrective procedure. Such a case might be faced when two or more faults occur in a temporal sequence. For example, when one of the two packs of an aircraft overheats, the corrective procedure then requires shutting the pack down. Similarly, the shutdown procedure should also be applied if the second pack overheats. In this case the cabin of the aircraft would be left without air pressurisation and temperature conditioning and an emergency descent should be launched.
- The last issue is how to raise the operators' awareness of the consequential (or dependent) results of applying different corrective procedures. For example, when an aircraft depends for conditioning its cabin on one pack as the second one has been shut down due to an overheat fault, the occurrence of a bleed air leak failure in the engine on which the operative pack depends should be cured by cutting off bleed air. That would result in no air supply to the operative pack and the cabin would consequently be unpressurised and, again, an emergency descent should be launched.

In the same work, Hill (1993) has proposed an approach that can address the above questions and automate the notion of the quick reference handbook. In that approach the corrective procedures are (a) prioritised through exploiting a model of Goal Order Search Tree (GOST); (b) arranged in a way that does not imply contradictions through exploiting a network propagation algorithm that counts the contradictions among the goals in the GOST; (c) provided to the pilots as electronic guidance through the Central Warning System (CWS[1]).

Figure 2-22 shows an excerpt of the GOST of an initial configuration of an aircraft. GOST consists of a number of nodes, arrows and rules (logic gates) arranged in several levels. Nodes represent component statuses and critical functions of the aircraft. Arrows link the nodes to each other through rules (AND or OR) and show the paths in which statuses of the components and functions are propagated across different levels. Two types of arrows are used, the solid arrows representing direct propagation and the dashed arrows representing the inverse (not propagation). Every node encloses three attributes which are illustrated as follows:

(v, p, d)

Where v: is the value that shows whether the represented state is achieved or not;
p: is the priority that represents the importance of the given component or function in the overall functionality.
d: is the denied priority that represents the maximum priority of the state when it is not achieved and as required by the optimum configuration.

[1] In modern aircraft the CWS has different names. For example, in the Airbus A320 it is called Electronic Centralised Aircraft Monitor (ECAM) and in the Boeing 757 and 747-400 it is called Engine Indicating and Crew Alerting System (EICAS) (Hill, 1993).

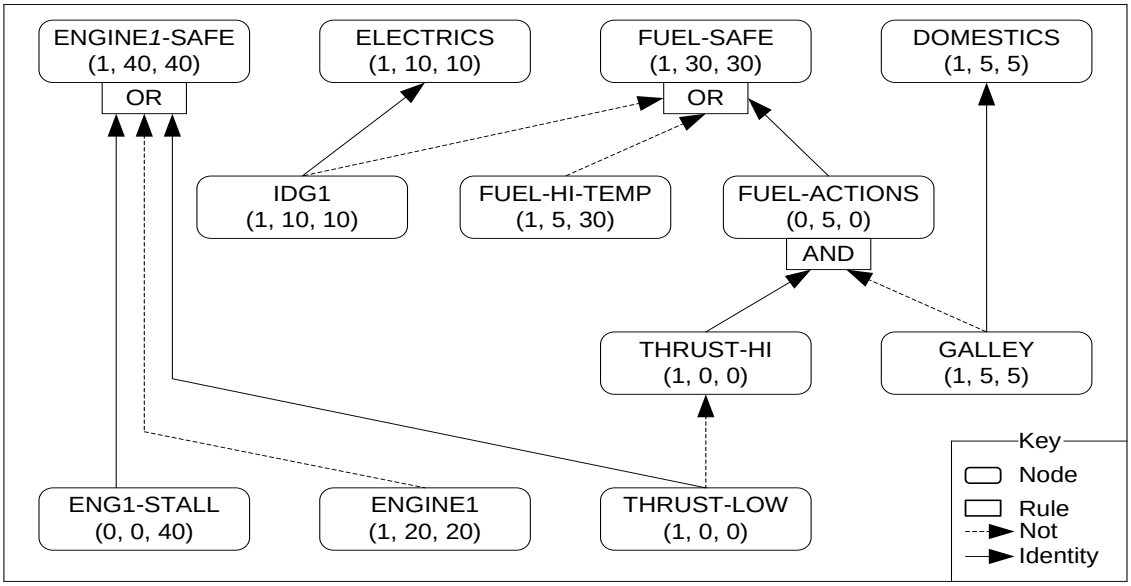


Figure 2-22: An Excerpt of the GOST of Initial Configuration of an Aircraft (Hill, 1993).

In Figure 2-22, GOST incorporates goals ENGINE*1*-SAFE, ELECTRIC, FUEL-SAFE and DOMESTICS, which are associated with the respective attributes (1, 40, 40), (1, 10, 10), (1, 30, 30) and (1, 5, 5). The occurrence of a high fuel temperature, for instance, is represented by the node FUEL-HI-TEMP in which the value (v) of the enclosed attributes would be changed from 0 to 1, consequently. This fault may occur due either to there being extra load on the Inertial Drive Generator (IDG) or to there being insufficient cooling fuel.

Fuel overheating can be controlled either by disconnecting the IDG or by switching the galley off to reduce the load on the IDG and increasing the thrust to increase the flow of the fuel. According to the associated priorities and functional dependability, switching the galley off is the best corrective action. After achieving that action the associated attributes of the relevant nodes would be calculated and updated according to the new configuration.

Nodes whose attributes would be changed are DOMISTIC, FUEL-HI-TEMP, FUEL-ACTIONS, THRUST-HI, GALLEY, ENG-STALL and THRUST-LOW to become respectively as follows: (0, 10, 0), (1, 50, 50), (1, 10, 30), (1, 10, 30), (0, 10, 30), (0, 10, 40) and (0, 10, 30). As such, the up-to-date priorities would contribute similarly to the decision on the best procedure to control future faults (Hill, 1993).

Despite the attractive control abilities of exploiting GOST, Hill (1993) has also identified a number of limitations. As the aircraft system has a number of different flying phases, the calculation of priorities must take the impact of those phases into account. Another limitation is that as the value (v) of the nodes' attributes is a binary value, the presentation of the statuses of functions and components is restricted to two digital values (1 or 0). This does not align with the need of analogue values in different cases, e.g. presenting the thrust level. The last limitation

is the lack of identification of the transitory actions that are applied to obtain temporary statuses; such statuses are needed to serve specific controlling purposes and would be changed thereafter.

2.4.2 Fault-Tolerant Control System (FTCS)

Fault-Tolerant Control Systems (FTCS) can be classified into Active Fault-Tolerant Controlling (AFTC) and Passive Fault-Tolerant Controlling (PFTC). Both AFTC and PFTC assume that faults are already known and taken into controlling consideration during the design of the controller of the monitored system. Accordingly, the controller would be able to deal with the real-time presence of those faults and maintain the safety of the controlled system (Jiang, 2005).

2.4.2.1 Active Fault-Tolerant Controlling (AFTC)

Research on the AFTC has been motivated by the aircraft flight control system. The essential goal of that research is to add self-repairing capability by which an aircraft could achieve its mission or land safely in the presence of faults (Lopez and Sarigul-Klijn, 2010). According to Jiang (2005), AFTC might also be called reconfigurable controlling, self-repair controlling or restructure-able controlling.

Figure 2-23 shows an illustration and the main components of the AFTC. The typical AFTC consists of four basic components: (a) reconfigurable controller; (b) fault detection and diagnosis; (c) reconfiguration mechanism; (d) command/reference governor. The reconfigurable controller is a digital controller whose structure and parameters are directed by the reconfiguration mechanism. The reconfiguration mechanism, in turn, is responsible for scheduling the work scenario of the monitored system. In order to support the reconfiguration mechanism, the command/reference governor and the system operators could also inject or adjust control commands (Lopez, Sarigul-Klijn, 2010; Jiang, 2005; Zhang and Jiang, 2008).

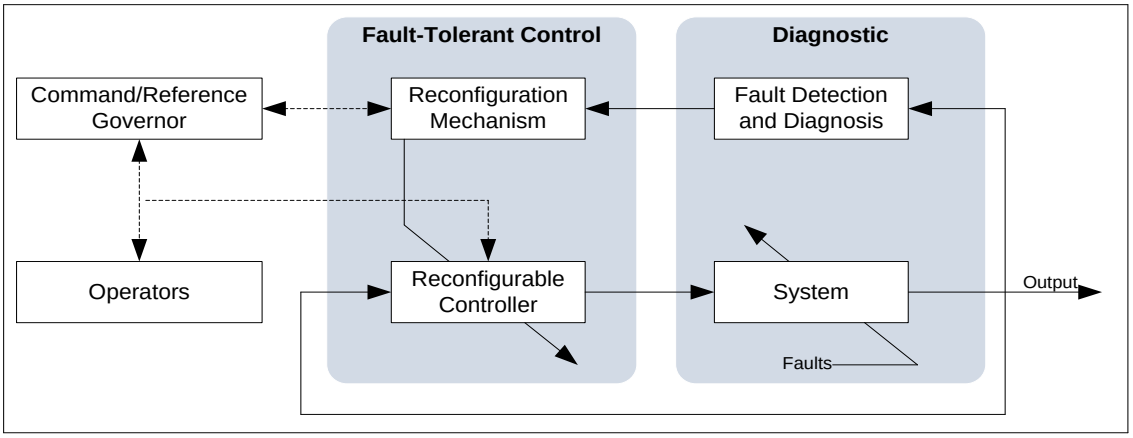


Figure 2-23: An Illustration of AFTC (Lopez, Sarigul-Klijn, 2010).

In real time, AFTC always achieves fault controlling by applying the corresponding corrective procedures. On the occurrence of an engine fault of a two-engine aircraft, for example, the corrective procedure includes the following actions: (a) cut-off fuel flow to the faulty engine; (b) achieve cross feed from the tanks that were feeding the faulty engine; (c) apply the corresponding command movements to control the surface and compensational instructions to the operative engine (Delta Virtual Airlines, 2003).

One of the drawbacks that have been highlighted in applying the AFTC is that there is some undesirable time delay in controlling highly critical faults and such delay may pose serious risks. Additionally, due to the involvement of a number of computational components, AFTC might also be subject to application of incorrect controlling decisions (Jiang, 2005).

2.4.2.2 Passive Fault-Tolerant Control (PFTC)

PFTC relies mainly on redundant components, such as multiple control computers and backup sensors and actuators (Zhao and Jiang, 1998). The idea of the PFTC is based on providing the pivotal components with redundant copies and multiple disjoint controllers that activate the redundant copy and deactivate the primary component whenever it fails (Heimerdinger and Weinstock, 1992).

Typically, provision of redundant components is implemented by hot or cold standby redundancy. In hot standby redundancy, the system is provided with parallel redundant components which operate simultaneously (powered up). Each component monitors the output of the other(s). Should any of them fail, the others take over (Shooman, 2002). Figure 2-24, illustrates an example of two parallel redundant components.

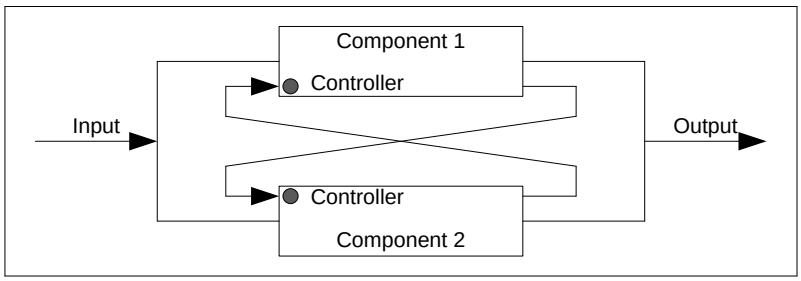


Figure 2-24: An Example of two Hot Standby Components.

In cold standby redundancy, only one component is on-line (powered up) and other copies are on standby (powered down). Should the on-line component fail, it is powered down and one of the standby components is powered up by a controller (Shooman, 2002). Figure 2-25, illustrates an example of two cold standby components.

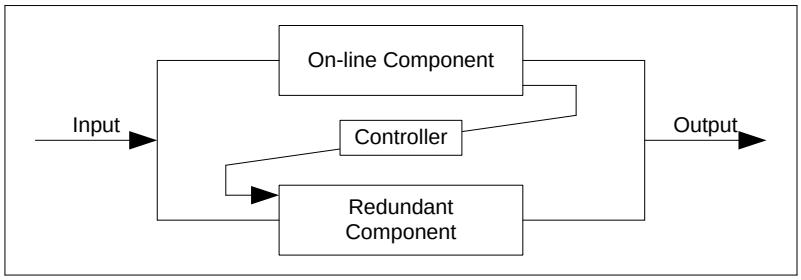


Figure 2-25: An Example of two Cold Standby Components.

Similar to the AFTC, PFTC also has drawbacks which have been identified as the extra cost and space needed to add the redundant components. Additionally, providing systems with redundant components, particularly the means of transportation, may simply add extra weight and impair the system's performance. Thus, the idea of a hybrid FTCS has also been exploited. In the hybrid FTCS the AFTC and PFTC are exploited together based on the advantages offered by each (Jiang, 2005).

2.5 On-line Safety Monitor

This section discusses on-line safety monitors which are developed to deliver some or all of the safety tasks that have been discussed so far: fault detection and diagnosis, alarm annunciation and fault controlling.

Industrial experience evidences that safety monitors play a key role in the safety of critical system (IAEA, 2008; Papadopoulos, 2003; Zolghadri, 2002). In the avionics industry, it has become widely accepted that on-line monitors contribute to reduce pilot errors that may occur during and due to emergency conditions. Prompt fault detection and providing the pilot with informative alarm can improve the ability of pilots to take correct and on-time controlling actions. Fault controlling techniques effectively reduce the pilots' workload and thus they are allowed enough time to plan and decide on more important tasks, e.g. an emergency landing (Zolghadri, 2002).

Similarly, in the operational safety of nuclear power plants, on-line safety monitors have demonstrated a number of safety merits, namely (a) detecting the abnormality of a wide range of on-line conditions, such as vibration, acoustic and loose parts; (b) diagnosing the underlying causes of reactor noise and the upset of the components; (c) early warning of imminent failure or component degrading and thus reduce the maintenance cost (IAEA, 2008; Hashemian, 2011); (d) achieving automated on-line calibration of the important transmitters, such as levels, pressures and flows, of nuclear power plants, and this can effectively reduce the radiation exposure of the operators and eliminate the consumption of manpower associated with manual calibration (Hashemian, 2004).

The typical development of monitors is generally from monitoring knowledge about the behaviour of the monitored system coupled with a number of reasoning algorithms. In the relevant literature, monitors are classified, based on their architectural nature into two concurrent classes: monolithic monitors and multi-agent monitors. Each of these two classes is classified further into model-based and data-based monitors, based on the type of the encoded monitoring knowledge. Figure 2-26 illustrates this classification.

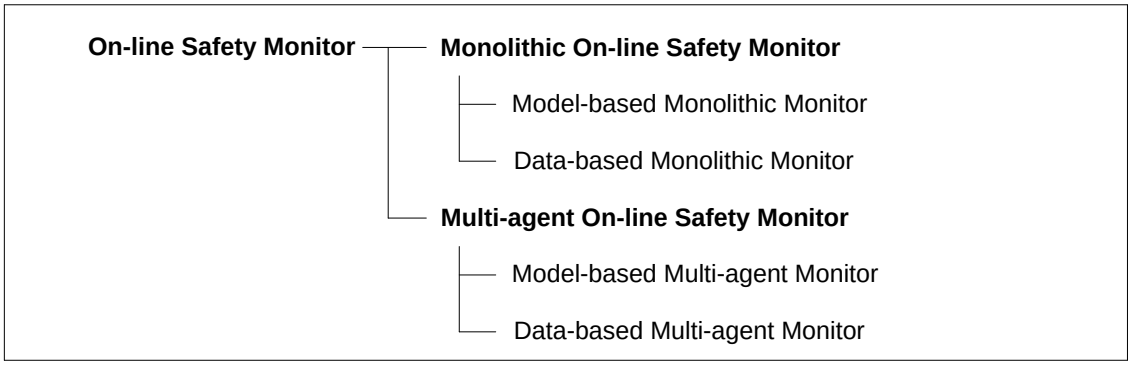


Figure 2-26: A Classification of the On-line Safety Monitor Approaches.

2.5.1 Monolithic On-line Safety Monitors

An example of the monolithic monitors is the one proposed by Papadopoulos (2003). The monitor delivers three safety tasks: (a) fault detection and diagnosis; (b) alarm annunciation; (c) fault controlling. The monitor itself has been developed from model-based monitoring knowledge and three generic mechanisms: event monitor, diagnostic engine and event processor. Figure 2-27 illustrates an architectural view of the monitor and its components.

Monitoring knowledge, or as it is called by Papadopoulos (2003) the monitoring model, is constructed of two constituents. The first is a hierarchy of state-machines that holds the normal and abnormal behaviour of the system and its sub-systems. The second constituent is a diagnostic model, which is a number of fault trees that relate possible functional failures to their underlying causes. Synthesis of those fault trees is achieved by a semi-automated safety assessment technique called Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) (Papadopoulos *et al.*, 2001).

The main role of the event monitor is to detect and identify three types of events: failure events, normal events and corrective events. The event monitor achieves its role through the continuous monitoring of a list of events (parameters) after instantiating them with up-to-date sensory measurements. After the occurrence of an event, the event processor (a) examines the impact of that event on the state-machines of the system and its sub-systems; (b) updates the event list accordingly.

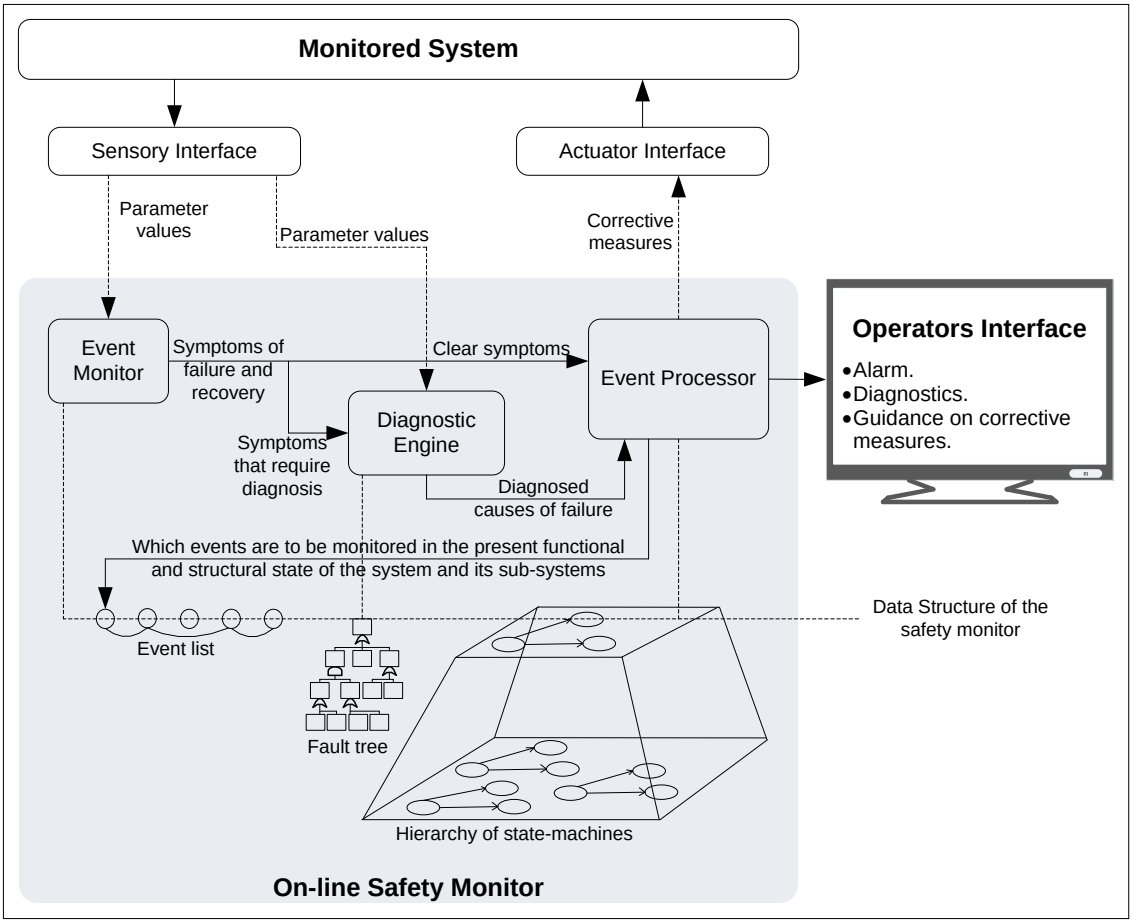


Figure 2-27: Architectural View of the Safety Monitor Developed by Papadopoulos (2003).

For diagnosing the underlying causes of the failure events, the diagnostic engine traverses the relevant fault trees, whose top event encloses the detected failure. The trees are traversed top-down and at the same time their encoded symptoms are evaluated to track the correct propagation path and eventually diagnose the underlying causes.

In a quite similar approach, Peng *et al.* (2007) have also proposed a monolithic monitor that exploits the three mechanisms: event monitor, diagnostic engine and event processor. The monitor also exploits model-based monitoring knowledge to reference the monitoring reasoning. However, this monitor differs from the one proposed by Papadopoulos (2003) in replacing the hierarchy of the state-machines with the control chart of the monitored system. By this replacement the potential of applying conflicted corrective measures in the monitor of Papadopoulos (2003) has been eliminated. Figure 2-28 illustrates an architectural view of this monitor and its components.

The motivation for the above two monitors was the observation that, in modern industrial practice, vast amounts of knowledge derived in off-line safety assessments cease to be useful following the certification and deployment of a system. Thus, a key contribution of these

monitors is that they bring this knowledge forward to the operational phase of a system and usefully exploit it for developing cost-effective on-line safety monitoring.

Monolithic monitors have also been developed from data-based monitoring knowledge. Doan and Srinivasan (2008) have developed a data-based monolithic monitor to monitor a multi-phase process of a chemical plant. Knowledge about the potential behaviour of the plant is extracted from the process historian database by multivariate statistical process control and principal component analysis. By reasoning among the initial conditions and event timings and the extracted knowledge, the monitor detects, diagnoses, and reports abnormalities.

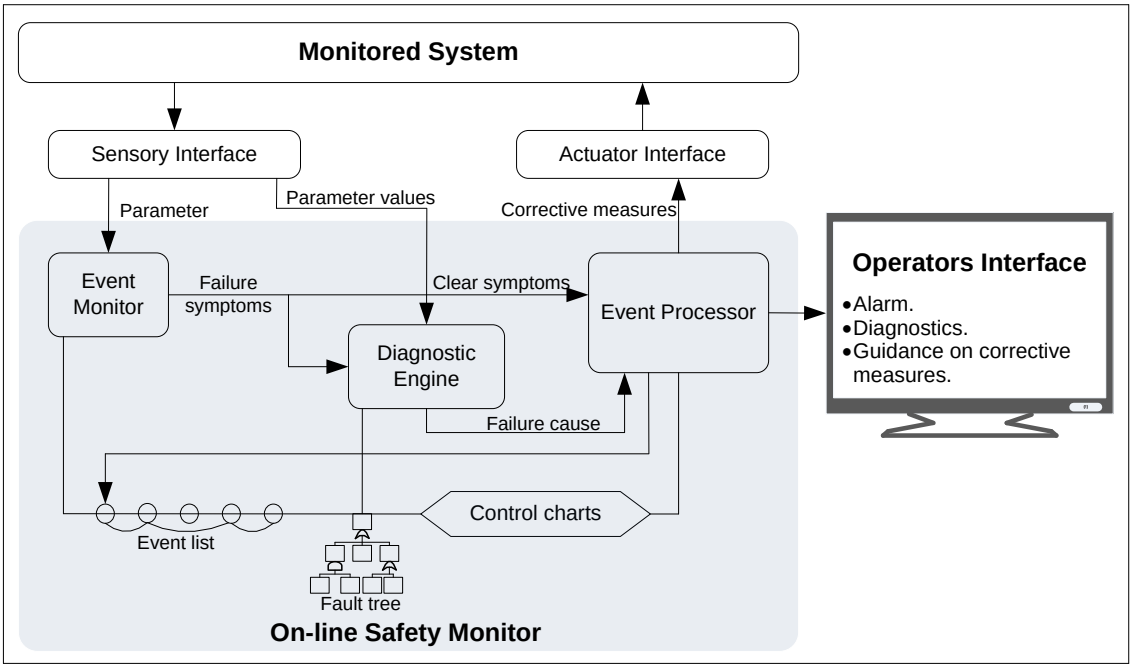


Figure 2-28: Architectural View of the Safety Monitor Developed by Peng *et al.* (2007).

The significant limitation that can be identified with the monolithic monitors is a consequence of their monolithic concept in which all the components of the monitored system are delegated to a single reasoning monitor. This does not align well with the distributed and complicated nature of most modern systems, which typically incorporate a huge number of components that work collaboratively in complex cooperative structures.

2.5.2 Multi-agent On-line Safety Monitors

Considering the distributed reasoning paradigm and thus developing distributed on-line safety monitors appears as the best turn to address the limitation of the monolithic monitors. Multi-agent systems have been proposed and researched, as a means to develop these monitors. In addition to incorporating a multi-agent system, monitors are also developed from model-based and data-based monitoring knowledge.

Among the model-based multi-agent monitors are the one developed by Ren *et al.* (2006) and that developed by Eo *et al.* (2001, 2000). In the former monitor, the multi-agent system is arranged in two levels, lower level and higher level. At the lower level a number of monitoring agents are deployed and at the higher level a supervisor agent is deployed. The lower level agents monitor the parameters of the sub-systems. To achieve that role, each agent is provided with a corresponding portion of the monitoring knowledge to inform its local reasoning. Every portion consists of a functional model augmented with operators' expertise. At the higher level, the supervisor agent is provided with a different behavioural model, the Markov model[1]. This agent reasons qualitatively on that model and collaborates with the lower level agents to reach global diagnostic decisions. Collaboration between the agents of the two levels is achieved according to an especially developed protocol.

The monitor developed by Eo *et al.* (2001, 2000) consists of a number of functional behaviour models and monitoring agents. Agents are deployed to monitor the functions of the monitored system locally and collaborate with each other to achieve global diagnostic decisions. During the monitoring time, the real-time reasoning of every agent is informed by its corresponding portion of the functional behaviour model. For each function the model records different states and each state, in turn, holds details about the function in a certain mode.

Among the data-based multi-agent monitors are those developed by Ng and Srinivasan (2010) and Niu *et al.* (2007). In the work of Ng and Srinivasan (2010) the monitor is developed to detect and diagnose faults of chemical processes. To deliver this task a number of agents are deployed over the monitored process. Each agent monitors the entire process and reasons locally by exploiting one of the following reasoning methods: self-organisation maps, principal component analysis, neural network or non-parametric approaches. Moreover, to eliminate the possibility of producing conflicting local detection and diagnosis results, agents are also provided with decision fusion methods, which are voting-based fusion and Bayesian probability fusion, and a collaboration protocol to reason globally and reach consensus decisions.

In a slightly different monitoring idea, the monitor of Niu *et al.* (2007) is developed from a multi-agent system, fusion algorithm and pattern recognition techniques. Agents are deployed over the monitored process; each acts as a classifier and achieves pattern recognition. Every agent monitors the entire process from different sensory data sources (from commensurate and non-commensurate sensors). By combining between the Bayesian and the method of majority voting, the agents reason among their sensory data and achieve local decision fusion on detecting and diagnosing faults of the monitored process. According to a dynamic fusion algorithm, agents also collaborate with each other to achieve global decision fusion and reach better fault detection and diagnosis decisions. To track the behaviour of the monitored process,

[1] More information about the Markov model can be found in Rabiner and Juang (1986).

every agent is also provided with a co-decision matrix from which they decide on the information that should be exchanged in different behavioural contexts, i.e. phases.

Despite the success of multi-agent systems in implementing distributed safety reasoning, two limitations have also been highlighted:

- The typical lack of effective collaboration protocols. Collaboration protocols are a pre-requisite for establishing integrated reasoning among the monitoring agents and thus delivering timely, consistent and effective monitoring tasks (Wallace *et al.*, 2011; Roos *et al.*, 2002 and 2003a,b).
- The logical omniscience problem. Splitting and distributing monitoring knowledge among a number of the monitoring agents may result in leaving them susceptible to monitoring conditions that fall beyond their provided knowledge and reasoning abilities (Sayda, 2011; Mangina, 2005).

2.6 Summary

This chapter has provided a refined outline for addressing the problem of on-line safety monitoring. It shows that monitoring critical systems and improving their safety can be realised through the delivery of three safety tasks: fault detection and diagnosis, alarm annunciation and fault controlling. Accordingly, in the context of this thesis the focus is on a developing a monitor that can deliver these tasks.

The chapter also considered whether the model-based or data-based monitoring approach is appropriate to develop the proposed monitor and concluded the following:

- Both approaches necessitate input from different engineering disciplines to construct monitoring knowledge that can support the delivery of the three safety tasks. The produced knowledge would tend, therefore, to be very expensive.
- Due to the involvement of different engineering disciplines, the produced knowledge is most likely to be inconsistent and thus the ultimate effectiveness of the developed monitor cannot be assured.

Taking advantage of the cost-effective monitoring ideas of Peng *et al.* (2007) and Papadopoulos (2003), the decision has been made to exploit safety assessment models as monitoring knowledge to support the real-time reasoning of the proposed monitor. Assessment models, especially those constructed on a model-based concept can inform the monitoring reasoning with thorough and consistent knowledge about the normal and abnormal behaviour of the monitored system and support the effective delivery of the three safety tasks.

In this thesis, the targeted safety assessment models are specifically those produced by the state-of-the-art safety assessment techniques[1]; Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) (Papadopoulos *et al.*, 2001) and Architectural Analysis and Design Language (AADL) (Feiler and Rugina, 2007). The validity of this exploitation is further investigated and demonstrated in the next chapter.

With regard to the architectural nature of the monitors and real-time reasoning, the chapter also reviewed, examined a number of monolithic and multi-agent monitoring concepts and concluded the following:

[1] It must also be noted that there is a plethora of similar techniques that produce assessment models to some extent similar to those produced by the HiP-HOPS and AADL. Consider, for example, Galileo-dynamic fault tree analysis (Dugan *et al.*, 1999; Sullivan *et al.*, 1999), Altarica (Bieber *et al.*, 2002; Bieber *et al.*, 2004) and FSAP/NuSMV-SA (Bozzano *et al.*, 2003; Bozzano and Villafiorita, 2003a,b).

- Due to the limited ability of monolithic monitors to cope with the distributed and complicated nature of modern systems, they thus have been excluded as a development option.
- The demonstrated success of multi-agent monitors makes them rather promising as an option for developing the proposed monitor.

Bearing in mind that the multi-agent systems suffer some monitoring limitations, namely, the lack of effective collaborative protocol and the logical omniscience problem, Chapter Four develops a multi-agent system that can address such limitations and achieve effective safety reasoning.

Chapter Three

Distributed On-line Safety Monitor: an Overview and Distributed Monitoring Model

3.1 Introduction

This chapter provides an overview of the proposed monitor and describes the way of organising the monitoring knowledge as an executable model. The chapter firstly introduces issues concerning the monitored system and states the exact meaning of terms that will be used in the rest of the thesis. It then proceeds to outline the overall position, role, architecture and constituents of the monitor. The development of the distributed monitoring model (as the first constituent of the monitor) is then discussed. In this context, formalisation issues are addressed and techniques to filter and validate sensory measurements and detect, diagnose and control sensory faults are developed.

3.2 Monitored System

Critical systems are sophisticated machines which typically have large numbers of software and hardware components and which exhibit complex and dynamic behaviour. A modern aircraft is an example of such systems; it involves hundreds of thousands of components and delivers a flight mission over a number of behavioural phases: pre-flying, taxiing, take-off, climbing, cruising, approaching, and landing.

Large scale and complex dynamic behaviour may call into question the effective delivery of monitoring tasks. Large scale means large volumes of monitoring data and many potential deviations from normal behaviour which may be difficult to detect and handle without errors or omissions. In systems that have many operational phases, it is often difficult to know when a particular condition should be considered normal or abnormal and what are the effects and appropriate remedial actions.

Thorough knowledge about the architecture and behaviour of the system is, therefore, necessary to achieve effective safety monitoring.

3.2.1 System Architecture and Operational Structure

Large-scale systems are mostly designed as a hierarchy of components that can themselves be systems (Pressman, 2001 and 1998). In a hierarchy, relationships among components are typically implemented as between a parent, child and siblings. For example, the components of an aircraft system include the flight control system, navigation system, power plant systems,

surface systems, landing gear systems, wide area augmentation system and global positioning system. In the hierarchical architecture, some of those components may appear as parents and children at the same time. The navigation system can be seen as a child of the flight control system and at the same time it is a parent of some child components such as the wide area augmentation system and global positioning system. Similarly, these systems may also have their own child components. Components - often at the same level of a hierarchy - also appear as siblings that work collaboratively to integrate and achieve different functions across different phases; e.g. power plant systems and surface systems.

In order to simplify the architectural view of the monitored system, a generic hierarchical view of the system's components is introduced here. As shown in Figure 3-1, in this view, components that appear at the lowest hierarchical level (level*0*) are classified as basic components (BC). Components that appear at levels extending from level*1* to level*n-1* are classified as sub-systems (Ss). Finally, the component that appears at the top level (level*n*) is classified as the system (S).

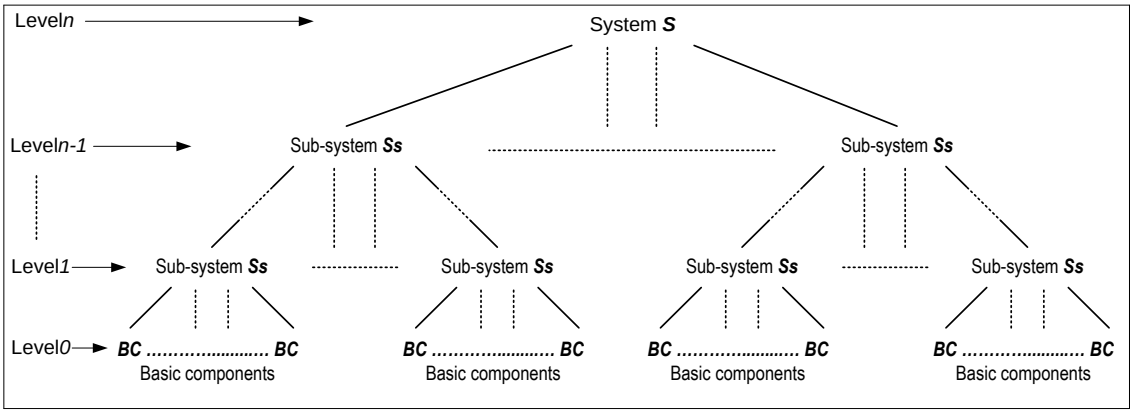


Figure 3-1: General Illustration of the Components Classification.

During operation, a system's architecture may appear in different topologies, in which some components might be engaged or not. For example, when an aircraft is in the cruise phase, the landing gears are not engaged, whereas they should be engaged during the pre-flight, taxiing and landing phases. The term *Operational structure* is commonly used to denote the topology in which architectural components are engaged.

3.2.2 Dynamic Behaviour

Regulating and guiding the behaviour of a dynamic system (or a phased-mission system) might require an automated controller. The controller takes sensory measurements into a computational process to produce controlling instructions. Instructions, in turn, are applied by the actuators (Romagnoli and Palazoglu, 2006).

Hence, normal behaviour is typically triggered by operators or initiated by some basic components (sensors) and applied to the environment of the system through other basic components (actuators). Transition from the cruise phase into the approach phase is triggered, for instance, by an event initiated by the navigation sensor. The navigation sensor is a basic component and it delivers a measurement to its parent, the navigator system. The navigator system, however, is not able to instruct the systems of the aircraft to instantiate approach phase. It rather calculates the measurement and signals the occurrence of that event to its parent component, the flight control system (FCS). The FCS then instructs its children, the body motion system and power plant system, which similarly instruct their lower level components until reaching the basic component (actuator) level. This interaction shows how the normal behaviour moves bottom-up and top-down across the hierarchical levels and triggers normal transition of system components. Events that trigger such normal transitions are referred to as *normal events*.

Abnormal behaviour is initiated by faults of the basic components. This behaviour can also move bottom-up and top-down across the hierarchical levels and trigger abnormal transition of the components of the system. For example, an aircraft fuel system may partially fail and feed only one of two engines due to a fault of a fuel pump (a basic component). The degraded functionality is propagated higher up to the parent component, the controller of the power plant system, which in turn safely shuts down the affected engine (a child) by switching the auto-throttle off.

As discussed in Chapter Two, fault controlling is typically implemented by active fault-tolerant control (AFTC) and passive fault-tolerant control (PFTC). Both types of fault-tolerant control require a while to allow the controller to apply the corresponding corrective measures and either resume the normal behaviour when the measures succeed in rectifying the conditions or transit to failure conditions otherwise (Marshall and Chapman, 2002). The intended functionality of the monitored system may appear, therefore, in different states that can be classified[1] as follows:

- *Error-free State (EFS):* in which the intended functionality is delivered perfectly.
- *Error State (ES)*: in which the system delivers none or only part of the intended functionality. Depending on the ability to recover from this state, ES can be classified further into the following three types of state:
  - *Permanent Degraded State (PDS)*: in which part of the intended functionality is lost and cannot be recovered, e.g. the state in which a two-engine aircraft is flying with a faulty engine.
  - *Failure state (FS)*: in which all functionality is lost and it cannot be recovered.

[1] States classification is based on the work of Feiler and Rugina (2007).

- *Temporary Degraded or Failure State (TDFS)*: in which part or all functionality is lost, but there are corrective measures that can be taken to resume an error-free state or in case of a further failure transit to a permanent degraded or failure state.

Events whose occurrences result in transitions from the error-free state to any error state are termed *failure events*. Events whose occurrences are outcomes of applying corrective measures and result in transitions from a temporary degraded or failure state to resume an earlier state are termed *corrective events*.

3.3 Basic Terms and Definitions

Discussing the safety monitor necessitates the employment of various safety terms. There is, however, no exact consensus on those terms (Joshi *et al.*, 2006). For the sake of clarity, this section states the meaning of key terms, as commonly understood by many researchers and as used in this thesis.

3.3.1 Failure, Error, Fault, Error Propagation and Symptom

A *Failure* is an event that results in delivering a deviant function (Joshi *et al.*, 2006). In the monitoring context, failure could be verified from deviations of one of more observable parameters. *Error* can be defined as a discrepancy between the actual and intended values of the output of a basic component. An erroneous output is caused by an internal *fault* of the component itself or as a consequence of an erroneous input that is *propagated* by another component (Vesely *et al.*, 2002). During its propagation, an error may expose itself as a *symptom*. Thus, a *symptom* can be defined as an event that follows an initiating component fault and precedes a system failure.

3.3.2 Phase, Mode and State

A *Phase* can be defined as a distinct operational structure of a system along with its own deliverable functionality, e.g. the flying phases of an aircraft system. Because the components of phased-mission systems may not contribute to all the phases of the system, the term *mode* is used to refer to the explicit operational structure and delivered functionality of the components (Feiler *et al.*, 2006; Feiler and Rugina, 2007). The aircraft fuel system, for example, contributes to all flying phases but has only two modes; the first is the refuelling mode which takes place during the pre-flying phase and the second is the consumption mode, which takes place during the other phases.

The term mode has also been used by Feiler and Rugina (2007) and Feiler *et al.* (2006) to refer to the situation in which a system or a sub-system has different operational structures but delivers the same functionality. Figure 3-2 illustrates an example of two operational structures of a flight control system (FCS) of a space shuttle. Initially, FCS operates in the primary mode; should the primary computer fail, the backup computer takes over and thus the backup mode is launched (Lansier *et al.*, 2010).

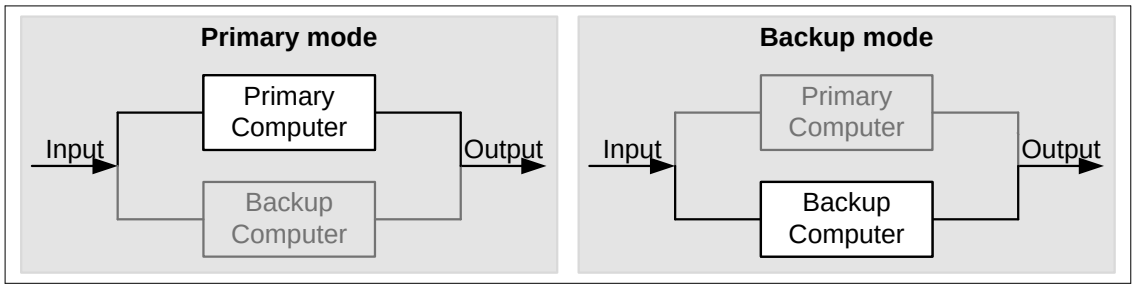


Figure 3-2: The Primary and Backup Modes of the FCS.

In Feiler and Rugina (2007), a s*tate* is defined as the condition of health of the functionality of a system or any of its components during a *mode* (see also the distinction between error-free and error states adopted in section 3.2.2).

3.4 Effective Hierarchical Level for Monitoring the Operational Parameters

The different levels in a system hierarchy can be used in different ways for better supporting effective monitoring. Normal and abnormal behaviour are typically affected by the basic components (level*0*) that represent the physical control of the process and thus one could say that level*0* is an effective level at which to detect faults early before they escalate to hazardous failures. However, a difficulty with selecting level*0* to monitor the parameters is the absence of behavioural knowledge to distinguish between normal and abnormal conditions. For example, at the higher architectural levels the decreasing of velocity and altitude seems normal during the approaching phase of an aircraft, since the flight control system (FCS) has already launched that phase. On the other hand, this condition would seem like a malfunction if the action of the FCS is not considered by the monitor and the focus is only on the basic components at level*0*. Basic components simply do not always know the context within which they operate. Level*0* is also inappropriate due to the potentially huge number of the basic components, which could make computation too expensive or even make the monitor unworkable; computationally, it will be hard dealing with complex monitoring data of a huge number of components.

At the highest (i.e. system) level (Level*n*) the behaviour of the monitored system could be understood more easily; e.g. decreasing the velocity and altitude seems normal after the

launching of the approach phase by the FCS. At that level, moreover, the computation cost would be minimised, as the monitor would be reasoning on a smaller number of parameters. However, the necessity of early fault detection means that this level cannot be selected as a starting point for observing the health of the system. By the time faults are detected at this level, they would have propagated from the basic components level through to sub-system levels and reached the system level. In this case, diagnosing the underlying causes will not be easy and consequently fault controlling decisions cannot be taken accurately.

This discussion suggests that the effective level is one of the sub-system levels that extend from level*1* to level*n-1*, and the correct selection can be made by achieving a reasonable balance among three monitoring factors:

- System behavioural understanding.
- Early fault detection.
- Computational cost.

Figure 3-3 schematically illustrates the general relationship between the above three factors and the hierarchical levels of a system. The chart shows that at the basic component level, computational cost and early detection are at their peak, whereas behavioural understanding is at its minimum. On the contrary, at the system level (level*n*) the computational cost and early detection are at their minimums, whereas behavioural understanding is at its peak.

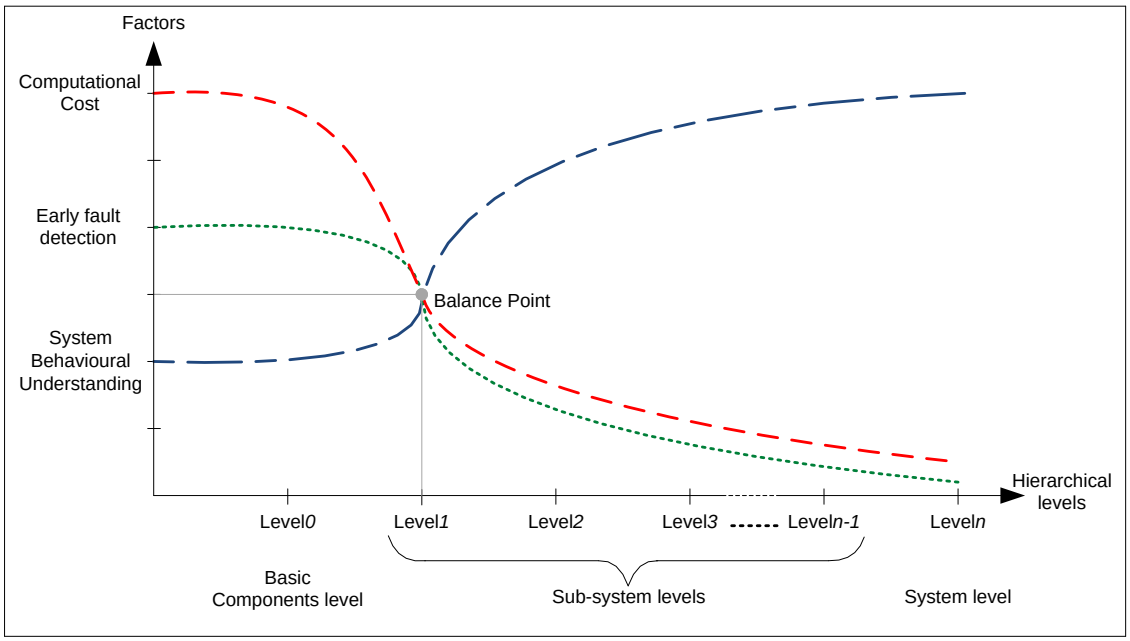


Figure 3-3: The Balance Point among the Three Monitoring Factors across the Hierarchical Levels.

There is a level where there is an optimal trade-off among the three factors that one would look to optimise. This level might differ from one system to another and depend on how levels are

organised. In general, level*1* could be a good candidate. At this level, usually groups of components are organised as sub-systems and achieve substantial functions over different behavioural contexts. Moreover, at this level, computational cost reduces sharply, behavioural understanding increases sharply and very early fault detection can be achieved. Without loss of generality, in the rest of the thesis, it is assumed that primary detection of the symptoms of failure occurs at level*1*.

3.5 The Distributed On-line Safety Monitor

Figure 3-4 illustrates the position, architecture and constituents of the distributed on-line safety monitor. As shown in that illustration, the monitor takes a position between the monitored system and the operator interface.

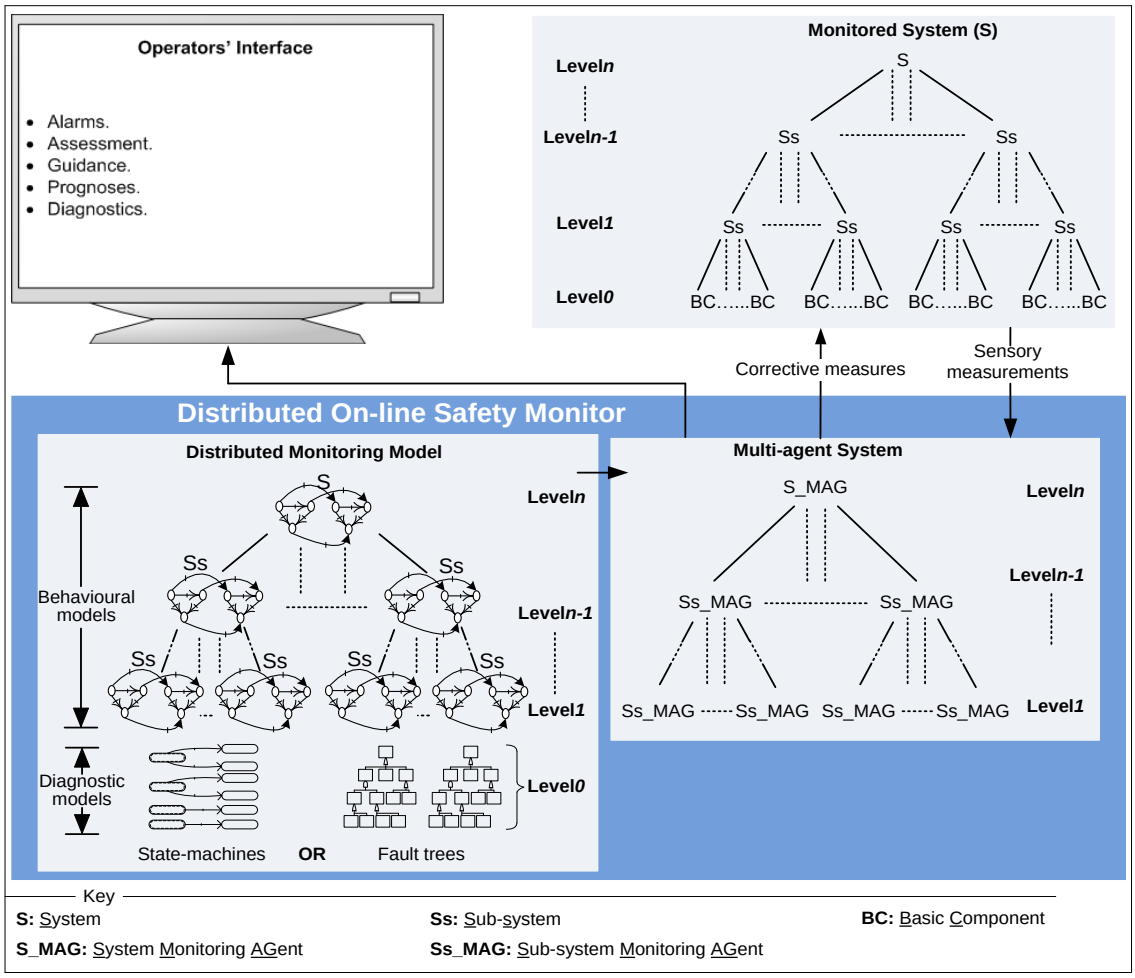


Figure 3-4: Monitor's Position, Architecture and Constituents.

3.5.1 Role of the Monitor

During normal conditions, the role of the monitor is confined to providing operators with simple feedback. The monitor plays its primary and major role during abnormal conditions, which are

triggered by the occurrence of faults. In that role three safety tasks are delivered: prompt fault detection and diagnosis, effective alarm annunciation and fault controlling. Figure 3-5 shows the sub-tasks performed in the context of these tasks.

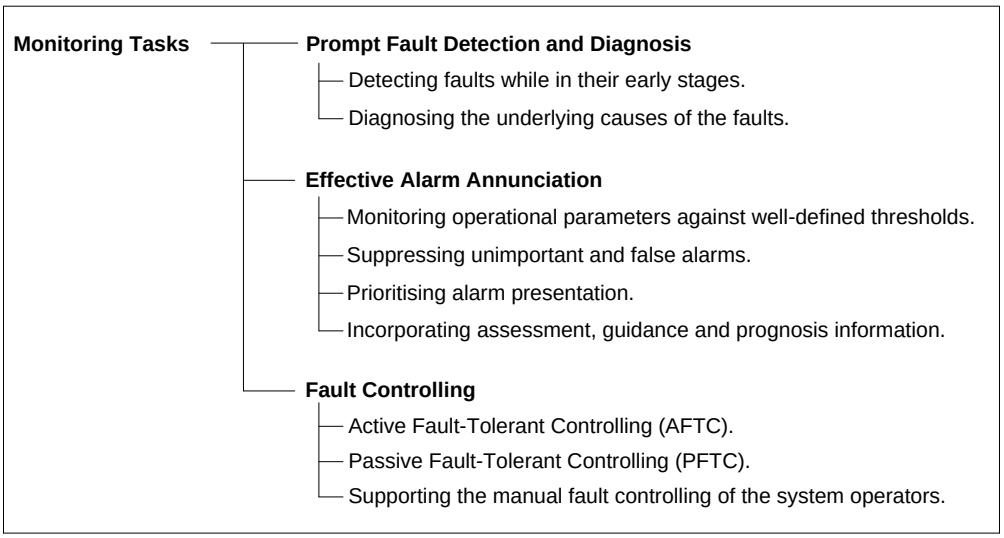


Figure 3-5: Deliverable Safety Monitoring Tasks and their Sub-tasks of the Monitor.

3.5.1.1 Prompt Fault Detection and Diagnosis

The term prompt is used here to refer to the timeliness of the delivery of the task. It implies the detection of faults while in their early stages and before they develop into real hazards, in parallel with diagnosing the underlying causes. That is supported by selecting an appropriate hierarchical level at which to monitor the operational parameters (section 3.4) and also by setting and monitoring those parameters against well-defined thresholds, which will be discussed next.

3.5.1.2 Effective Alarm Annunciation

Bearing in mind alarm annunciation issues discussed in section 2.3, the monitor has been designed to provide the operators with effective alarm annunciation that incorporates the following processes:

- Setting well-defined thresholds whose violation represents actual deviations of the monitored parameters. Well-defined thresholds should not be so sensitive as to result in triggering false alarms, but also not too indifferent and deprive the operators of prompt fault detection and alarm annunciation.
- Suppressing unimportant and false alarms whose release would overwhelm and confuse the operators. This is achieved by, firstly, effective tracking of the behaviour of the monitored system and distinguishing among the occurrence of normal, corrective and failure events;

secondly, by releasing an alarm only on the occurrence of genuine failure events and not on other events, such as consequent, precursor or causal events. Finally, developing techniques to filter out and validate the sensory measurements.

- Prioritising alarm presentation. This can be achieved by distinguishing the important alarms by using different colours, vibration or alerting sounds, and hiding the presentation of the less important alarms, e.g. optional access to the diagnostics list on the operators interface.
- Incorporating information that could help the operators to direct the system effectively in the presence of faults and control abnormal conditions. Information is presented as (a) assessment of the operational conditions following the occurrence of the fault; (b) guidance on the corrective actions that should be taken manually by the operators; (c) timely prognosis of the future effects of the occurred fault. In order to avoid overwhelming the operators, prognoses would be presented in a timely manner and in the context of behavioural transitions of the monitored system.

3.5.1.3 Fault Controlling

The monitor can achieve both active fault-tolerant controlling and passive fault-tolerant controlling[1] and also support manual fault controlling by assessment, guidance and prognoses to control abnormal conditions that may fall beyond the trained skills of the operators.

3.5.2 The Architecture and Constituents of the Monitor

As shown in Figure 3-4, the monitor consists of two elements. The first is the distributed monitoring model, which holds the required monitoring knowledge upon which the safety reasoning on the monitored parameters and operational conditions is based. The second is the multi-agent system, which is a number of monitoring agents developed and deployed to achieve distributed reasoning. Agents use knowledge provided by the monitoring model to observe the parameters and reason on the conditions of the monitored system. In this chapter the development of the distributed monitoring model is discussed, whereas discussion on the development of the multi-agent system is given in the next chapter.

3.6 Distributed Monitoring Model

To deliver the three safety tasks the monitoring model should incorporate behavioural and diagnostic knowledge. Behavioural knowledge should support tracking the state transitions of the monitored system. Diagnostic knowledge should support tracking the error propagation paths from the detected failure events towards underlying causes.

[1] That depends, however, on whether the monitored system is provided with such provision.

Practically, behaviour can be tracked by verifying and distinguishing among the occurrences of normal, abnormal and corrective events. Diagnosis requires verifying the occurrence of symptoms to track the error propagation path and then diagnosing the underlying causes. In order to verify their occurrence, the events and symptoms are augmented with formal expressions that can be evaluated in the real-time computation. These are referred to as *monitoring expressions*[1].

3.6.1 Monitoring Expressions

In its simple form a monitoring expression is a constraint consisting of three parts:

- An observation, which could be (a) a sensory measurement; (b) a calculation of a number of sensory measurements; (c) a state[2] of a parent or child component, or a logical combination of children's states.
- A relational operator, which could be equality (==), inequality (< >), greater than (>), less than (<), greater than or equal (>=), and less than or equal (<=).
- A threshold, which is a value set against an observation and when it is crossed, the evaluation of the expression results in a true truth value. The true value verifies the occurrence of an event or symptom.

In order to illustrate the formalisation of events and symptoms as monitoring expressions, Figure 3-6 introduces an example of a condensing (simple batch) sub-system. The sub-system is a component of a chemical plant; its role is to condense gas into liquid and it delivers its functionality across two modes: condensing mode (CM) and emptying mode (EM). The sub-system consists of:

- A condenser vessel which has an inlet to let hot gas in and an outlet to let condensed liquid flow out. The vessel is also surrounded by a cooling jacket which has coolant flowing in and out to reduce the temperature and support the condensing process.
- A level sensor (LS) to measure the liquid level inside the vessel.
- A flow control valve coupled with a position sensor (VP) to control the outlet line.
- A pump coupled with a speed sensor (PS) to suck the liquid from the vessel and pump it through the outlet line.
- A flow meter (FM) to measure liquid volume flows through the outlet line.

[1] The term monitoring expression appeared earlier in Dheedan and Papadopoulos (2010, 2011and 2012).

[2] As discussed in section 3.2.2, the dynamic behaviour is communicated across the hierarchical architecture. States of the parents and children is a means to communicate the local behaviour of the components across the hierarchy. This will be discussed further and illustrated by an example in the next section (section 3.6.2).

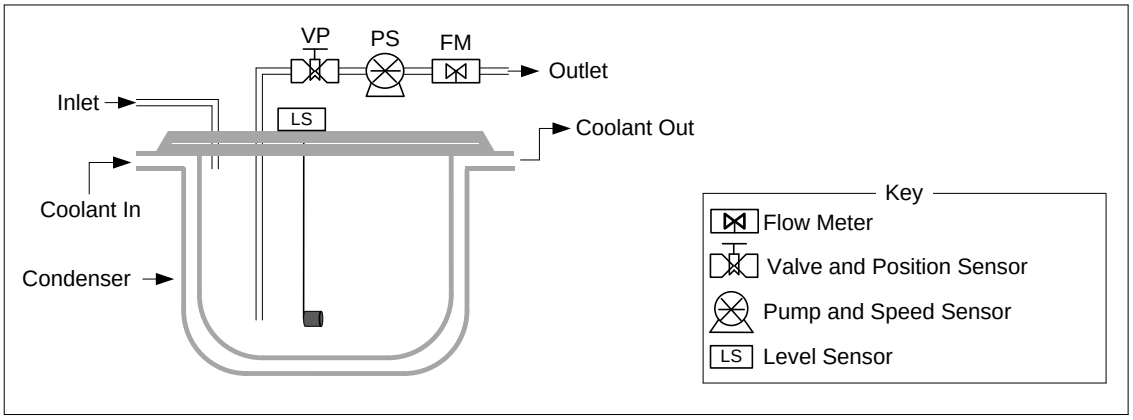


Figure 3-6: Condensing (Simple Batch) Sub-system.

During the emptying mode, the sub-system is monitored against the failure event "No flow at the outlet". An expression that can verify the occurrence of this event can be formed as follows:

$$|FM| < 0.04$$

Where |FM| : is the absolute measurement delivered by flow meter (FM).
0.04: is the threshold formed according to the potential bias of the flow meter (FM). The range of measurements between -0.04 and +0.04 is assumed to represent no flow. This range is the possible bias of measurements delivered by sensor FM.

In a different case the threshold may appear as a Boolean value, e.g. when the sub-system reports its state to its parent. Such an event can be formed as follows:

$$State\_name == True$$

The threshold may appear also as two values that restrict a measurement within two thresholds, e.g. $T_1 < measurement < T_2$, where T_1 and T_2 are thresholds.

A monitoring expression may also be formed as a logical combination of constraints. This is needed to monitor an event or symptom from different observations. Consider, for example, the following expression forms:

constraint AND constraint
constraint OR constraint

3.6.1.1 Monitoring Expressions and Time Factor

Expressions may involve time as a factor to monitor the (a) long term change of quantities in response to time for calculating the change in the trends of parameters (via use of

differentiation); (b) accumulation or summation of values over a period of time (via integral calculus).

For example, during the emptying mode, the vessel of the condensing sub-system is monitored against a failure event of structural leak. Such an event can cause either outward or inward leakage between the coolant and the condensed liquid. Thus the event can be monitored from the difference between the liquid volume that flows out of the vessel and the decrease in volume inside the vessel (liquid level). Assuming that both flow meter FM and level sensor LS have no sensory failure, the expression of this event can be formed as follows:

$$\left|\left(\int_{T-\Delta t}^{T} FM(t)dt\right) - (LS(T-\Delta t) - LS(T))\right| > \text{allowable\_descripancy} \quad \text{..... 3-1}$$

Where $\int_{T-\Delta t}^{T} FM(t)dt$: is the actual volume of liquid flowed out from the vessel over an interval of Δt. It is calculated from the integral of measurements collected from FM over an interval extended from time T-Δt in the past to the current time T.

$(\mathrm{LS}(\mathrm{T}-\Delta\mathrm{t})-\mathrm{LS}(\mathrm{T}))$: is the actual decrease in the liquid level inside the vessel over an interval of Δt. It is the difference between two measurements delivered by level sensor (LS) at time T-Δt in the past and at the current time T.

Allowable_decripancy: is a small allowable discrepancy that may normally occur between the above two trends (flowed liquid and decreased level) due to the possible bias of the sensors.

3.6.1.2 Filtering Spurious Sensory Measurements

Sensors may deliver spurious measurements due to:

- Additive white Gaussian noise, such as electromagnetic interference, ionisation radiation and thermal noise.
- Behavioural transitions of the system; such transitions are typically followed by an interval of unsteady conditions in the value of controlled parameters before these can reach a new steady state.

One way to filter out such measurements is by tying the expressions to a time factor and forming *timed expressions*[1]. In this technique, an expression is evaluated successively over a filtering interval and based on a number of measurements collected at different times. The final evaluation result is obtained by making cumulative conjunctions among the successive results. If the final result is true then that means measurements remain the same over the filtering interval and thus the occurrence can be verified.

[1] See also Dheedan and Papadopoulos (2010, 2011).

The filtering interval can be identified based on experimental examination of conditions that may result in spurious measurements and the interval at which expressions are demanded to be evaluated by the monitor. In practice, sensors deliver measurements that often show probabilistic distribution. Assuming that this distribution matches normal distribution in which there is a non-zero probability of spurious measurements and by collecting a number (n) of experimental variables (x) and calculating their mean (μ), standard deviation (σ) and standard score (z), the bell-shaped normal distribution curve can be illustrated as shown in Figure 3-7. The shaded area represents the probability in which spurious measurements fall. Accordingly, the probability of delivering spurious measurements can be expressed as P[z > Limit].

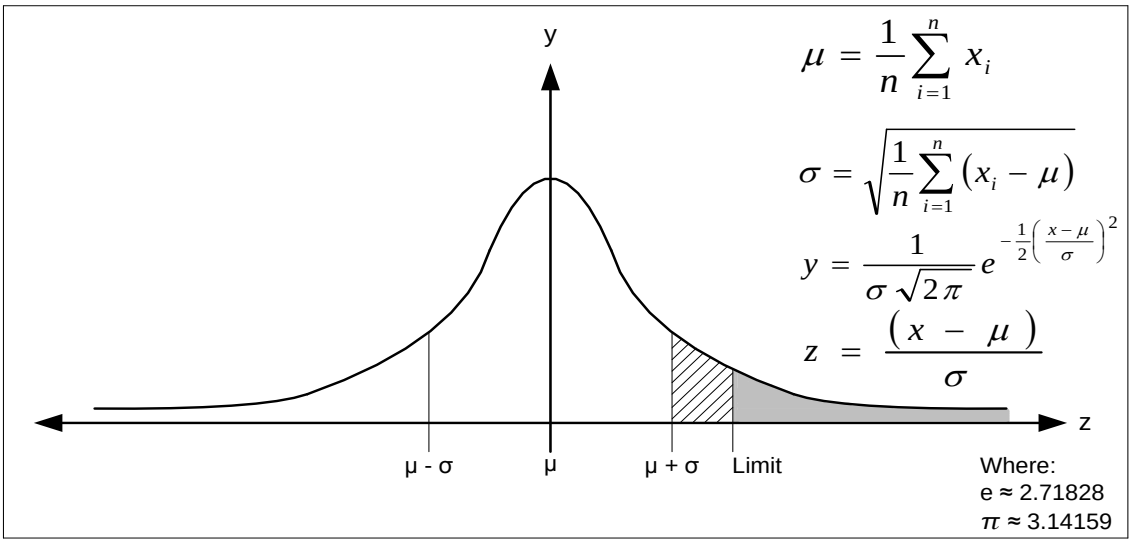


Figure 3-7: Probabilistic Distribution of Genuine and Spurious Sensory Measurements.

For example, the failure event "No flow at the outlet" is formed above as |FM| < 0.04. If the probability of sensor FM delivering one spurious measurement is 0.01, then consequently the probability of a false evaluation of the expression is 0.01. Assuming also that the expression is demanded to be evaluated every 3 seconds, the measurements can be filtered over an interval of 3 seconds. The timed expression is thus as follows:

|FM| < 0.04 for 3 seconds **3-2**

The probability of a false evaluation after making conjunction between two evaluations is 0.0001 (it is calculated as 0.01 X 0.01). Since the conjunction of two evaluations is less likely to be false, making conjunction among more evaluations would make a false evaluation very unlikely.

It must also be pointed out that, when the value of the delivered measurements lies close to the value of spurious measurements, false evaluation is also possible (this is represented by the

striped area in Figure 3-7). Considering the possible sensory bias, relaxing the threshold, prolonging the filtering interval and increasing the number of evaluations could effectively avoid false evaluation and address this issue. However, serious consideration must be given to not formalising too indifferent an expression, as discussed in section 2.3.1.2.

3.6.1.3 Sensory Measurements Validating and Sensory Faults Detecting, Diagnosing and Controlling

In practice, sensors may fail permanently and deliver spurious measurements that persist over or even go beyond the filtering interval. In addition to misleading the monitor; such measurements could also affect the controller of the monitored system and result in hazardous failures. Sensory measurements should, therefore, be validated and faulty sensors should be detected, diagnosed and controlled.

To achieve that, a technique of formalising special monitoring expressions is developed. The technique is based mainly on the sub-grouping approach of Yu and Su (2006) and the Sensory Failure Diagnosis Tree (SFDT) approach of Kim *et al.* (1990, 1988), see also section 2.3.2.2. Drawing from the sub-grouping approach, sensors that can detect each other's faults are identified and based on the idea of SFDT the proper expression is formed.

For example, to validate measurements and detect and diagnose a fault of the flow meter FM of the condensing sub-system, the level sensor LS is involved in the validating group. Accordingly, the monitoring expression can be formed as follows:

$$\left(\mathrm{LS(T-\Delta t)} > \mathrm{LS(T)}\right) \boldsymbol{AND} \left(|\mathrm{FM}| == 0 \text{ for 3 sec } \mathbf{OR} \left|\int_{T-\Delta t}^{T} FM(t)dt\right| > \text{max flow}\right) \quad \textbf{3-3}$$

Where $\left(\mathrm{LS(T-\Delta t)} > LS(\mathrm{T})\right)$: is verified true when the liquid level inside the vessel decreases over an interval of Δt.

$FM == 0$ for 3 sec: is verified true when FM fails and keeps delivering equal to zero measurements over an interval of 3 seconds, such a failure may occur due to a short circuit.

$\left|\int_{T-\Delta t}^{T} FM(t)dt\right| >$ max flow: is verified true when FM fails and delivers abnormal measurements that fall out of the expected range, as "max flow" is the acceptable volume measured by FM over the interval Δt.

To control sensory failures, the technique suggests isolating the faulty sensor by ignoring its measurements and measuring the same trend from an alternative sensor or from a number of sensors whose measurements can be calculated to correspond as alternative to the isolated measurement. In the case of isolating the flow meter (FM), alternative sensors can be the liquid level sensor (LS) or the pump speed sensor (PS).

3.6.1.4 Grammar of the Monitoring Expression

Using E-BNF notation, Figure 3-8 presents a grammar[1] to formalise different events and symptoms as monitoring expressions. An *expression* can be a *constraint* or a logical combination of two or more constraints. A *constraint*, in turn, might be formed as an *observation*, *relational operator* and *threshold*. When the *threshold* is a range that bounds an *observation*, the *constraint* is formed differently as a *threshold*, *relational operator*, *observation*, and another *relational operator* and *threshold*.

```
expression                  ::= constraint, { logical_operator, constraint };
constraint                  ::= observation, relational_operator, threshold
                              | threshold, relational_operator, observation,
                                relational_operator, threshold;
observation                 ::=  measurement | behavioural_state;
measurement                 ::= term, { ( + | - ), term };
term                        ::= factor, { ( * | / ), factor };
factor                      ::= basic, [** basic]
                              | abs basic
                              | not basic;
basic                       ::= constant
                              | sensor_identifier | sensor_identifier (Δt)
                              | (expession)
                              | T (expression, Δt)
                              | I (expression, Δt)   | I (measurement, Δt)
                              | V (expression, Δt) | V (measurement, Δt)
                              | D (expression, Δt) | D (measurement, Δt);
behavioural_state           ::= state, { logical_operator ,  state};
state                       ::= EFS |  TDFS | PFS | FS;
logical_operator            ::= "AND" | "OR" ;
constant                    ::= boolean | integer | real;
relational_operator         ::= <>   | ==    | <   | >   | <=   | >=;
threshold                   ::= observation;

where
Δt          ::= a time interval;
EFS         ::= Error-free State;
TDFS        ::= Temporary Degraded or Failure State;
PDS         ::= Permanent Degraded State;
FS          ::= Failure State;
```

Figure 3-8: Grammar of the Monitoring Expressions.

An *observation* is either a *measurement* or *behavioural state*. A *measurement* can be a *term* or a combination of subtraction or addition of a number of *terms*. A *term* is either a *factor* or a combination of multiplication or division of a number of *factors*. A *factor* can be (a) a single *basic*; (b) a *basic* raised to the power of another *basic*; (c) an absolute value of a *basic*; (d) a negation of a *basic*. A *basic* can be one of the following:

- A *constant*, which is a *Boolean*, *integer* or *real* number.

[1] This presentation is an extension of the grammar presented earlier in Papadopoulos (2000).

- A *sensor_identifier* of a sensor from which up-to-date measurements are collected or a time *sensor_identifier(Δt)* of a sensor from which historical measurements[1] are collected over an interval of *Δt*.
- An *expression*, which is as defined above.
- A timed expression *T(expression,Δt)*, as shown by expression 3-2;
- An integral of an expression *I(expression,Δt)* or measurements *I(measurement, Δt)*.
- A difference (variation) of an expression *V(expression,Δt)* or measurements *V(measurement,Δt)* over a period of time.
- A differential of an expression *D(expression,Δt)* or measurements *D(measurement,Δt)*.

A *behavioural state* can be a single *state* or a logical combination of two or more *states*. A *state* can be an error-free state *EFS*, failure state *FS*, temporary degraded or failure state *TDFS* or permanent degraded state *PDS*. Finally, a *threshold* is an *observation*, which is as defined above.

3.6.1.5 Consistent Updating and Evaluation of the Monitoring Expressions

Expressions that do not involve a time interval need an updatable buffer that can hold an up-to-date measurement, i.e. single-measurement buffer. However, the evaluation of timed expressions and expressions involving the time factor needs updatable buffers that can hold more than one historical measurement over an interval, i.e. a multi-measurement buffer. To maintain consistent observation, the multi-measurement buffer needs a systematic updating process.

Figure 3-9 shows an example of the updating process of a three-measurement buffer. At a current time (T), the buffer holds the current measurement updated at the current time M(T) and historical measurements M(T-Δt/2) and M(T-Δt). When an interval of Δt/2 has elapsed, the time of the three measurements will differ as follows: M(T) becomes M(T-Δt/2), M(T-Δt/2) becomes M(T-Δt), and M(T-Δt) becomes M(T-Δt-Δt/2). As the latter measurement falls out of interval Δt, it is replaced by the up-to-date measurement M(T).

When a further interval of Δt/2 has elapsed, the time of the three measurements will differ again as follows: M(T-Δt/2) becomes M(T-Δt), M(T-Δt) becomes M(T-Δt-Δt/2) and M(T) becomes M(T-Δt). As the measurement M(T-Δt-Δt/2) falls out of Δt, it is replaced with the up-to-date measurement M(T). This updating is applied continuously over time and after every elapsing of Δt/2. It can be seen how the updating maintains a systematic interval of Δt/2 among the

[1] A historical measurement is one that is collected at a certain time in the past and held to be involved in an evaluation of an expression.

measurements and replaces measurements that fall out of Δt. This structure could hold sensory measurements that suffice for the calculation and evaluation of expressions like expression 3-1, expression 3-2 and expression 3-3.

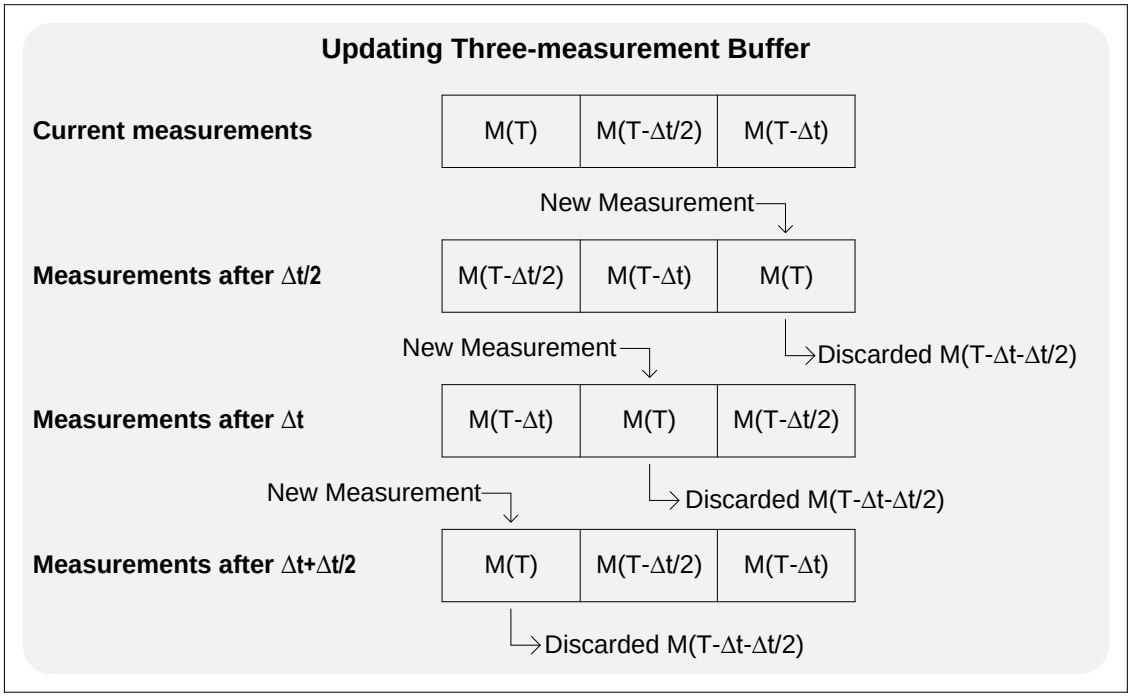


Figure 3-9: A General Illustration of Updating a Multi-measurement Buffer.

Computationally, every multi-measurement buffer is implemented as a number of structures held by an array and each structure holds two attributes, the sensory measurements and the time at which that measurement is collected. The following pseudo code shows the systematic updating of a multi-measurement buffer:

```
For (i = FM; i <= LM, i++)                  //FM and LM are respectively the first and last
  {                                         //measurements in the buffer.
    if TM[i].Time – CurrentTime > Δt         //TM is a buffer array (see also Figure 3-14). If
     {                                      //the collecting time of the current
                                            //measurement falls out of Δt.
       TM[i].Measu= update(SensorID);       //Then update the buffer by collecting an up-to-
                                            //date measurement from the relevant sensor.
       TM[i].Time = CurrentTime;            //Update the collecting time of the
     }                                      //current measurement.
  }
```

Along with this systematic updating, an issue that may slow down the evaluation and ultimately undermine the effectiveness of the monitor has also been considered. The issue may arise from expressions that logically combine two or more expressions, each of which needs a different filtration or calculation interval. To address this issue without violating the consistency of the

evaluation process, a technique of three-truth values logic has been introduced[1]. To illustrate the technique, the following two forms of expressions are considered as an example:

(Expression) OR (Expression, ⊿t)..... **3-4**
(Expression) AND (Expression, ⊿t) **3-5**

Evaluating expressions 3-4 or 3-5 may require waiting time equal to Δt. As the part *Expression* of 3-4 and 3-5 can be evaluated instantly, the part (*Expression, ⊿t*) will stay unknown for an interval of Δt. The three truth-value technique exploits the fact that disjunction of true with unknown is true and the conjunction of false with unknown is false, to mask the unknown value and evaluate those expressions instantly. Hence, when the part *Expression* of 3-4 is evaluated with true and the part *Expression* of 3-5 is evaluated with false, both expressions can be evaluated instantly with the values true and false, respectively. Thus, the technique masks unknown truth values and computes the ultimate truth value of compound expressions from the known and unknown values of constituent parts.

However, when the part *Expression* of 3-4 is evaluated with false and the part *Expression* of 3-5 is evaluated with true, both expressions should stay unknown until their second parts have been evaluated. To address this case and achieve prompt evaluation, Bayesian probability could be exploited for that purpose. Despite the significant advantage of such a strategy, it is considered to be outside the intended scope of this thesis and thus it is left as an open issue for future research.

3.6.2 Modelling Behavioural Knowledge

State-machines are widely used to model the behaviour of systems (Pressman, 2001 and 1998). In a state-machine, normal, failure and corrective events trigger transitions and states appear as either a starting place or a destination of each transition. Figure 3-10 introduces the modelling notation of the state-machine. Every state is represented as a rounded rectangle and the initial error-free state is always initiated by a solid-headed and pointed-ended arrow. Transitions among states of different modes are represented by solid-headed arrows and transitions between states of the same mode are represented by hollow-headed arrows.

Obtaining behavioural knowledge requires two analysis steps. The first is analysing the normal behaviour (EFSs) of the system and its sub-systems. This can be carried out through design models, such as Unified Modelling Language (UML), which offer detailed specifications of the deliverable functions over each possible mode (Object Direct, 2010; Weilkiens, 2006).

[1] See also Dheedan and Papadopoulos (2010, 2011).

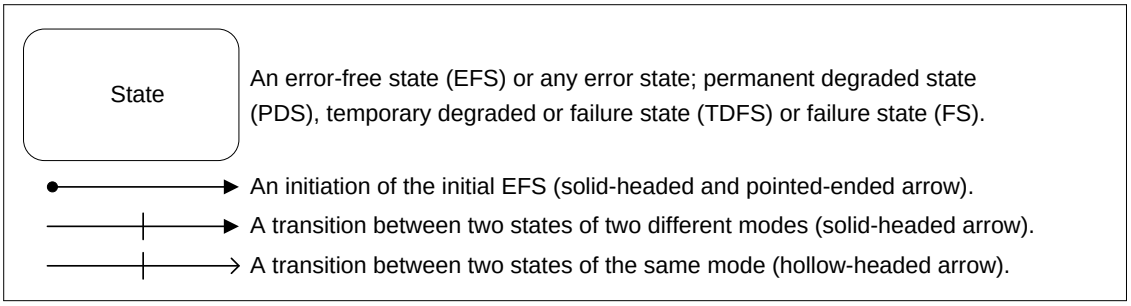


Figure 3-10: Behavioural Modelling Notation.

In the second step, the functions of each mode are analysed with the aim of identifying their potential failures, effects of failures on the operating conditions and possible corrective measures. Safety analysis techniques, such as functional failure analysis (FFA) or HAZard and OPerability Study (HAZOP), can be used to extract such knowledge (Papadopoulos, 2000 and 2003; Pumfrey, 1999). In this way, knowledge about the abnormal behaviour, assessment, guidance and corrective measures can be obtained. Incorporating non-conflicting corrective measures, which has been highlighted as an important issue in Chapter Two (section 2.4.1), should also be considered in this step.

In alignment with monitoring parameters at level*1* (section 3.4) and to record the behaviour and potential interaction of the components at that level and higher up across the hierarchy of the monitored system, the behaviour of each component of levels extending from level*1* to level*n* is modelled in a state-machine.

This modelling approach results eventually in a hierarchy of communicating state-machines. At level*1*, state transitions are triggered by events initiated by the basic components. Transitions at this level may result in state transitions in the state-machine of the parent at the immediate upper level. This scenario is repeated upwardly between the state-machine of each child and the state-machine of its parent until the top level. Similarly, it may be repeated downwardly between the state-machine of each parent and the state-machines of its children until level*1*.

Figure 3-11 shows a simple example of state-machine communication. The hierarchy records the behaviour of a hypothetical system S and its two sub-systems: A and B. System S delivers its functionality over two modes, MX and MY. Each mode is modelled in three different states: EFS, PDS and FS. Sub-systems A and B deliver their functionality over two modes: MX and MY. Each of those modes is modelled in two states: EFS and FS. Sub-system A delivers functions F1 and F2 in the respective modes MX and MY. Sub-system B delivers functions F3 and F4 in the respective modes MX and MY.

The hierarchy shows how state transitions at level*1* are triggered by normal and failure events and how the state-machines can communicate across hierarchical levels. For instance, normal

event NE1, which is initiated by basic components (level*0*), triggers a state transition in the state-machine of sub-system A; the transition from EFS of mode MX (A_MX_EFS) to EFS of mode MY (A_MY_EFS). This transition triggers a state transition in the state-machine of system S from EFS of mode MX (S_MX_EFS) to EFS of mode MY (S_MY_EFS); see how the MY_EFS of A appears as a trigger event in the state-machine of system S. Transition to the S_MY_EFS of system S triggers a lower level transition from B_MX_EFS to the B_MY_EFS of sub-system B; see how the MY_EFS of S appears as a trigger event in the state-machine of sub-system B. One can also see how failure events can trigger state transitions in the state-machine of the sub-systems and how the resulting error states are communicated to the state-machine of system S (the parent).

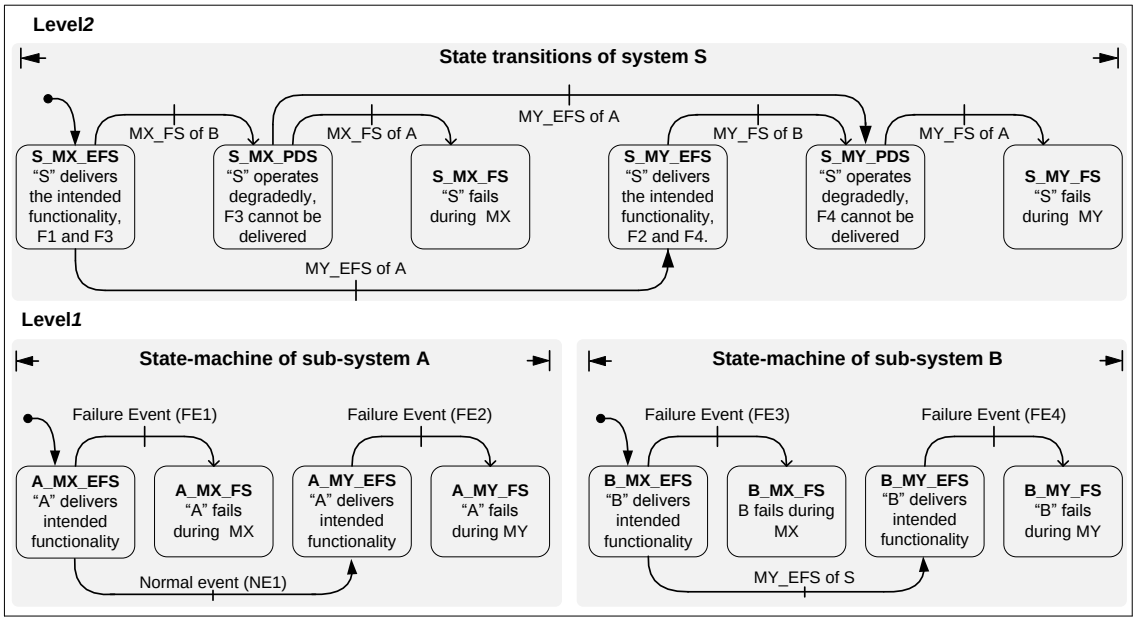


Figure 3-11: An Example of the Hierarchical Behaviour Model.

By executing the normal, failure and corrective events on a hierarchy of state-machines, the behaviour of the monitored system and its sub-systems can be tracked. This technique could also reduce the workload of the monitoring agents and rationalise the entire monitoring process. Computationally, when a sub-system is in a state, the monitoring focus is only on events that represent plausible exits (*active events*) from the current state, e.g. when sub-system A is in state A_MX_EFS, active events are FE1 and NE1, when it transits to state A_MY_EFS, the active event will be FE2.

Active events of every state are formalised as monitoring expressions. In real time, monitoring agents evaluate those events cyclically based on up-to-date sensory measurements, thereby achieving a *monitoring cycle*. After achieving every cycle, a new cycle is launched in which up-to-date measurements are collected and every expression is evaluated again.

Incorporating alarms, Assessment/Guidance, and Corrective Measures in the Behavioural Model

Along with the behavioural modelling, the computational distinction among the normal, failure and corrective events should be established. Moreover, the proper places to incorporate alarm, assessment and guidance clauses and corrective measures should also be identified.

To distinguish between events, the principle is applied that an alarm should be released on the occurrence of failure events only. Thus, corresponding alarm clauses should be associated with the failure events of level*1*, the level at which events are monitored (section 3.4). Computationally, if an occurred event is associated with a "none" then it is either a normal or a corrective event; on the contrary, the otherwise clause means that it is a failure event and the associated clause should be quoted and released as an alarm. While assessment is a description of the given conditions and guidance is about the best actions to be applied in those conditions by the operators, their clauses should thus be enclosed by the states; this is according to the definition of the state stated in section 3.2.2.

To find the appropriate place for incorporating corrective measures, further consideration of the nature of those measures is needed. Typically, there are two different types of corrective measures.

- The first type should be taken after diagnosing the underlying causes. This is appropriate when the verified failure event could be caused by multiple faults of the basic components. The correct place to incorporate such measures is then level*0*. The failure event "No flow at the outlet" of the condensing sub-system (section 3.6.), for example, could be caused by inadvertent closure of valve (VP), a fault of pump (PS), blockage of the outline or abnormal emptiness of the condensing vessel. Typically, measures to correct any of those causes vary from one cause to another. In this case, measures should be incorporated in the diagnostic model (e.g. fault tree), precisely in association with the potential causes.
- The second type of corrective measures should be taken at level*1*, when level*1*'s sub-systems supported by higher level components (sub-systems or system) apply measures to respond to deviations that have a clear cause. At level*1*, corrective measures are mostly applied with directions coming from higher levels. For example, switching to the backup computer sub-system at level*1* is instructed directly by the flight control system (FCS) at level*2*, whenever the primary computer sub-system at level*1* fails (Figure 3-2). The instructions are implemented at level*1* by switching the primary computer off and backup computer on. Corrective measures should also be taken at level*1*, when level*1*'s sub-systems supported by level*0*'s basic components apply measures to respond to deviations that have a clear cause. Expression 3-3, for example, relates a failure event of the

condensing sub-system directly to a fault of the flow meter (FM). In this case, measures are taken to isolate the FM and depend alternatively on the measurements obtained from the liquid level sensor (LS).

Accordingly, state-machines of level*1* would record the (a) state transitions; (b) alarm and assessment/guidance clauses; (c) corrective measures. The higher level state-machines should record state transitions and assessment/guidance clauses.

To present the graphical state-machines in an executable format, state-transition tables represent a classic choice. A state-transition table is usually defined as an alternative and formal form to present graphical state-machines and it typically offers the required capacity and flexibility to incorporate knowledge about the operational conditions (Breen, 2005).

State-transition Table of State-machines of Level*1*

The standard form of state-transition table consists of three columns: current state, trigger event and new state. According to the monitoring requirements at level*1*, this form is extended to incorporate seven columns: current state, conditions, event, alarm, controlling, diagnosis and new state. Figure 3-12 presents the formal grammar of the table using the E-BNF notation.

The table appears as a number of columns and rows. Each column consists of (a) *column headers*, which are *CURRENT*, *CONDITIONS*, *EVENT*, *ALARM*, *CONTROLLING*, *DIAGNOSIS* and *NEW STATE*; (b) *column cells*, of which there are two or more of each of the following: *current state cells*, *conditions cells*, *event cells*, *alarm cells*, *controlling cells*, *diagnosis cell*, and *new state cells*.

A *current state cell* encloses the *current state name* formed as three parts separated by two underscores; the first part is the *component initials*, the second part is the *mode initials* and the third part is the *state initial*, which might be error-free state *EFS*, temporary degraded or failure state *TDFS*, permanent degraded state *PDS* or failure state *FS*. The *current state name* may be associated with a number as a state may appear more than once throughout the same mode. The *conditions cell* encloses clauses of *assessment* and *guidance*, each of which is preceded by the respective terms *Assessment:* and *Guidance:*. The *event cell* contains either an expression, which is as defined in Figure 3-8, or *none* when there is no exit from the current state.

An *alarm cell* may enclose an *alarm clause*, which can be quoted and announced as an alarm on the occurrence of a failure event, or *none* when the event is either a corrective or normal event. A *controlling cell* may contain (a) *none*, when the occurred event is either a corrective or normal event; (b) *impossible*, when the occurred event is a failure event that is not controllable; (c) *after_diagnosis*, when the occurred failure event needs to be controlled at the basic

components level (level*0*) and after diagnosing the underlying cause; (d) one or more *corrective measures*, when the occurred event is a failure event and can be controlled at level*1*. *Corrective measures* can be instructed to the actuators through the *actuator identifier* or isolate a faulty sensor by instantiating its *sensor identifier* with alternative sensory measurements collected from other sensors through their *sensory identifiers*. For the latter case, a measure may appear as a mathematical expression, which has the same grammar as shown in Figure 3-8.

```
state_transition_table   ::= column{column}, row{row};
column                   ::= column_header, column_cell;
column_header            ::= “CURRENT STATE” , “CONDITIONS” , “EVENT” , “ALARM” , “CONTROLLING”
                             , “DIAGNOSIS” , “NEW STATE”;
column_cell              ::= current_state_cell{current_state_cell}- , conditions_cell{conditions_cell}-,
                             event_cell{event_cell}- , alarm_cell{alarm_cell}- , controlling_cell{controlling_cell}-
                             , diagnosis_cell{diagnosis_cell}- , new_state_cell{new_state_cell}-;
current_state_cell       ::= state_name;
state_name               ::= component_initials, “_”, mode_initials, “_”, state_initials, [number];
number                   ::= digit{digit};
digit                    ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
conditions_cell          ::= assessment, guidance;
assessment               ::= “Assessment:”, (*declaration of the current operational conditions*);
guidance                 ::= “Guidance:”, (*best directive actions after the occurrence of faults*)
                              | “none”;
event_cell               ::= expression | “none”;
alarm_cell               ::= (*alarm clause*) | “none”;
controlling_cell         ::= “none” | “impossible” | “after_diagnosing”
                             | {corrective_measure}-;
corrective_measure       ::= “-“, sensor_identifier, “=”, alternative;
                             | “-“, actuator_identifier, “=”, measure;
alternative              ::= term, { ( + | - ), term };
term                     ::= factor, { ( * | / ), factor };
factor                   ::= basic, [** basic]
                             | abs basic
                             | not basic;
basic                    ::= constant
                             | sensor_identifier
                             | sensor_identifier (Δt);
constant                 ::= boolean | integer | real;
measure                  ::= constant;
diagnosis_cell           ::= “needed” | “not_needed” | (*name of the faulty component*);
new_state_cell           ::= state_name | none;
row                      ::= “CURRENT STATE”, “CONDITIONS”, “EVENT”, “ALARM”, “CONTROLLING”,
                             “DIAGNOSIS”, “NEW STATE”
                             | current_state_cell, conditions_cell, state_row{state_row};
state_row                ::= event_cell, alarm_cell, controlling_cell, diagnostic_cell, new_state_cell;
```

Figure 3-12: Grammar of State-transition Table of State-machine of Level*1*.

A *diagnosis cell* may contain (a) *needed*, when the failure and its underlying cause are in a one-to-many relationship and thus the diagnostic process should be launched; (b) *not needed*, when the occurred event is either a normal or corrective event; (c) *the name of the faulty component*,

when the occurred failure event and its underlying cause are in a one-to-one relationship (the cause is known). Finally, a *new state cell* contains either a *state name* which is formed identically to the current state name or *none* when there is no further transition.

Table 3-1 shows an example of a state-transition table. The table presents the formal form of the state-machine of sub-system A (level*1*) that is shown by Figure 3-11. The table shows how the alarm clauses, corrective measures and diagnostic statuses can be associated with failure events and the *conditions* column incorporates clauses of the assessment and guidance associated to every state.

Table 3-1: State-transition Table of the State-machine of Sub-system A (Shown in Figure 3-11).

CURRENT STATE	CONDITIONS	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
A_MX_EFS	**Assessment:** "A" delivers indented function in mode X**. Guidance:** none	Expression of FE1	Function F1 fails.	impossible	needed	A_MX_FS
		Expression of NE1	none	none	not_needed	A_MY_EFS
A_MY_EFS	**Assessment:** "A" delivers indented function in mode Y**. Guidance:** none	Expression of FE2	Function F2 fails.	impossible	needed	A_MX_FS
A_MX_FS	**Assessment:** "A" fails during MX. **Guidance:** do not order function F1.	none	none	none	not_needed	none
A_MY_FS	**Assessment:** "A" fails during MY. **Guidance:** do not order function F2.	none	none	none	not_needed	none

This form of state-transition table is presented as a text file to the monitoring process. The file is parsed, interpreted and loaded as a number of interrelated structures for real-time reasoning. While parsing, interpreting and loading the file, a direct access address is given to the structures of every event, state, expression and buffer. This eliminates the seek time of the real-time reasoning, which contributes to achieving fast tracking of the behaviour of the monitored sub-system and supports the efficiency of the monitor.

According to the grammar defined in Figure 3-12, Figure 3-13 shows the pseudo code that declares the needed structures to hold the table. The code is augmented with comments to explain the declaration of the structures, their attributes and arrays to hold those structures. Figure 3-14, moreover, shows an illustrative view of those structures and how they are linked to each other.

For every state that appears in the current state column of the state-transition table there is a structure named *State*. The structure holds five attributes: *StateName*, *Assessment*, *Guidance*, *FirstEvent* and *LastEvent*. To hold the number of states, a one-dimension array of states is also declared: *States [n]*.

```
structure Buffer                //A structure of an updatable buffer, e.g. Figure 3-8. It holds the following attributes:
  {
   Double Interval;             //An interval (Δt) over which sensory measurements are collected.
   Int FirstMeasurement;        //The position of the first measurement in the buffer.
   Int LastMeasurement;         //The position of the last position in the buffer.
  };

structure ExpToken              //A structure of a constituent (or a token) of an expression, and holds the following
  {                             // attributes:
   String TokenName;            //The name of token, it might be sensor ID, an interval, logical expression or any other
                                //possible constituent of an expression according to the grammar of Figure 3-7.
   Int BufferPosition;          //If the TokenName holds an interval that equals zero or more, then
  };                            //this attribute holds the position of a buffer structure.

structure Expression            //A structure of an expression; for every expression it holds the following attributes:
  {
   Int FirstToken;              //The position of the first ExpToken in the Tokens array (Tokens[jj]).
   Int LastToken;               //The position of the last ExpToken in the Tokens array (Tokens[jj]).
  };

structure CMeasure              //A structure of the corrective measures of an event; it holds the following attributes:
  {
   String ControllingStatus;    //This might be instantiated with none, impossible, after_diagnosis or
                                //corrective_measures, as defined in the grammar of Figure 3-11.
   Int FirstCMToken;            //If the ControllingStatus holds corrective_measures, then this attribute is instantiated
                                //with the position of the first token of those measures in the array CMTokens[l].
   Int LastCMToken;             //If the ControllingStatus holds corrective_measures, then this attribute is instantiated
  };                            //with the position of the last token of those measures in the array CMTokens[l].

structure Event                 //A structure of an event; for every event it holds the following attributes:
  {
   String Alarm;                //An alarm clause or none.
   Int ControllingPosition;     //The position of the measure structure in the array that holds a structure for every
                                //event (CMeasures[l]).
   String Diagnosis;            //diagnostic status.
   Int StatePosition;           //The position of the structure of triggered state in the states array (States [n]).
   Int ExpressionPosition;      //The position of the expression of the event in the expressions array (Expressions[y]).
   Int DiagnosticModelPosition;         //The position of the diagnostic model in the array of diagnostic models,
  };                                    //this for failure event whose causes need to be diagnosed. The relevant
                                        //diagnostic model is also defined as a number of interrelated structures as
                                        //shown in Figure 3-20 or Figure 3-25.

structure State                 //A structure of a state, for every state it holds the followings attributes:
  {
   String StateName;            //The name of the state.
   String Assessment;           //An assessment clause.
   String Guidance;             //A guidance clause.
   Int FirstEvent;      //The position of the structure of the first exit event held by the array Events [m] .
   Int LastEvent;       //The position of the structure of the last exit event held by the array Events [m].
  };

State [ ] States = new States [n];          //An array to hold n state structures, where n is the number of states that
                                            //appear in the current state column of the state-transition table.
Event [ ] Events = new Event [m];           //An array to hold m event structures, where m is the number of events in
                                            //the state-transition table.
CMeasure [ ] CMeasures =  new CMeasure [l]; //An array to hold l structures of corrective measures, where l is
                                            //the number of the controlling cells of the table.
Expression [ ] Expressions = new Expression [y]; //An array to hold y expression structures, where y is the number
                                            //of expressions which is either equal or less than m, as there
                                            //might be more than one event that have the same expression.
ExpToken [ ] Tokens = new ExpToken [j];     //An array to hold j ExpToken structures, where j is the number
                                            //of the tokens of the expressions.
Buffer [ ] Buffers = new Buffer [k];        //An array to hold k buffer structures, where k is the number of buffers
                                            //needed to hold measurements of the events of the table.
```

Figure 3-13: Data Structures to hold State-transition Table of Level*1*.

Every event that represents exit from a state is declared as a structure named *Event*. For every event, the structure holds six attributes: *Alarm*, *ControllingPosition*, *Diagnosis*, *StatePosition*, *ExpressionPosition* and *DiagnosticModelPosition*. As there could be a number of exit events from a state, events are held by a one-dimension array, declared as *Events [m]*. Distinction among the events of every state is achieved by holding the position of the first and last events in the attributes *FirstEvent* and *LastEvent* of the *State* structure.

For every event there is a structure to hold its associated corrective measures (*structure CMeasure*). The structure has three attributes: *ControllingStatus*, *FirstCMToken* and *LastCMToken*. If the *ControllingStatus* is instantiated with *corrective_measures*, then the *FirstCMToken* and *LastCMToken* hold the first and last positions of the tokens of those measurements in an array of measures' tokens. However, if the *ControllingStatus* is instantiated with *none*, *impossible* or *after diagnosis* (as shown in Figure 3-12) then the *FirstCMToken* and *LastCMToken* hold *nil*. The *CMeasures* array is declared to hold corrective measures of a number of events. More than one failure event may share the same structure of the corrective measures (*CMeasure*), such that the position of the common measures instantiates the attribute *ControllingPosition* of those events.

Every event has an expression which in turn consists of different constituents (*Tokens*) as defined in Figure 3-8. Accordingly, every expression is held in a structure (*Expression*) which has two attributes: *FirstToken* and *LastToken*. As there are a number of expressions, an array (*Expressions [y]*) is declared to hold a number of structures. A number of events may have the same expression; accordingly the position of such expression instantiates the attribute *ExpressionPosition* of the structures of those events.

Every token is held in a structure *ExpToken* that has two attributes: *TokenName* and *BufferPosition*. If the token held by *TokenName* is a time interval or zero-time interval, then a buffer is created. For every buffer there is a structure whose position is held by the attribute *BufferPosition*. The structure of a buffer (*structure Buffer*) holds three attributes: *Interval*, *FirstMeasurement* and *LastMeasurement*. If the interval is zero then the *FirstMeasurement* and *LastMeasurement* hold the same position of the measurement (i.e. a single-measurement buffer). This buffer is held by an array of updatable buffers.

On the contrary, if the interval is more than zero then the *FirstMeasurement* and *LastMeasurement* hold the first and last position of the multi-measurement buffer held by the updatable buffers array. More than one expression may share the same buffer and in such a case the position of the common buffer instantiates the *BufferPosition* of the *ExpToken* of those expressions.

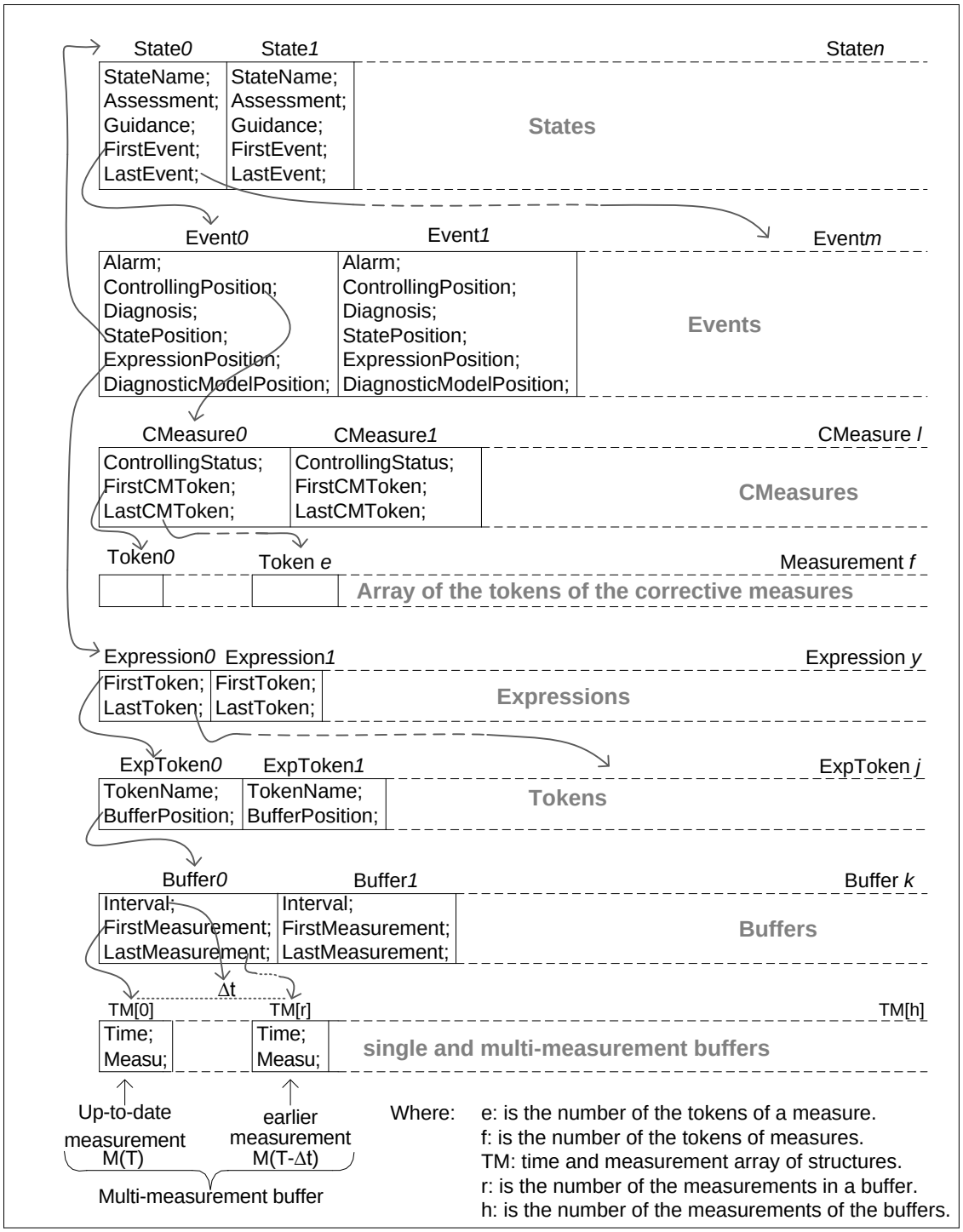


Figure 3-14: An illustrative View of Structures to hold the State-transition Table of Level*1*.

State-transition Table of State-machines of Levels Extending from Level*2* to Level*n*

To present the state-machines of levels extending from level*2* to level*n*, the standard form of state-transition table is extended to include an extra column (conditions column). Figure 3-15 presents the formal grammar of the table using the E-BNF notation.

According to the grammar, the state-transition table appears as a number of columns and rows. Each column consists of column headers and column cells. The column headers are *CURRENT*

STATE, *CONDITIONS*, *EVENT* and *NEW STATE*. A column cell, in turn, can be two or more of each of the following: *current state cells*, *conditions cells*, *event cells* and *new state cells*. The contents of these cells and the rest of the grammar are the same as the grammar explained for level*1*'s state-transition table in Figure 3-12. Table 3-2 shows an example of a state-transition table. The table presents the formal form of the state-machine of system S (level*2*), which is shown by Figure 3-11.

state_transition_table	::= column{column}, row{row};
column	::= column_header, column_cell;
column_header	::= "CURRENT STATE" , "CONDITIONS" , "EVENT" , "NEW STATE";
column_cell	::= current_state_cell{current_state_cell}- , conditions_cell{conditions_cell}- , event_cell{event_cell}- , new_state_cell{new_state_cell}-;
current_state_cell	::= current_state_name;
current_state_name	::= component_initials, "_", mode_initials, "_", state_initials, [number];
number	::= digit{digit};
digit	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
conditions_cell	::= assessment, guidance;
assessment	::= "Assessment:", (*declaration of the current operational conditions*);
guidance	::= "Guidance:", (*recommendations on how to direct the system after the occurrence of faults*) | "none";
event_cell	::= expression | "none";
new_state_cell	::= new_state_name | "none";
new_state_name	::= component_initials, "_", mode_initials, "_", state_initials, [number];
row	::= "CURRENT STATE", "CONDITIONS", "EVENT", "NEW STATE" | current_state_cell, conditions_cell, state_row{state_row};
state_row	::= event_cell, new_state_cell;

Figure 3-15: Grammar of State-transition Table of Levels Extending from Level*2* to Level*n*.

Beyond providing assessment and guidance, the conditions column could also support providing the operators with timely prognoses of the future effects of failures that may occur in the earlier modes. For example, the failure state MX_FS of sub-system B (B_MX_FS) triggers a transition from state S_MX_EFS into state S_MX_PDS of system S. Accordingly, assessment and guidance are provided as "S operates degradedly, F3 cannot be delivered" and "do not order function F3", respectively.

While system S is in that state, state MY_EFS of sub-system A (A_MY_EFS) may trigger state S_MY_PDS. Accordingly, assessment and guidance will be provided as "S operates degradedly, F4 cannot be delivered" and "do not order function F4", respectively. Note how the

assessment and guidance are about function F3 after the occurrence of abnormal transition and they come to be about function F4 after the normal transition. This is because the failure state of sub-system B affects the delivery of F3 during mode MX and F4 during mode MY. This is also applicable to the state-transition tables of level*1*.

Table 3-2: State-transition Table of State-machine of System S (Shown in Figure 3-11).

CURRENT STATE	CONDITIONS	EVENT	NEW STATE
S_MX_EFS	**Assessment**: S delivers its intended functionality, F1 and F3. **Guidance**: none.	MX_FS_B == true	S_MX_PDS
		MY_EFS_A == true	S_MY_EFS
S_MX_PDS	**Assessment**: S operates degradedly and F3 cannot be delivered. **Guidance**: do not order function F3.	MX_FS_A == true	S_MX_FS
		MY_EFS_A == true	S_MY_PDS
S_MY_EFS	**Assessment**: S delivers its intended functionality, F2 and F4. **Guidance**: none.	MY_FS_B == true	S_MY_PDS
S_MY_PDS	**Assessment**: S operates degradedly and F4 cannot be delivered. **Guidance**: do not order function F4.	MY_FS_A == true	S_MY_FS
S_MX_FS	**Assessment**: S fails. **Guidance**: do not order functions F1, F2, F3 and F4.	none	none
S_MY_FS	**Assessment**: S fails. **Guidance**: do not order functions F2 and F4.	none	none

As before, this form of state-transition table is also presented as a text file that is parsed, interpreted and loaded as a number of interrelated structures for real-time reasoning. Figure 3-16 shows the declaration of the needed structures to hold the table. The difference between this declaration and the other shown in Figure 3-13 is that the *Alarm*, *Controlling* and *Diagnoses* attributes are not declared, as they are not needed in the state-transition tables of levels extending from level*2* to level*n*.

There is also no need for multi-measurement buffers at this level. There is rather a Boolean attribute (*ChildParentState*) to be instantiated with true when the parent or a child sends the corresponding state and with false otherwise.

```
structure ExpToken              //A structure of a constituent (token) of a certain expression, and holds the
  {                             //following attributes:
   String Token;                //This is to hold token itself, it might be a state of the parent or a child, a logical
                                //or a relational operator (equality), as defined in Figure 3-7.
   Boolean ChildParentState;    //Every state of the parent or a child in the expression is instantiated with true when
  };                            //it is sent by the parent or a child.

structure Expression            //A structure of an expression, for every event; it holds the following attributes:
  {
   Int FirstToken;              //The position of the first token of an expression in the Tokens array (Tokens[j]).
   Int LastToken;               //The position of the last token of the given expression in the Tokens array (Tokens[j]).
  };

structure Event                 //A structure of an event, for every event; it holds the following attributes:
  {
   Int ExpressionPosition;      //The position of the expression of the event in the expressions array (Expressions[y]).
   Int StatePosition;           //The position of the triggered state in the states array (States [n]).
  };

structure State                 //A structure of a state; for every state it holds the following:
  {
   String StateName;            //The name of the state.
   String Assessment;           //An assessment clause.
   String Guidance;             //A guidance clause.
   Int FirstEvent;    //The position of the first exit event from the state held by the events array (Events [m]).
   Int LastEvent;     //The position of the last exit event from the state held by the events array (Events [m]).
  };

State [ ] States = new State [n];          //An array to hold n state structures, where n is the number of states in the
                                           //state-transition table that appear in the current state column.
Event [ ] Events = new Event [m];          //An array to hold m event structures, where m is the number of events in
                                           //the table.
Expression [ ] Expressions = new Expression [y];   //An array to hold y expressions, where y is the number of
                                           //expressions which is either equal or less than m, as there might be more
                                           //than one event that have the same expression.
ExpToken [ ] Tokens = new ExpToken [j];    //An array to hold j ExpToken structures, where j is the number
                                           //of the tokens of the expressions.
```

Figure 3-16: Structures to hold State-transition Table of Levels Extending from Level*2* to Level*n*.

3.6.3 Diagnostic Model

Failures and their underlying causes are not always in a one-to-one relationship. Accurate diagnosis of the causes requires, therefore, a diagnostic model to record firstly, the potential faults of the basic components and paths in which consequent errors propagate and cause the failures at level*1*; secondly, symptoms associated with the propagation path of each fault. By traversing the diagnostic model and examining the symptoms, the exact propagation path can be tracked starting from its exposure as a failure event towards its causes. In such an approach the causes are tracked downwardly from level*1*, where the failure is detected, towards the basic components at level*0*, where the errors are originated.

As discussed in Chapter Two, model-based diagnosis approaches have developed a number of effective diagnostic models. Among them is the fault tree, whose effectiveness as a diagnostic and fault propagation model has been demonstrated by Papadopoulos (2002, 2003 and 2000). Accordingly, the fault tree is considered as a diagnostic model in the context of this work.

Moreover, in the work of Feiler and Rugina (2007), a number of state-machines have proved their success as fault propagation models. This has also been demonstrated by Joshi *et al.* (2007a, b), where a fault tree is derived from a number of state-machines. Thus, in this work, state-machines are suggested as an alternative type of diagnostic model.

3.6.3.1 Fault Tree as a Model-based Diagnosis Approach

Figure 3-17 shows the basic notation to model a graphical fault tree. A fault tree can be constructed of three different types of nodes; or-node (ORN), and-node (ANDN) and basic event node (BEN). Nodes are arranged in different levels and in terms of parents, siblings, and children.

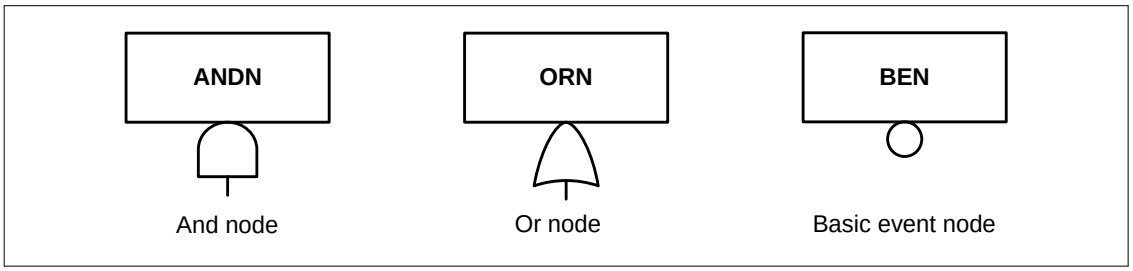


Figure 3-17: Notation of the Fault tree Model.

ORNs and ANDNs enclose symptoms that verify the correct track of error propagation paths and also show how the logical combination of their inputs can result in the enclosed symptom. BENs enclose (a) symptoms that verify the occurrence of the faults; (b) references to the fault and the name of the basic components; (c) corrective measures. Every node may appear either as an ANDN or ORN and has further lower level children or as BEN and has no children. The top node of the fault tree can be ANDN or ORN and encloses a reference to the ultimate failure event caused by the BENs. By exploiting the logical connection between the ANDNs and ORNs and their children, the fault tree can illustrate the possible propagation paths and associated symptoms of the fault enclosed by BENs.

To demonstrate the ability of the fault tree to serve as a diagnostic model, Figure 3-18 revisits the condensing sub-system[1] (CS) and captures an excerpt of the sub-system's behavioural model and a fault tree of the failure event "No flow at the outlet". The figure shows how the fault tree relates the failure event (detected at level*1*) to its underlying causes, which could be any of the following faults: PS is off, VP is closed, outlet line is blocked and outlet line is leaky. For those faults the fault tree incorporates also corrective measures, e.g. "*open VP*" enclosed by the "*BEN_2*".

[1] As shown in Figure 3-6.

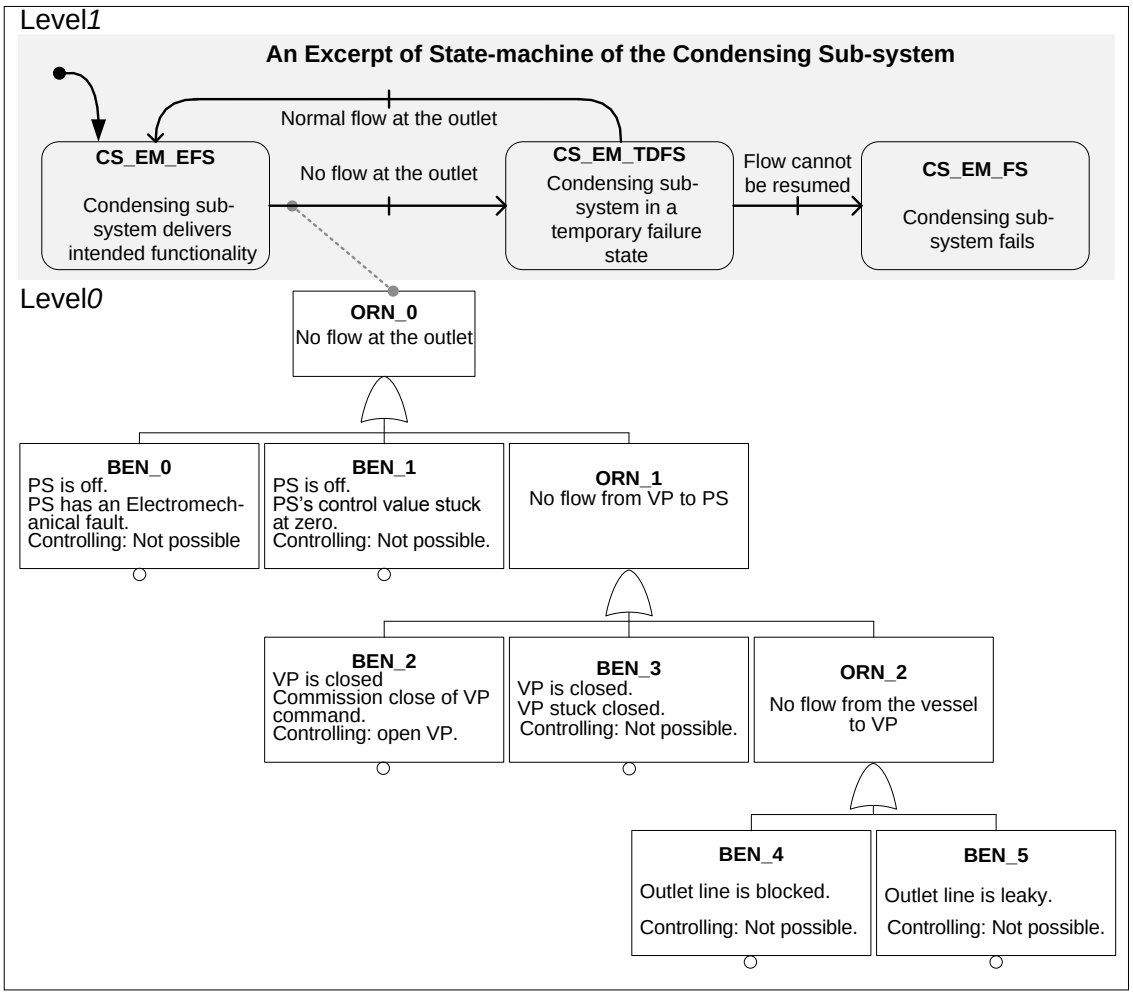


Figure 3-18: An Excerpt of State-machine and a Fault tree of Condensing Sub-system.

The figure shows also that the failure event "No flow at the outlet" triggers a state transition from the EFS of the emptying mode (CS_EM_EFS) to the TDFS of the same mode (CS_EM_TDFS). It should be pointed out that an analytical redundancy (section 3.6.1.3) is exploited to validate the sensory measurements of the sub-system and thus faults of the sensors are not modelled in the fault tree.

Deriving Formal Diagnostic Model from a Fault Tree

Using E-BNF notation, Figure 3-19 presents the formal grammar of the diagnostic model derived from a fault tree. The grammar defines the diagnostic model as a *fault tree* that consists of a top node (*top_node*) and a number of nodes (*nodes*). The top node encloses the following:

- Name of the node, which consists of the term *NodeName:* and the name of the node (*node_name*). The name of the node consists of two parts. The first indicates the type of node which could be one of the three terms: ANDN, ORN and BEN, while the second is a number. The separation between the two parts is made by an underscore.

- Symptom, which consists of the term *Symptom:* and a monitoring expression (*expression*) that is defined as shown in Figure 3-8. This expression is the same as the one formalised to the failure event that appears in the transition table of level*1*. Hence, every failure event can be related to its diagnostic model while parsing and loading the state-transition table and the diagnostic model; the monitoring model of a sub-system of level*1*.
- Child name, which consists of the term *ChildName:* and the name of the first-left-side node that is *node_name* (*node_name* is as explained above).

```
diagnostic_model        ::= fault_tree;

fault_tree              ::= top_nde, nodes;

top_node                ::= "NodeName:", node_name, ".",
                            "Symptom:" expression, ".",
                            "ChildName:", node_name;

node_name               ::= "ANDN_" | "ORN_" | "BEN_", number;

number                  ::= digit, {digit};

digit                   ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;

nodes                   ::= {ORN}, {ANDN} , {BEN};

ORN                     ::= "NodeName:", node_name, ".",
                            "symptom:", expression | "none", ".",
                            "Child:", node_name, ".",
                            "Sibling:", node_name | "none", ".";

ANDN                    ::= "NodeName:", node_name, ".",
                            "symptom:", expression | "none", ".",
                            "Child:", node_name, ".",
                            "Sibling:", node_name | "none", ".";

BEN                     ::= "NodeName:", node_name, ".",
                            "Symptom:", expressionme | "none", ".",
                            "Fault:", (*fault and name of the faulty component*), "."
                            "Controlling:", "-", corrective_measure | "none", ".",
                            "Sibling:", node_name | none, ".";

corrective_measure      ::= "-", actuator_identifier, "=", measure;

measure                 ::= boolean | integer | real;
```

Figure 3-19: Grammar of Diagnostic Model Derived from Fault Tree.

The number of nodes of the fault tree (*nodes*) may appear as a combination of *ANDN*, *ORN* or *BEN*. Both *ANDN* and *ORN* have similar enclosures that can be listed as follows:

- Name of the node, which is as defined above.
- Symptom, which consists of the term *Symptom:* and either a monitoring expression as defined in Figure 3-8 or *none* when there is no sensor to provided measurements that can correspond to form an expression.
- Child, which is as same as the child defined for the top node.
- Sibling, which consists of the term *Sibling:* and either the name of the first-right-side node, (the name of this node is as explained above) or *none* when the node has no siblings.

The node name, symptom and sibling enclosed by the basic event node (*BEN*) are the same as those of the *ANDN* and *ORN*. Moreover, as the *BEN* has no children, it does not enclose a child name and rather has the following additional enclosures:

- Fault, which consists of the term *Fault:* and a clause that names the fault and the name of the faulty basic component (**Fault and name of the faulty component**).
- Controlling, which consists of the term *Controlling:* and either corrective measure (*corrective_measure*) when the fault is correctable or *none* otherwise. Corrective measures appear as an actuator identifier (*actuator_identifier*), equality (=) and a measure (*measure*). *Measure*, in turn, could be *Boolean, integer* or *real*.

According to the grammar, Figure 3-20 shows a diagnostic model derived from the fault tree of Figure 3-18.

NodeName: ORN_0.
Symptom: T (|FM| < 0.03, 3 sec).
ChildName: BEN_0.

NodeName: BEN_0.
Symptom: |PS| < 20.
Fault: PS has an electromechanical fault.
Controlling: none.
Sibling: BEN_1.

NodeName: BEN_1.
Symptom: |PS| < 20.
Fault: PS's control value stuck at zero.
Controlling: none.
Sibling: ORN_1.

NodeName: ORN_1.
Symptom: none.
Child: BEN_2.
Sibling: none.

NodeName: BEN_2.
Symptom: VP == 0.
Fault: commission close of VP command.
Controlling: VP = 1.
Sibling: BEN_3.

NodeName: BEN_3.
Symptom: VP == 0.
Fault: VP stuck close.
Controlling: none.
Sibling: ORN_2.

NodeName: ORN_2.
Symptom: VP == 1 and |PS| > 20.
Child: BEN_4.
Sibling: none.

NodeName: BEN_4.
Symptom: none.
Fault: outlet line is blocked;
Controlling: none.
Sibling: BEN_5.

NodeName: BEN_5.
Symptom: none.
Fault: outlet line is leaky.
Controlling: none.
Sibling: none.

Figure 3-20: Formal Diagnostic Model Derived from Fault Tree of Figure 3-18.

Traversal of a Diagnostic Model Derived From a Fault Tree

Similar to the behavioural model, this form of diagnostic model is presented as a text file, which in turn is parsed, interpreted and loaded to real-time reasoning. According to the grammar defined in Figure 3-19, Figure 3-21 shows the declaration of the needed structures to hold the model.

```
enumerate NodeType                  //A numeration of the possible types of the nodes.
   { ORN, ANDN, BEN };

structure Symptom                   //A structure of a symptom (an expression). For every symptom it holds the following
   {                                //attributes:
    String SymptomStatus;           //To be instantiated with "none" or "expression", according to the symptom status.
    Int FirstSymToken;              //The position of the first token of a given symptom in the tokens array.
    Int LastSymToken;               //The position of the last token of that symptom in the tokens array.
   };

structure Node                      //A structure of a node. For every node it holds the following:
   {
    NodeType Type;                  //The type of the node as numerated above.
    Int SymptomPosition;            //The position of the structure of the enclosed symptom in the symptoms array
                                    //(Symptoms[t]).
    String Fault;                   //The fault clause enclosed by the BEN.
    String MeasureStatus;           //To be instantiated with "none" or "measure", according to the controlling status.
    Int FirstMeasToken;             //The position of the first token of a given measure, if the MeasureStatus is not none.
    Int LastMeasToken;              //The position of the last token of that measure, if the MeasureStatus is not none.
    Int ChildPositon;               //The position of the structure of the first-left-side child node.
    Int SiblingPosition;            //The position of the structure of the first-right-side sibling node.
    Int FirstNodePosition;          //To hold the position of the structure of the first node of the tree in the nodes array
                                    //(Nodes[s]).
    Int LastNodePosition;           //To hold the position of the structure of the last node of the tree in the nodes array
   };                               //(Nodes[s]).

Node [ ] Nodes = new Node [s];          //An array to hold s nodes, where s is the number of the nodes of the
                                        //trees.
Symptom [ ] Symptoms = new Symptom [t]; //An array to hold t symptoms, where t is the number of the symptoms
                                        //of the trees.
```

Figure 3-21: Data Structures to hold Diagnostic Model Derived from a Fault Tree.

The declaration enumerates firstly the type of node; ORN, ANDN and BEN. It then shows that every diagnostic model is held as a number of nodes. For every fault tree there is a number of structures to hold the nodes (*Nodes*). The structures are held by an array called *Nodes [s]*, where *s* is the number of the nodes of the diagnostic models of a sub-system. The *Node* structure has the following attributes:

- *NodeType*, which holds the type of the node, while interpreting the diagnostic model, node type is derived from the first part of the node name.
- *SymptomPosition*, which holds the position of the structure of every symptom enclosed by the nodes. The structure of the symptom (*Symptom*), in turn, has three attributes: *SymptomStatus*, *FirstSympToken* and *LastSympToken*. While interpreting and loading the model, if the symptom is *none* then *SymptomStatus* is instantiated with *none* and the other two attributes with *nil*. On the contrary, if the symptom is an expression then the

SymptomStatus and *FirstSympToken* are instantiated with the position of the first and last positions of tokens of that expression in the array of symptoms tokens. Some nodes may have the same symptom; accordingly the *SymptomPosition* of those nodes are instantiated with the position of the structure of the common symptom.

- *Fault* to hold the name and the fault of the faulty component of the BEN; it holds *none* for the ANDNs and ORNs.
- *MeasureStatus*, *FirstMeasPosition* and *LasstMeasPosition*, these three attributes belong to the corrective measure enclosed by every BEN. If the node is ANDN or ORN then the *MeasureStatus* is instantiated with *none* and the other two attributes with *nil*. Similar instantiation is achieved when the node is BEN and it encloses *none* measurement. On the contrary, if the node is BEN and it encloses a measure, then *FirstMeasPosition* and *LasstMeasPosition* are instantiated with the position of the first and last token of that measure in an array of measures tokens.
- *ChildPosition*, which holds the position of the structure of the first-left-side child when the node is either ANDN or ORN and *nil* when the node is a BEN, as typically it does not have children.
- *SiblingPosition*, which holds the position of the structure of the first-right-side sibling or *nil* when the node has no sibling.
- *FirstNode* and *LastNode*, which hold the positions of the first and last nodes of a fault tree in the array *Nodes[s]*. These positions are used later to update the symptoms of the diagnostic model in the correct time, which is immediately after verifying the occurrence of the relevant failure event[1].

While loading the monitoring model, the position of the top node of the tree instantiates the attribute[2] *DiagnosticModelPosition* in the relevant *Event* structure (shown in Figure 3-13). Figure 3-22 shows the pseudo code of the diagnostic algorithm that can traverse a diagnostic model derived from a fault tree. The code is augmented with comments to explain each of its actions. To traverse the model, a heuristic traverse is initiated for every faced node by examining the enclosed symptoms. This traverse is further supported by a blind-depth-first traversal as follows:

- If the current node is either ANDN or ORN and its enclosed symptom is verified true or has no expression (none), then the blind-depth-first traverse is initiated to its first-left-side child. If, on the contrary, the enclosed symptom is verified false, then there are no further nodes to be traversed.

[1] Use of these attributes can be seen in the pseudo code that associates plan L_1-P_{17} in section 4.7.2.
[2] Use of this attribute can be seen in the pseudo code that associates plan L_1-P_{26} in section 4.7.2.

- If the current node is BEN and its enclosed symptom is verified true or the symptom is *none*, then it is diagnosed as a cause node. Otherwise, if the enclosed symptom is verified false the blind-depth-first traverse is initiated to its first-right-side sibling.

The diagnostic process is launched after verifying the occurrence of the failure event at level*1* and by calling the diagnostic algorithm with the position of the top node (as argument of the diagnostic function). The Identification of the position of the top node and the call of the diagnostic function are achieved by the monitoring agent and this will be explained further in section 4.7.2.

```
Boolean diagnosis (int CNode)                  //The diagnostic process is launched based on the position of the
{                                              //tope node (CNode).
 SPosition = Nodes [CNode]. SymptomPosisition;  //The position of the structure of the enclosed symptom is retrieved.
 SStatus = Symptoms [SPosition]. SymptomStatus; //From the structure the status of the symptom is retrieved.
 if (SStatus != none)                          //If the status is not none; there is an expression to be evaluated.
  {
   FToken = Symptoms [SPosition]. FirstSymToken;  //Retrieve the position of the first token of the expression.
   LToken = Symptoms [SPosition]. LastSymToken;   //Retrieve the position of the last token of the expression.
   SEvaluation = Evaluation (FToken, LToken);     //Evaluate the expression of the symptom and hold the result in the
  }                                               //Boolean variable SEvaluation.
 if (SEvaluation || SStatus == none)             //If the evaluation is true or the symptom has no expression then
  {
   if (Node [CNode].Type == BEN)                 //If moreover the type of the current node (CNode) is BEN.
    {
     DiagnosticsArray [a] = CNode;               //Add the position of the current node to the array that holds the
                                                 //positions of the causal BENs.
     a= a + 1;                                   //Increase the counter of the array positions to be ready to hold
                                                 //another cause (if there is any)
     return true;                                //return true for the diagnosis function.
    }
   else                                          //If the current node is not a BEN; either an ORN or ANDN, then.
    {
     NPosition = Nodes [CNode]. ChildPosition;   //Retrieve the position of the first-left-side child.
     NType = Nodes [CNode]. Type;                //Retrieve the type of the CNode, whether an ANDN or an ORN
     while (NPosition != nil)                    //For every child of the CNode.
       {
        Result = diagnosis (NPosition);          //Recursively call diagnosis function
        NPosition = Nodes [NPosition].SiblingPosition; //Retrieve the position of the first-right-side sibling.
        if (Result && NType == ORN) return true; //If the function returns true and the parent node is an ORN then
                                                 //return true; the underlying cause has been diagnosed.
        if (!Result && NType == ANDN) return false; //If the function return false and the parent node is an ANDN then
                                                    //return false as this path is not the error propagator.
        return Result;                           //Return the result, if the otherwise.
       }
    }
  }
 else return false                               //If the evaluation of the symptom is false then return false.
}
```

Figure 3-22: Pseudo-code of Diagnostic Algorithm of a Fault Tree Diagnostic Model.

3.6.3.2 State-Machine as a Model-based Diagnosis Approach

To demonstrate the ability of state-machines to serve as a diagnostic model, Figure 3-23 revisits the condensing sub-system (CS), shown in Figure 3-6. Figure 3-23 captures an excerpt of the sub-system's behavioural state-machine and a number of diagnostic state-machines that can relate the failure event "No flow at the outlet" (of the condensing sub-system shown in Figure 3-6) to its underlying causes.

The behavioural state-machine at the top of Figure 3-23 shows how the failure event triggers a state transition from the EFS of the emptying mode (CS_EM_EFS) to the TDFS of the same mode (CS_EM_TDFS). It should be pointed out that an analytical redundancy (section 3.6.1.3) is exploited to validate the sensory measurements of the sub-system and thus faults of the sensors are not modelled in the diagnostic state-machines.

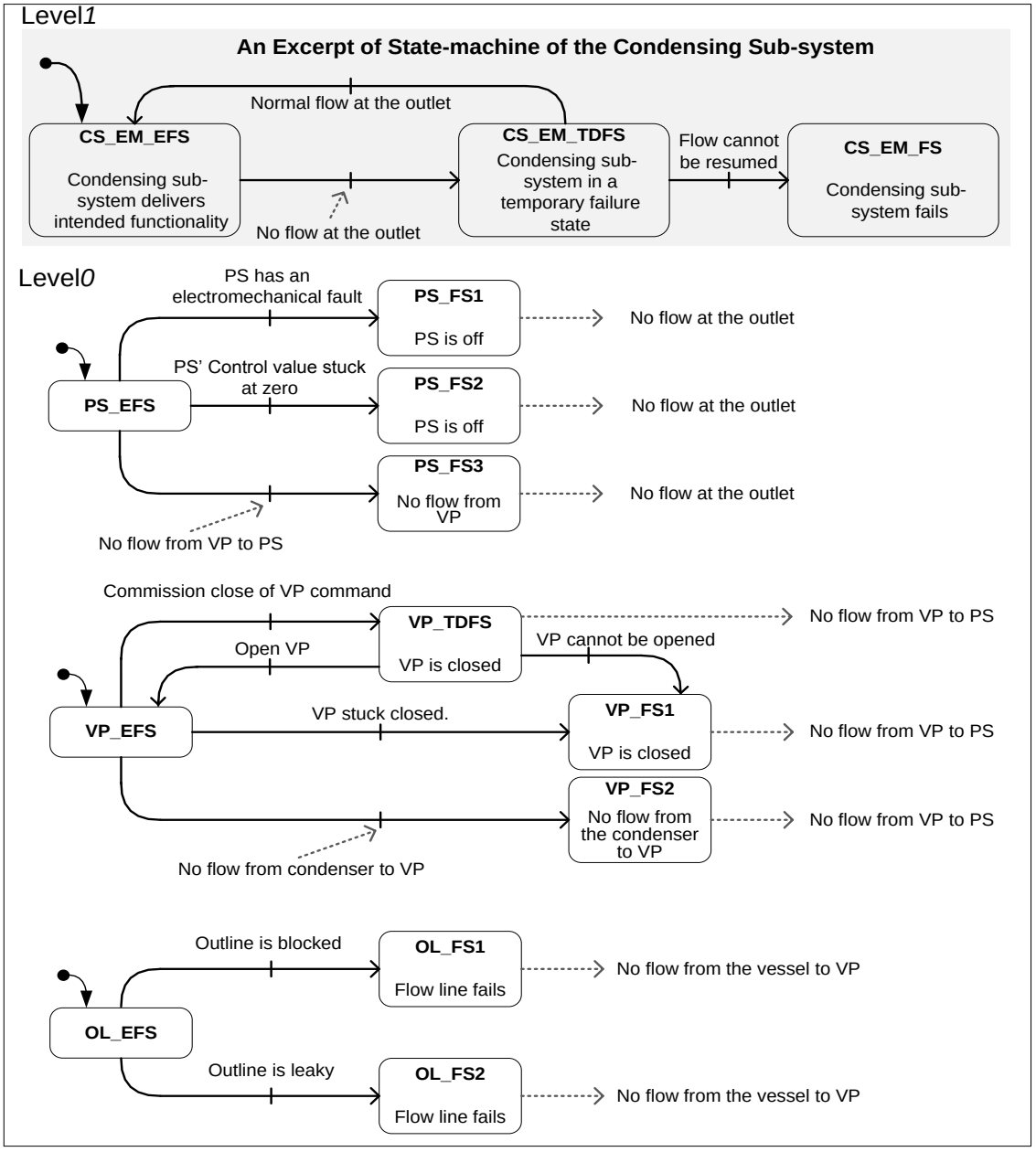


Figure 3-23: Diagnostic State-machines of "No Flow at the outlet" of Condensing Sub-system.

Similar to the fault tree, state-machines can also relate the failure event "No flow at the outlet" to its possible underlying causes, which can be any of the following faults: PS is off, VP is closed, line is blocked and line is leaky. For those faults, the state-machines incorporate also corrective measures, e.g. controlling field "open VP" enclosed by the BEN of "VP is closed". Propagation paths among the state-machines are represented by dotted hollow-headed arrows.

Deriving Formal Diagnostic Model form State-machines

Using E-BNF notation, Figure 3-24 presents the formal grammar of the diagnostic model derived from a number of state-machines. In this grammar, the diagnostic model is defined as a *state-machines* model consisting of a *failure event* and *propagator*. The *failure event* consists of the term "*FailureEvent:*" and expression of the failure event of level*1*. The *propagator* consists of the term "*Propagator:*" and an *error*. The *error*, in turn, consists of an error state (*estate*) or a logical combination of a number of error states.

diagnostic_model	::= state_machine;
state_machine	::= failure_event, propagator;
failure_event	::= "FailureEvent:", expression, ".";
propagator	::= "Propagator:", error;
error	::= estate {logical_operator, estate};
estate	::= propagated_state | fault_state;
propagated_state	::= "EStateName:", estate_name "Symptom:" expression | "none", ".", "Propagator:", error, ".";
fault_state	::= "EStateName:", estate_name, "Symptom:" expression | "none", ".", "Fault:", (*fault and name of the faulty component*), ".", "Controlling:", corrective_measure | "none" ".";
estate_name	::= component_initials, "_", estate_initials [number];
estate_initials	::= TDFS | PDS | FS;
number	::= digit{digit};
digit	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
corrective_measure	::= "-", actuator_identification, "=", measure;
measure	::= boolean | integer | real;
logical_operator	::= "AND" | "OR" ;

Figure 3-24: Grammar of Diagnostic Model Derived from State-machines.

An error state (*estate*) could be a *propagated state* or a *fault state*. While the former is triggered by an error propagated from error states of other components, the latter is triggered by an internal fault of the given component. The enclosures of those states differ accordingly. A *propagated_state* encloses the following:

- The name of the error state, which consists of the term *EStateName* and the name of the error state (*estate_name*). The *estate_name* is composed of two parts separated by an underscore; the first part is the initials of the component name; and the second part is the initials of an error state (TDFS, PDS or FS) associated with a number, as every state might appear more than once.

- Symptom, which consists of the term *Symptom:* and either a monitoring expression as defined in Figure 3-8 or *none* when there are no sensors to provide measurements that can correspond to form an expression.
- Propagator, which consists of the term *Propagator:* and error which is as defined above.

A *fault_state* encloses the name of the error state and symptom, the same as those of the *propagated_state*. A *fault_state* does not have a propagator and rather encloses the following:

- Fault, which consists of the term *Fault:* and a clause that names the fault and the name of the faulty basic component (**Fault and name of the faulty component**).
- Controlling, which consists of the term *Controlling:* and either corrective measure (*corrective_measure*) when the fault is correctable or *none* otherwise. Corrective measures appear as an actuator identifier (actuator_identifier), equality (=) and a measure (*measure*). *Measure*, in turn, could be a *Boolean*, *integer* or *real*.

Figure 3-25 shows the executable diagnostic model derived from the state-machines of Figure 3-23.

FailureEvent: T(|FM| < 0.03, 3 sec).
Propagator: PS_FS1 **OR** PS_FS2 **OR** PS_FS3.

EStateName: PS_FS1.
Symptom: |PS| < 20.
Fault: PS has an electromechanical fault.
Controlling: none.

EStateName: PS_FS2.
Symptom: |PS| < 20.
Fault: PS's control value stuck at zero.
Controlling: none.

EStateName: PS_FS3.
Symptom: none.
Propagator: VP_TDFS **OR** VP_FS1 **OR** VP_FS2.

EStateName: VP_TDFS.
Symptom: VP == 0.
Fault: commission close of VP command.
Controlling: VP = 1.

EStateName: VP_FS1.
Symptom: VP == 0.
Fault: VP stuck close.
Controlling: none.

EStateName: VP_FS2.
Symptom: VP == 1 **AND** |PS| > 20.
Propagator: OL_FS1 **OR** OL_FS2.

EStateName: OL_FS1.
Symptom: none.
Fault: flow line is blocked;
Controlling: none.

EStateName: OL_FS2.
Symptom: none.
Fault: flow line is blocked.
Controlling: none.

Figure 3-25: Formal Diagnostic Derived from State-machines of Figure 3-23.

Traversal of a Diagnostic Model Derived from State-machines

Similar to the diagnostic model derived from a fault tree, the model derived from state-machines is also presented as a text file to be parsed, interpreted and loaded as a number of interrelated structures for the real-time diagnostic process. According to the grammar defined in Figure 3-24, Figure 3-26 shows the declaration of the structures needed to hold this model.

```
structure Propagator          //A structure to hold errors that are propagated and result in the failure event or an
  {                           //error state of another component. This is applicable to the propagated_state and not
                              //to the fault state, as defined by Figure 3-23.
   Int FirstPropToken;        //The position of the first token of the error propagator in the token array.
   Int LastPropToken;         //The position of the last token of the error propagator in the token array.
  };                          // A token could be an AND, OR or the position of an error state of another component.

structure Symptom             //A structure of a symptom (an expression). For every symptom it holds the following
  {                           //attributes:
   String SymptomStatus;      //To be instantiated with "none" or "expression", according to the symptom status.
   Int FirstSymToken;         //The position of the first token of a given symptom in the toke array.
   Int LastSymToken;          //The position of the last token of that symptom in the toke array.
  };

structure State               //A structure of a state; it holds the following attributes:
  {
   Int SymptomPosition;       //The position of the structure of the enclosed symptom.
   String Fault;              //The fault filed of the fault state.
   String MeasureStatus;      //The status of the measure, either none or measure.
   Int FirstMeasToken;        //The position of the first token of a given measure in the measures token array.
   Int LastMeasToken;         //The position of the last token of that measure in the measures token array.
   Int StatePropagator;       //The position of the structure of the error propagators of the propagated state, as
  };                          //defined by Figure 3-23.

structure StateMachine        //A structure of the state-machine of every failure events of a sub-system, it holds
  {                           //the diagnostic model through the following attributes:
   Int EventPropagator;       //The position of structure of the propagator that propagate the failure event.
   Int FirstStatePosition;    //To hold the position of the structure of the first state of the diagnostic model, in the
                              //array of states (States[s]).
   Int LastStatePosition;     //To hold the position of the structure of the last state of the diagnostic model, in the
  };                          //array of states (States[s]).

StateMachine [ ] StateMachines = new StateMachine [r]; //An array to hold r models, where r is the number of the
                                                       //diagnostic models of a sub-system.
State [ ] States = new State [s];          //An array to hold s states, where s is the  number of states of the
                                           //diagnostic models.
Symptom [ ] Symptoms = new Symptom [t];    //An array to hold t symptoms, where t is the number of
                                           //symptoms of the diagnostic models.
Propagator [ ] Propagators = new Propagator [t]; //An array to hold t propagator fields, where t is the number of
                                                 //propagator fields of the diagnostic models.
```

Figure 3-26: Data Structures to Hold Diagnostic Model Derived from State-machines.

The declaration shows that for every diagnostic model there is a structure to hold the state-machines (*StateMachine*) and a number of structures to hold the error states (*State*) of that model. As there could be a number of models for every sub-system, the structure *StateMachine* is held by an array *StateMachines [r]*, where *r* is the number of diagnostic models. Similarly, a

number of error state structures are held by an array *States [s]*, where *s* is the number of states of the diagnostic models of a sub-system.

Every *StateMachine* structure has the following attributes:

- *EventPropagator*, which holds the position of an error state or positions of error states that are combined logically and propagate the verified failure event.
- *FirstStatePosition* and *LastStatePosition*, which hold the positions of the structures of the first and last states of the diagnostic model in the States array; *States [s]*. These positions are used later to update the symptoms of the diagnostic model at the correct time[1], which is immediately after verifying the occurrence of the relevant failure event.

While loading the monitoring model, the position of the structure of the *StateMachine* instantiates the attribute[2] *DiagnosticModelPosition* in the relevant *Event* structure (shown in Figure 3-13)

Every *State* structure has the following attributes:

- *SymptomPosition*, which holds the position of the structure of every symptom enclosed by the states. The structure of the symptom (*Symptom*), in turn, has three attributes: *SymptomStatus*, *FirstSympToken* and *LastSympToken*. While interpreting and loading the model, if the symptom is *none* then *SymptomStatus* is instantiated with *none* and the other two attributes with *nil*. On the contrary, if the symptom is an expression then the *SymptomStatus* and *FirstSympToken* are instantiated with the position of the first and last positions of tokens of that expression in the array of symptoms tokens. Some states may enclose the same symptom; accordingly the *SymptomPosition* of those states are instantiated with the position of the structure of the common symptom.
- *Fault* to hold the name and the failure mode of the faulty component of the fault state; it holds *none* of the propagated states.
- *MeasureStatus*, *FirstMeasPosition* and *LasstMeasPosition*; these three attributes belong to the corrective measure enclosed by every *fault_state*, as defined in Figure 3-24. If the state is a *propagated_state* then the *MeasureStatus* is instantiated with *none* and the other two attributes with *nil*. Similar instantiation is achieved when the state is a *fault_state* and enclose *none* measurement. On the contrary if the state is a *fault_state* and encloses a measure then *FirstMeasPosition* and *LasstMeasPosition* are instantiated with the position of the first and last token of that measure in the array of measures tokens.
- *StatePropagator*, for the *propagated_states*; this attribute holds the position of the *Propagator* structure. This structure holds the error state or a logical combination of a

[1] Use of these attributes can be seen in the pseudo code that associates plan L_1-P_{17} in section 4.7.2.
[2] Use of this attribute can be seen in the pseudo code that associates plan L_1-P_{26} in section 4.7.2.

number of error states that propagate and trigger the given state. The *Propagator* structure has two attributes, *FirstPropToken* and *LastPropToken*. For the number of propagators enclosed by the state, the *Propagator* structure is held by an array *Propagators[t]*, where *t* is the number of propagators.

Figure 3-27 shows the pseudo code of the diagnostic algorithm that can traverse a diagnostic model derived from a number of state-machines. The code is also augmented with comments to explain each of its actions. Similar to the diagnostic algorithm of the fault tree, this algorithm also combines between blind-depth-first and heuristic traverses to relate a failure event to its underlying causes.

```
Boolean Diagnosis (Int EP)          //EP is the position of the propagator, which is a state or logical combination of
{                                   //states that results in the failure event (when this function is firstly called) or a state
                                    //or logical combination of states that results in an error state of a basic component
                                    //(when this function is called recursively)
 FPToken = Propagator [EP]. FirstPropToken;  //Retrieve the first position of the propagator in the PToken array.
 LPToken = Propagator [EP]. LastPropToken;   //Retrieve the last position of the propagator in the PToken array.
 For (i = FPToken, i <= LPToken, i++)        //From the first to the last positions.
  {
   RetrievedToken = PTokenArray[i];          //Retrieve a token
   if (RetrievedToken == AND) Logic = AND;   //If the token is an AND, instantiate the variable "Logic" with an AND.
   if (RetrievedToken == OR) Logic =OR;      //If the token is an OR, instantiate the variable "Logic" with an OR.
   else                                      //Otherwise, if the token is a position of an error state.
    {
     SPosition = States [RetrievedToken]. SymptomPosisition;      //From the position of the structure, the
                                                                  //enclosed symptom is retrieved.
     SStatus = Symptoms [SPosition]. SymptomStatus;   //From the structure the status of the symptom is retrieved.
     if  (SStatus != none)                            //If the symptom is an expression then
     {
      FEToken = State [RetrievedToken]. FirstSymToken; //Retrieve the position of the first token of the expression
                                                       //from the ExpArray.
      LEToken = State [RetrievedToken]. LastSymToken; //Retrieve the position of the last token of the expression
                                                      //from the ExpArray.
      SEvaluation = Evaluation (FEToken, LEToken);    //Evaluate the expression.
     }
     if (SEvaluation || SStatus == none)      //The evaluation results in true or the status of the symptom is none
     {
      if (State [RetrievedToken].Fault != none)          //If moreover the current state is a fault state then
      {
       DiagnosticArray [a] = RetrievedToken; //Add the position of this state to the diagnostic array
       a = a +1;                             //Increase the counter of the positions of the diagnostics array by 1.
       return true;                          //And return true.
      }
      else                          //Otherwise, if the state is a propagated state
      {
       Result = Diagnosis (State [RetrievedToken]. StatePropagartor);          //Recursively call the diagnosis.
       if (Result && Logic == OR) return true;          //If the Result is true and the logic between this state and
                                                        //previously traversed one is OR then return true.
       if (! Result && Logic == AND) return false;      //If the Result is true and the logic between this state and
                                                        //previously traversed one is AND then return false.
       return Result;                                   //Otherwise, return the Result.
      }
     }
     else return false;             //If the evaluation of the enclosed symptom is false then return false.
    }
  }
}
```

Figure 3-27: Pseudo code of the Diagnostic Algorithm of Model Derived from State-machines.

3.7 Summary

The chapter has discussed the position, role, architecture and constituents of the proposed monitor. A distributed monitoring model and multi-agent system have been shown as the two constituents of the monitor. The distributed monitoring model has been developed in this chapter. The distributed monitoring model is constructed from behavioural and diagnostic knowledge of the monitored system. To derive this knowledge, system design models and a number of safety analysis techniques have been exploited.

It was shown that this knowledge can be derived via application of state-of-the-art model-based safety assessment techniques[1], such as HiP-HOPS and AADL. More specifically, HiP-HOPS produces assessment models consisting of a hierarchy of state-machines as a behavioural model and a number of fault trees as error propagation models (Papadopoulos, 2002). AADL produces an assessment model consisting of a hierarchy of state-machines as a behavioural model and another set of state-machines as error propagation models (Feiler and Rugina 2007; Debruyne *et al.*, 2005). To make such assessment models executable for the purposes of safety monitoring, their behavioural and error propagation models were formalised as state-transition tables and diagnostic models, respectively.

The chapter also discussed the importance of formalising events and symptoms using precise monitoring expressions. Techniques that can filter and validate sensory measurements and diagnose and control sensory faults have also been developed.

Though they share some commonalities, HIP-HOPS and AADL differ. The specific derivation of a monitoring model from HiP-HOPS and AADL models and case studies are therefore demonstrated in Chapter Five and Chapter Six, respectively.

[1] Assessment models produced by these techniques are typically developed in two parts; the first is a technical assessment of the system and the second describes the fault controlling measures to avoid disastrous scenarios (Kelly, 1998). Thus, it could be said that in addition to the behavioural and diagnostic knowledge, such models could also provide non-conflicting corrective measures and quality of assessment and guidance.

Chapter Four

Distributed On-line Safety Monitor: Multi-agent System

4.1 Introduction

This chapter develops the multi-agent system, the second constituent of the proposed monitor. In this chapter, the appropriateness of Belief Desire Intention (BDI) agents as monitoring agents is considered and the deployment architecture of those agents as well as collaboration protocols among them are developed.

The monitoring roles, reasoning and implementation of the agents are then discussed. The chapter finally shows how the logical omniscience problem is considered and addressed during the design and implementation of the monitoring agents.

4.2 Belief-Desire-Intention (BDI) Agents as Monitoring Agents

The reasoning foundation of the BDI agent is inspired by the theory of human practical reasoning. Belief (B) is a mental attitude that abstracts "what is going on" in the environment surrounding the agent, which must continually perceive changes that occur over time and update belief accordingly. Desire (D) is another mental attitude that represents motivations that the agent reasons to bring about. Intention (I) is the third mental attitude and it is a desire to which the agent has committed and it will act to bring about (Bratman, 1978; Bordini *et al.*, 2007).

The computational model of a BDI agent is constituted of firstly, a belief base, which is a collection of literals updated continuously by adding the up-to-date belief and deleting the no longer valid one, and secondly, a plan library representing the desires of the agent. Each plan is a set of actions that are executed either to affect matters in the environment or to exchange messages among the agents (in a multi-agent system) to influence each other's beliefs.

In its practical reasoning, the agent reasons among its beliefs and desires cyclically. Every cycle, which is called a *reasoning cycle*, is triggered by a new belief and achieved in two processes. The first is the deliberation, in which the agent deliberates among its desires to select an intention (one or a set of plans). The second process is the means-ends, in which the agent selects a plan to be executed (Shoham, 1993; Bordini *et al.*, 2007). Figure 4-1 illustrates the three mental attitudes and the reasoning cycle of the BDI agent.

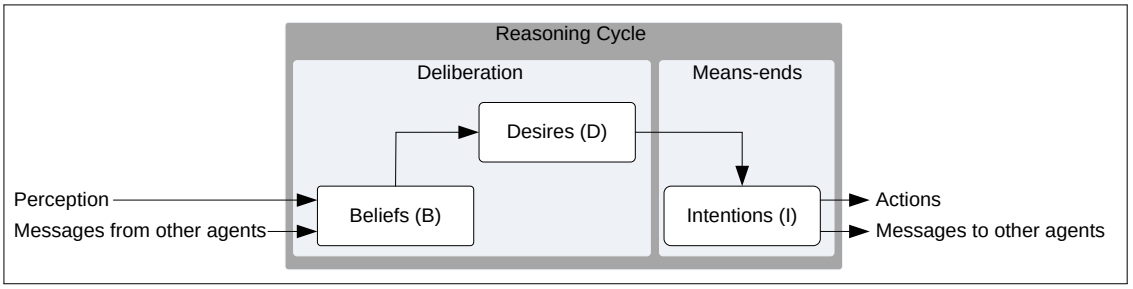


Figure 4-1: Reasoning Cycle of the BDI-agent.

In addition to the common ability of intelligent agents to achieve integrated reasoning among distributed processes (McArthur *et al.*, 2004), two more reasons underpin the particular adoption of BDI agents[1] as monitoring agents. Firstly, as the reasoning model of these agents is based on human reasoning, effective automation of the crucial responsibilities of system operators can be facilitated. Secondly, the informative communication[2] as well as the semi-independent reasoning of the BDI agents can support effective collaboration and integration of two different deployment approaches:

- Spatial deployment in which agents are installed on a number of distributed computational machines. Such deployment is needed when the sub-systems of the monitored system are distributed over a geographical area, e.g. a chemical plant.
- Semantic deployment in which monitoring agents are installed on one computational machine. Such deployment is appropriate when the sub-systems of the monitored system, although distributed, are close to each other, e.g. an aircraft system.

4.2.1 The Deployment of the Monitoring Agents

Monitoring agents are deployed hierarchically over the monitored system and its sub-systems following the corresponding structure of the distributed monitoring model. The agent that is deployed over the top hierarchical level (level*n*) is termed the system monitoring agent (S_MAG). Agents that are deployed over the sub-systems of levels extending from level*1* to level*n-1* are termed sub-system monitoring agents (Ss_MAGs). Thus, the multi-agent system appears as a hierarchy of monitoring agents and the relationships among the agents are as parent and children, or as siblings (when two agents lie on the same level of the architectural refinement).

Every monitoring agent is provided with a corresponding model (or portion) of the distributed monitoring model. Guided by knowledge encoded in their models, agents reason locally to

[1] In addition to the BDI reasoning model, there are moreover three different models: the logic-based reasoning model, the reactive reasoning model and the layered reasoning model. More information about these models can be found in Bellifemine *et al.* (2007).

[2] More information about the communication abilities of the BDI agents can be found in Bordini *et al.* (2007).

monitor the conditions of the components and collaborate to achieve global integration and deliver consistent safety tasks over the monitored system.

In Chapter Three, level*1* was selected as the most appropriate level for effective monitoring process (section 3.4). This has implications for the design of the state-transition tables of level*1*'s sub-systems and also the incorporation of diagnostic models in their monitoring models. Accordingly, these models differ from those of the higher-level components, which consist of state-transition tables only.

Due to the variety of the monitoring models and also the hierarchical deployment, monitoring agents appear in three different groups, as follows:

- Ss_MAGs that are deployed over the sub-systems of Level*1*. The monitoring model of each of these agents includes a state-transition table and a number of diagnostic models. Agents of this level have parent agents at the immediate upper level and have no child agents. To track the behaviour of their monitored sub-systems, they monitor the occurrences of the normal, corrective and failure events initiated by level*0* and collaborate with their parents by exchanging their state transitions. They also exchange messages with each other as siblings to share their sensory measurements whenever needed.
- Ss_MAGs that are deployed over levels extending from Level*2* to level*n-1*. The monitoring models of each of these agents include a state-transition table only and each of them has a parent agent and child agents. To track the behaviour of the monitored sub-systems, each agent collaborates with its children and parent by exchanging their state transitions.
- S_MAG that is deployed over level*n*. The monitoring model of this agent consists also of a state-transition table. S_MAG has no parent and to track system behaviour, it collaborates with its children by exchanging their state transitions.

4.2.2 Collaboration Protocols[1]

Figure 4-2 shows the collaboration protocol of the monitoring agents across the hierarchal levels. By implementing this protocol, agents can track the behaviour of the monitored sub-systems and the entire system.

Every new state that results from a state transition at level*1* is communicated by the agent to its parent agent, which in turn communicates its own new state higher up to its parent, and so on successively to the S_MAG at the top level (level*n*). The S_MAG, in turn, communicates its own new state to the children at level*n-1*. Every child agent communicates its own new state

[1] Protocols are illustrated according to the Prometheus methodology for developing intelligent agent systems (Padgham and Winikoff, 2004).

similarly to its children. This scenario is repeated successively between every agent and its children until the agents of level*1* are reached.

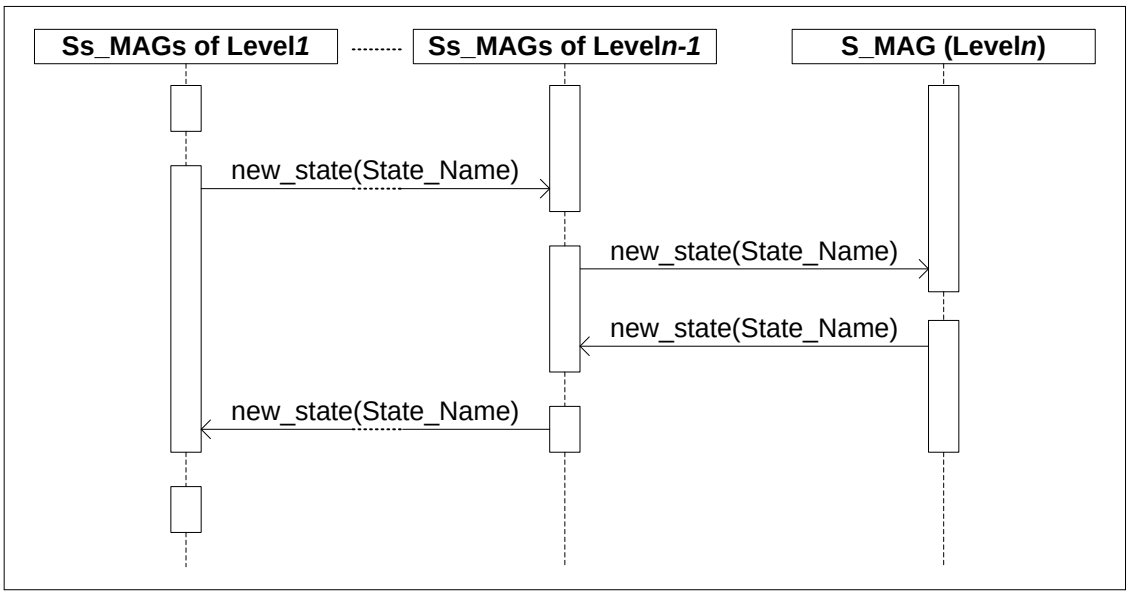


Figure 4-2: The Collaboration Protocol among the Monitoring Agents across the Hierarchical Levels.

Figure 4-3 shows the collaboration protocol by which the Ss_MAGs of level*1* share their sensory measurements, here termed *global measurements*. Any Ss_MAG may ask for a global measurement by sending a request (an ask message) to the intended Ss_MAG of the same level; every global measurement is annotated with the name of the intended Ss_MAG. The receiving Ss_MAG (asked Ss_MAG) should answer accordingly by sending a tell message.

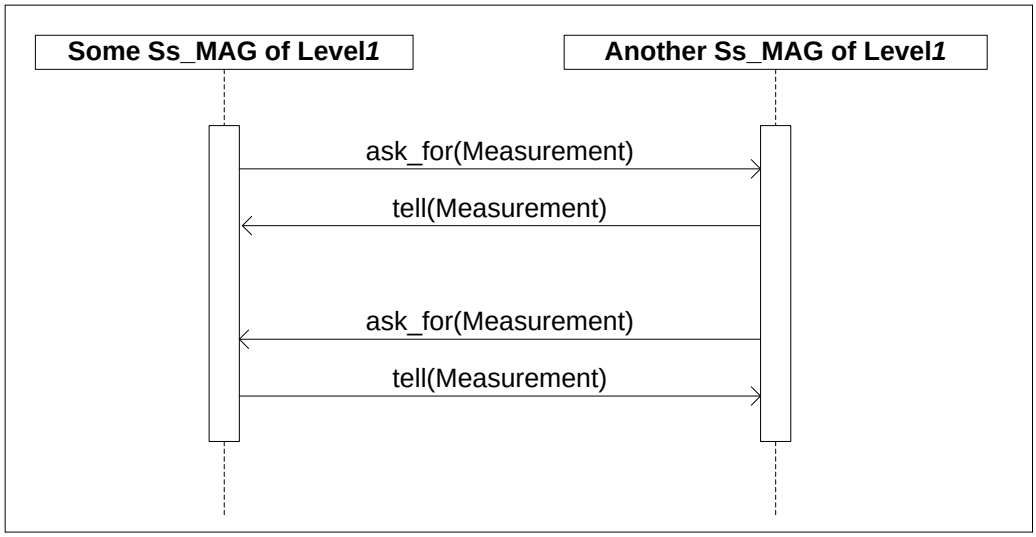


Figure 4-3: The Collaboration Protocol among MAGs of level*1*.

4.3 Belief, Desire and Intention of Ss_MAGs of Level*1*

Figure 4-4 shows a generic illustration of Ss_MAGs of level*1*. Belief bases of these Ss_MAGs are updated by (a) perceiving the monitoring model and monitoring expressions which are evaluated based on up-to-date sensory measurements; (b) receiving messages from their parents about higher level state transitions and from their siblings either asking or telling (the given Ss_MAG) about global measurements.

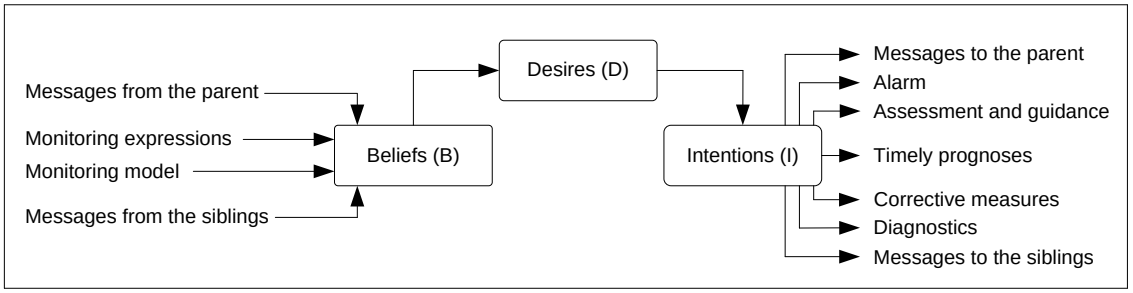


Figure 4-4: A Generic Illustration of Ss_MAG of Level*1*.

According to the up-to-date beliefs, the Ss_MAG reasons and produces the following actions:

- Send messages to inform the parent about the new states.
- Provide the operators with alarm, assessment and guidance and timely prognosis.
- Corrective measures, which may be taken by the Ss_MAG either directly by instructing the system's actuators or by encoding them to the system's controller.
- Diagnose the underlying causes of the detected failures.
- Send messages to ask or tell the siblings about global measurements.

4.3.1 Updating the Belief Base of the Ss_MAGs of Level*1*

The flowchart in Figure 4-5 illustrates the way in which Ss_MAGs of level*1* update their belief bases. Once the monitoring process is launched, the Ss_MAG perceives its state-transition table, and regards the initial state as the current state. From this state, the agent provides the operators with assessment and guidance and identifies the active events to launch a monitoring cycle. If the current state has no event that triggers a transition to another state (AE = nil), i.e. the state is a sink or permanent failure state, then the Ss_MAG does not launch a monitoring cycle.

On the contrary, if there are trigger events then before launching a monitoring cycle, Ss_MAG checks whether a belief "asked by a sibling" has been added; a sibling has asked for a global measurement. If so, then the Ss_MAG replies to that sibling by sending an up-to-date measurement and drops that belief. Otherwise, the Ss_MAG updates and evaluates the expression of the first active event. If that expression needs a global measurement, then the

belief “need to contact a sibling” is added and an ask message is sent to the intended sibling. According to that belief, the Ss_MAG will wait to receive an answer.

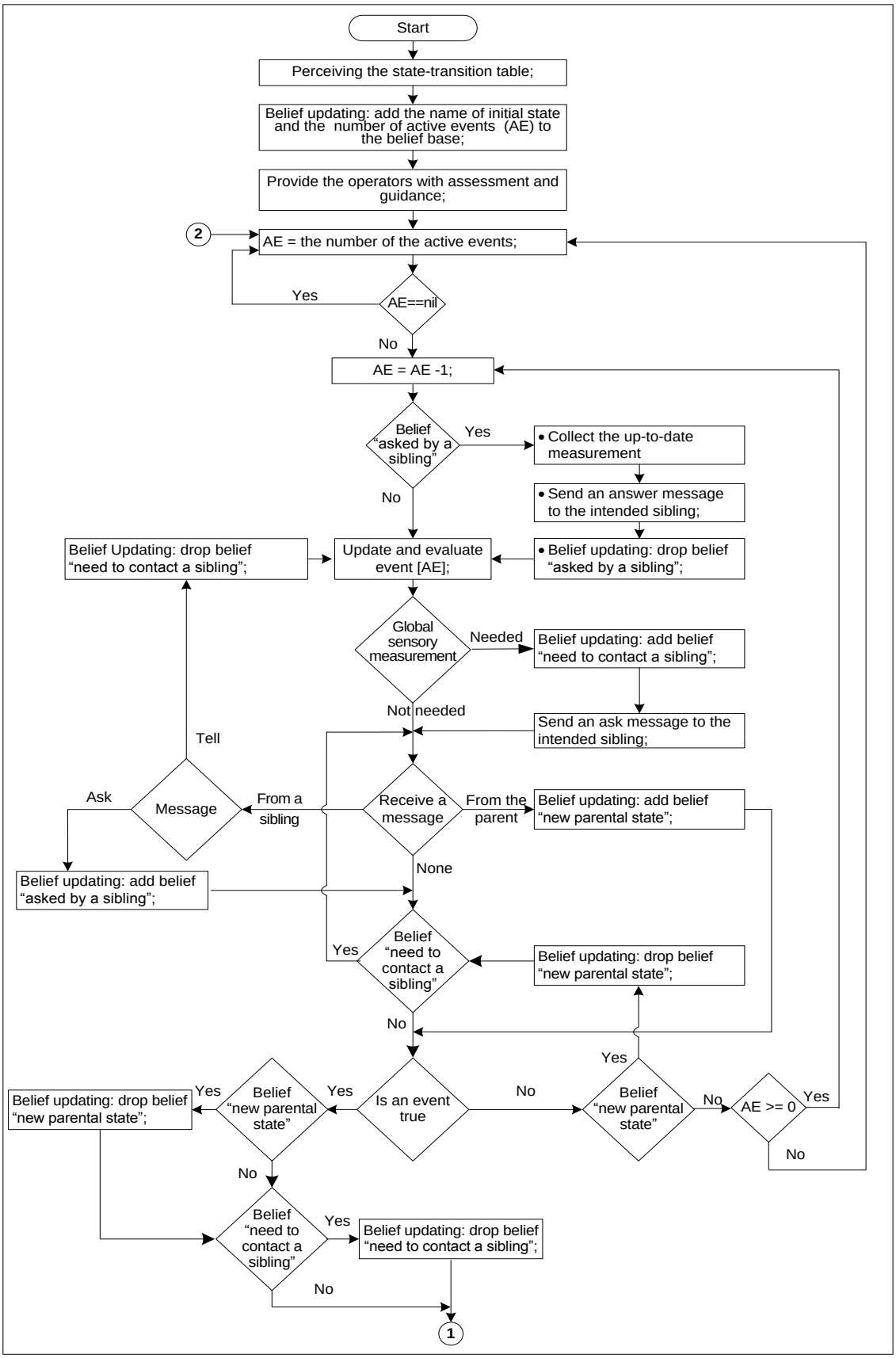


Figure 4-5: Flowchart of Updating the Belief Base of Ss_MAGs of Level*1*.

If the expression does not need a global measurement, then the Ss_MAG checks whether any message has been received. If a message has been received from a sibling then the Ss_MAG checks whether it is an "ask" or "tell" message. In the former case, the belief "asked by a sibling" is added, whereas in the latter case, the Ss_MAG takes the received measurement to instantiate the relevant expression.

If a message has been received from the parent, then the belief base is updated by adding the belief, "new parental state" and the agent checks whether the received state instantiates an expression and results in a true evaluation. If it does not then the belief "new parental state" is dropped and if the Ss_MAG still believes "need to contact a sibling" it will stay waiting. If the evaluation of the expression results in false and there is no "new parental state" belief, then the Ss_MAG checks whether the expression is the last (of the active events) or not. In the former case, a new monitoring cycle is launched, whereas in the latter case the next expression is evaluated.

However, if the received parental state triggers a state transition or an expression is evaluated as true, then the current beliefs are dropped and the state-transition table is perceived further. The flowchart of Figure 4-6 is the sequel to Figure 4-5 and illustrates the updating of the belief base after an expression is evaluated and verified true (the occurrence of an event is verified).

The Ss_MAG perceives the state-transition table to identify the type of event[1] that has occurred and adds that type to the belief base. If it is a normal or a corrective event, then the Ss_MAG drops the belief "normal or corrective event", perceives the state-transition table (Figure 3-13) further and achieves the following state-transition procedure:

- Transit to the corresponding state (new state).
- Update the belief base as follows: add the name of the new state as the current state and the number of active events (AE).
- Send a message to inform the parent about the new state.
- Provide the operator with assessment and guidance of the new conditions.
- Launch a monitoring cycle for the active events of the new state, as shown by the connection between circle 2 in Figure 4-6 and the corresponding circle in Figure 4-5.

On the other hand, if the occurred event is a failure event, then the Ss_MAG perceives the state-transition table and achieves the following procedure:

- Announce an alarm by quoting the corresponding clause from the alarm attribute.
- Perceive the controlling status attribute. If it is instantiated with "after_diagnosing" the belief "controlling is after_diagnosing" is added. When the status is corrective measures, they will

[1] The event's type is identified by perceiving the alarm cell, see section 3.6.2.

be applied accordingly. If that status is instantiated with "impossible", then the Ss_MAG perceives the diagnosis attribute.

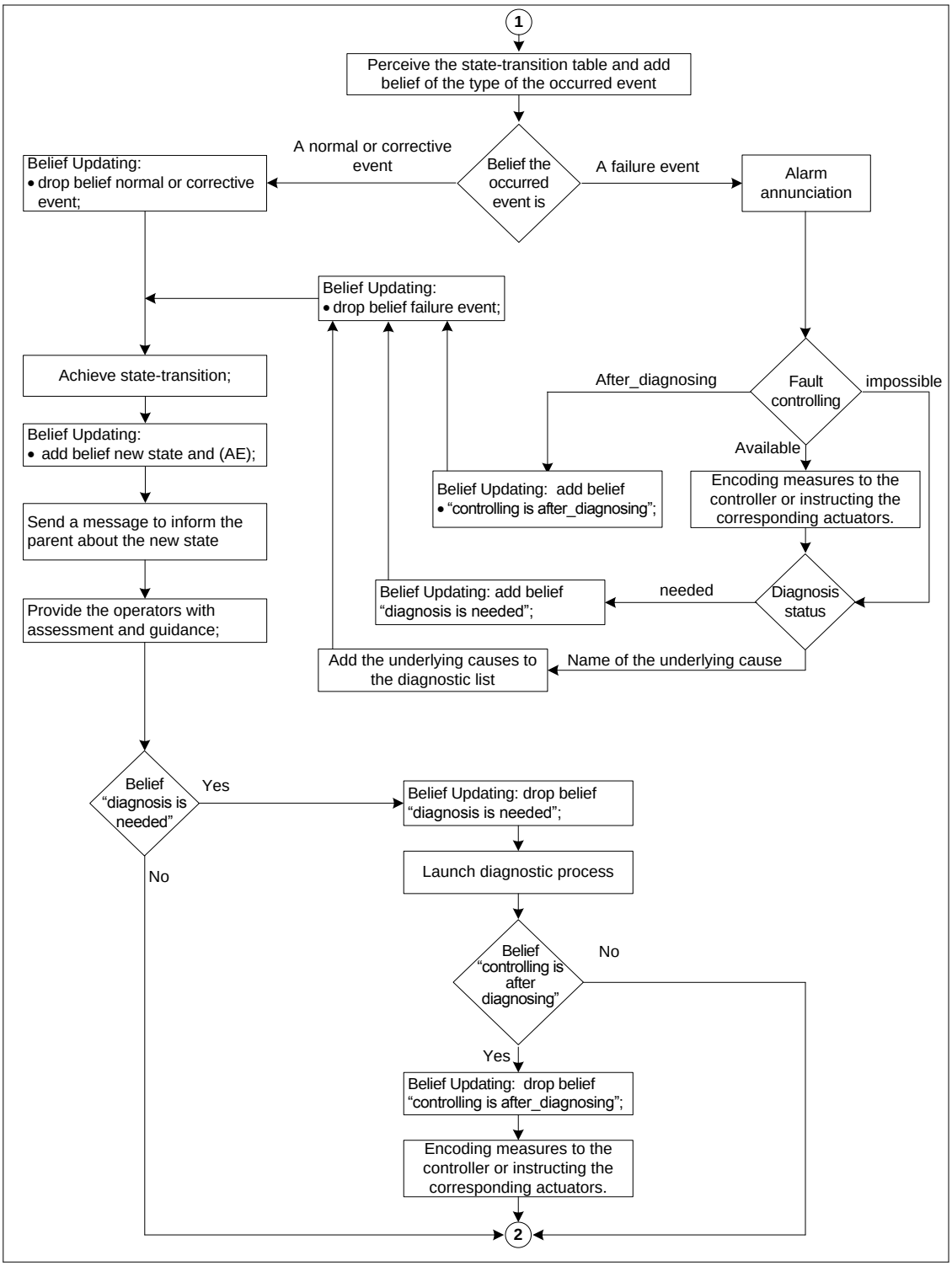


Figure 4-6: Sequel of Updating Belief Base of Ss_MAGs of Level*1* Started in Figure 4-5.

- Perceive the diagnosis attribute. If it is instantiated with "needed" then the belief "diagnosis is needed" is added. On the contrary, if the attribute holds the name of the underlying cause, then the name is added to the diagnostic list.

- Drop belief “failure event” and achieve the above state-transition procedure. Thereafter, if the Ss_MAG believes “diagnosis is needed”, then a diagnostic process is launched. The Ss_MAG may also believe that controlling is “after_diagnosing”; in this case the corrective measures would be found in association with the diagnosed causes and applied. If Ss_MAG does not believe “diagnosis is needed” or controlling is “after diagnosing”, then a reasoning cycle is launched to the current state, as shown by the connection between circle 2 in Figure 4-6 and the corresponding circle in Figure 4-5.

4.3.2 Desires of the Ss_MAGs of level*1*

Every Ss_MAG of level*1* has local and global desires. Local desires can be listed as follows:

- Monitoring the operational conditions of the assigned sub-system. This can be achieved by updating and evaluating monitoring expressions that represent the active events of the current state and tracking the behaviour of the sub-system through executing the occurred events on the state-transition table.
- Alarming the operator on the occurrence of failure events.
- Providing the operators with assessment, guidance and timely prognoses.
- Diagnosing the underlying causes of the occurred failure events.
- Controlling the occurred failure events.

Global desires can be listed as follows:

- Asking siblings about global measurements.
- Telling siblings about global measurements that they have already asked about.
- Informing the parent about new states resulting from state transitions.
- Receiving parental states and checking whether they trigger a state transition in the state-transition table of the monitored sub-system.

4.3.3 Intention of the Ss_MAGs of Level*1*

Figure 4-7 illustrates the deliberation process over the local and global desires of the Ss_MAGs of level*1*. The process is modelled as a state-machine model[1]; each state represents a committed intention and every transition is triggered by an up-to-date belief.

Once the Ss_MAG has a belief “current state” and the number of the active events (AE), its intention would be to provide the operators with assessment and guidance and to achieve

[1] Unfortunately there is no particular notation to model the way in which a BDI agent deliberates among its desires and commits to an intention. Therefore, the state-machine is used here to model the deliberation process.

monitoring cycles (update and evaluate expressions), if there are any active events. While achieving this intention, if the Ss_MAG has the belief, "asked by a sibling", it accordingly suspends the current monitoring cycle and commits to an intention of answering that request by sending the up-to-date global measurement. Thereafter, the Ss_MAG updates its belief base by dropping the belief, "asked by a sibling" and resumes the suspended monitoring cycle.

Similarly, if the Ss_MAG believes there is a "need to contact a sibling" then the current monitoring cycle is suspended and the Ss_MAG commits to a new intention and sends an ask message to the intended sibling and keeps waiting until an answer message is received. Once that message is received the belief, "need to contact a sibling" is dropped and the suspended cycle will accordingly be resumed.

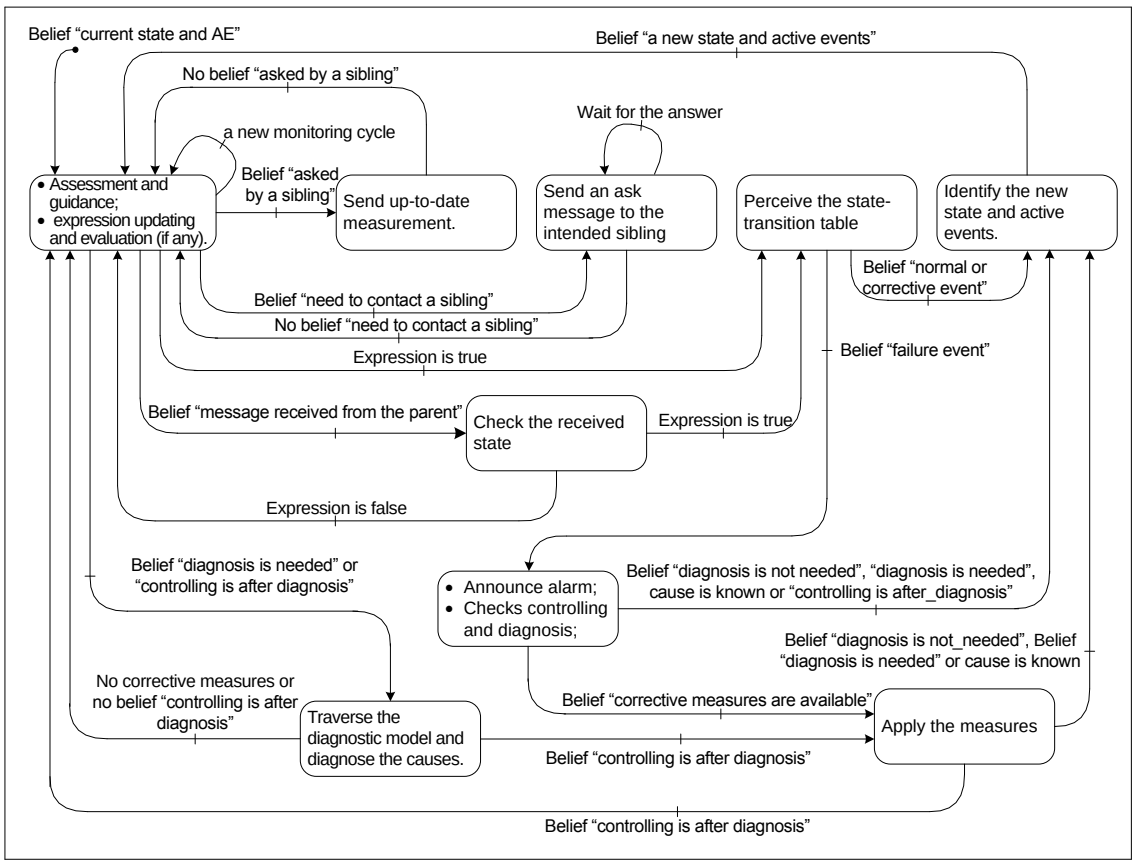


Figure 4-7: The Deliberation Process of the Ss_MAGs of Level*1*.

Whenever an Ss_MAG receives a message from the parents, the monitoring cycle is suspended and it checks whether the conveyed state triggers a state transition. If it does not, then the suspended cycle is resumed. On the contrary, if it does or any expression is verified true (the event has occurred) then the Ss_MAG perceives the state-transition table further.

If the Ss_MAG believes that the occurred event is a normal or corrective event, then the new state and the number of the active events are identified and a monitoring cycle is launched.

However, if the belief is a failure event, then an alarm is announced and the diagnosis status and availability of corrective measures are perceived. Thereafter, Ss_MAG commits to an intention of identifying the new state along with its active events.

After achieving this intention, the Ss_MAG will believe "a new state and active events", upon which the operators will be provided with assessment and guidance. Before launching a monitoring cycle for the new state, the Ss_MAG checks its belief base; if there is presence of the belief, "diagnosis is needed" or control is "after_diagnosis", then a diagnostic process is launched. After diagnosing the underlying causes, if there is a belief that control is "after_diagnosis" then corrective measures are applied and a monitoring cycle of the new state is launched. If such a belief is not present then the cycle is launched immediately.

However, if corrective measures are available in the state-transition table, the Ss_MAG accordingly applies them and whether it believes "diagnosis is not needed", "diagnosis is needed" or the cause is known, the new state along with its active events are identified.

4.4 Belief, Desire and Intention of Ss_MAGs of Levels Extending from Level*2* to Level*n-1*

Figure 4-8 shows a generic illustration of an Ss_MAG of the levels extending from level*2* to level*n-1*.

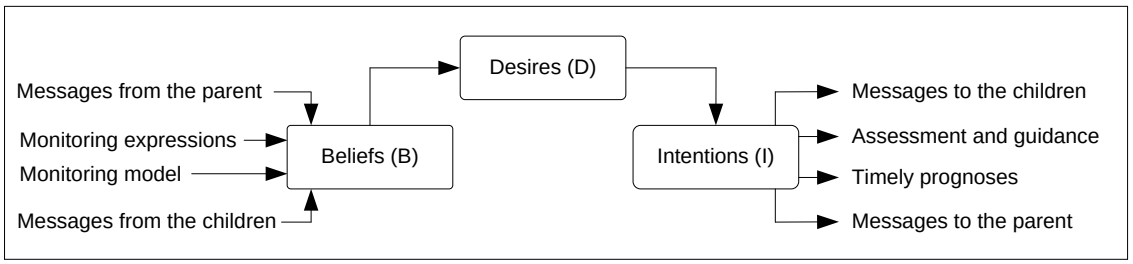


Figure 4-8: A Generic Illustration of Ss_MAGs of Levels Extending from Level*2* to Level*n-1*.

The illustration shows the resources for updating the belief base and the produced actions. Belief resources can be listed as follows:

- Messages received from the parent.
- Perceiving the monitoring model (which consists of a state-transition table) and the monitoring expression, which are evaluated based on the new states of the parent and children.
- Messages received from the children.

According to the up-to-date belief, an Ss_MAG produces the following actions:

- Send messages to inform the children about state transitions.
- Provide the operator with the corresponding multi-level assessment, guidance and timely prognosis.
- Send messages to inform the parent about state transitions.

4.4.1 Updating the Belief Base of Ss_MAGs of Level*2* to Level*n-1*

The flow chart of Figure 4-9 illustrates the way in which the Ss_MAGs of levels extending from level*2* to level*n-1* update their belief bases. Once the monitoring process is launched, the Ss_MAG perceives its state-transition table, identifies and regards the initial state as the current state, identifies the number of active events and updates its belief base accordingly. The initial state is identified through its typical appearance as the first state in the state-transition table.

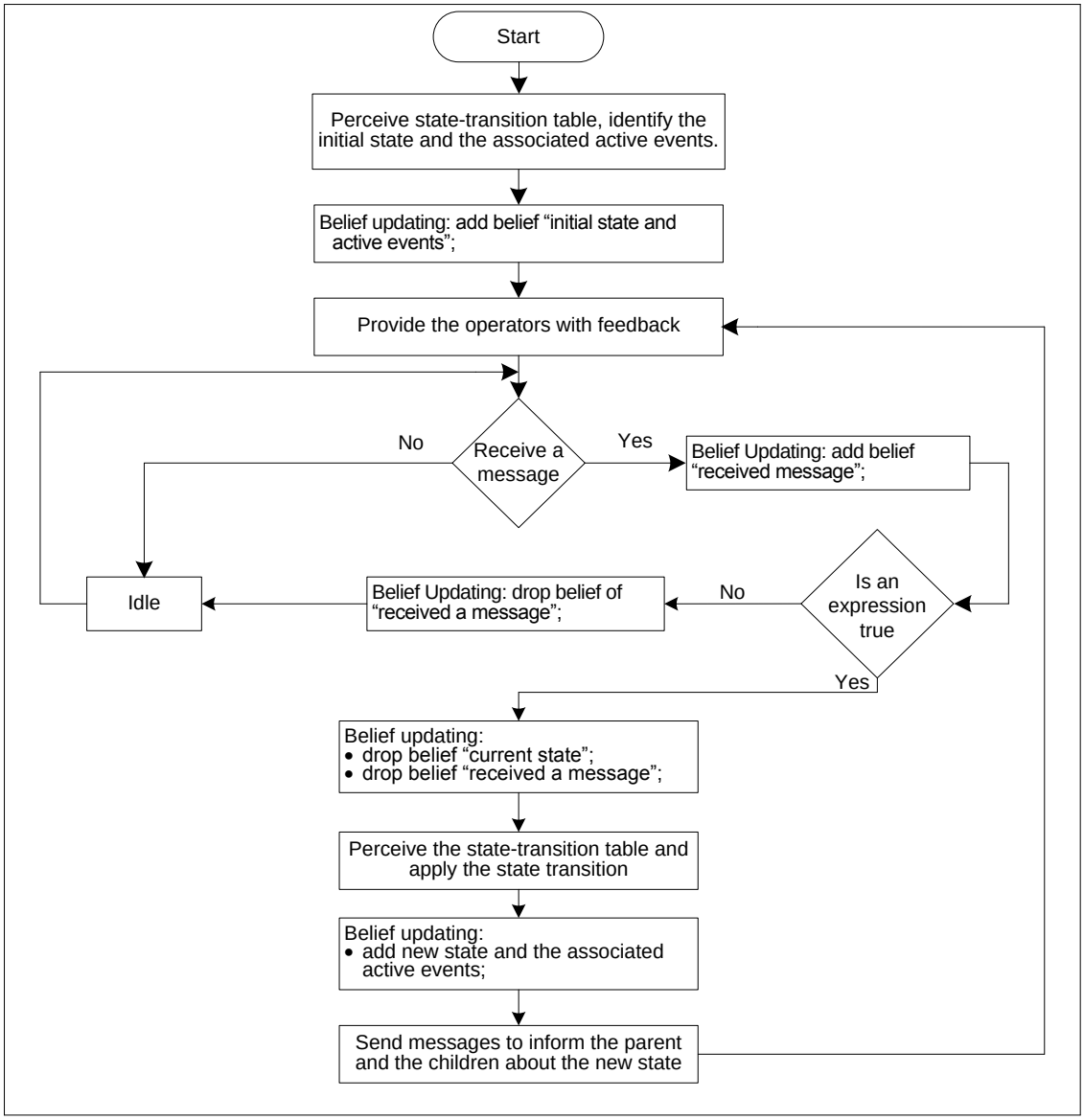


Figure 4-9: Flowchart of the Updating the Belief Base of Ss_MAGs of Level*2* to Level*n-1*.

After providing the operators with assessment and guidance, the Ss_MAG checks for the existence of any received message. If there is no message, it remains idle. If, on the contrary, the Ss_MAG receives a message from the parent or a child, then it checks whether that message conveys a state that instantiates an expression and triggers a state transition. If not, then the belief "received message" is dropped and the idle status is resumed.

However, if the received state instantiates and verifies an expression, then the Ss_MAG updates its belief base by dropping the beliefs about the current state and "received message" and perceives the state-transition table to add the name of the new state along with its associated active events. Messages conveying the new state will also be sent to inform the parent and children.

4.4.2 Desires of the Ss_MAGs of Level*2* to Level*n-1*

Similar to Ss_MAGs of level*1*, these Ss_MAGs also have local and global desires. Local desires can be listed as follows:

- Monitoring the conditions of the assigned sub-systems.
- Providing the system operators with multi-level assessment and guidance, and timely prognosis.

Global desires can be described as informing the parent and the children about the local state transitions.

4.4.3 Intentions of the Ss_MAGs of Level*2* to Level*n-1*

According to up-to-date beliefs, Ss_MAGs of levels extending from level*2* to level*n-1* deliberate on their desires to commit an intention as shown in Figure 4-10.

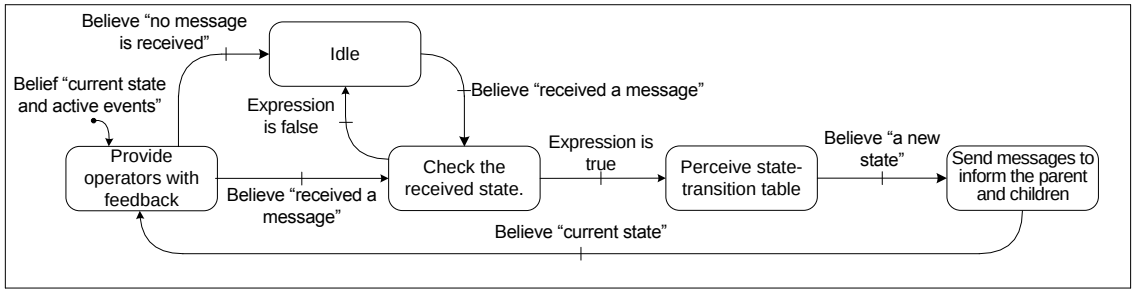


Figure 4-10: The Deliberation of Ss_MAGs of the Levels Extending from Level*2* to Level*n-1*.

Once the Ss_MAG has a belief, "current state and active events", it provides the operators with assessment and guidance. If an Ss_MAG does not receive a message from the parent or a child

then it goes idle. On the contrary, whenever an Ss_MAG receives a message, it adds a new belief, "received a message", and commits to an intention of perceiving whether that message conveys a state that triggers a state transition. If it does not, then the idle status is resumed. However, if it does, then the Ss_MAG achieves the following procedure:

- Perceive the state-transition table to identify and regard the new state as the current state along with identifying the associated active events.
- Inform the parent and children about the new state.
- Provide the operators with the corresponding assessment and guidance.
- Resume its idle status as long as there is no received message.

4.5 Belief, Desire and Intention of S_MAG of Level*n*

Figure 4-11 shows an illustration of the S_MAG of the top hierarchical level (level*n)*. The illustration shows the resources for updating the belief base and the produced actions. Belief updating resources are messages received from children, perceptions of the monitoring model (state-transition table) and monitoring expressions which are evaluated based on the new state of the children.

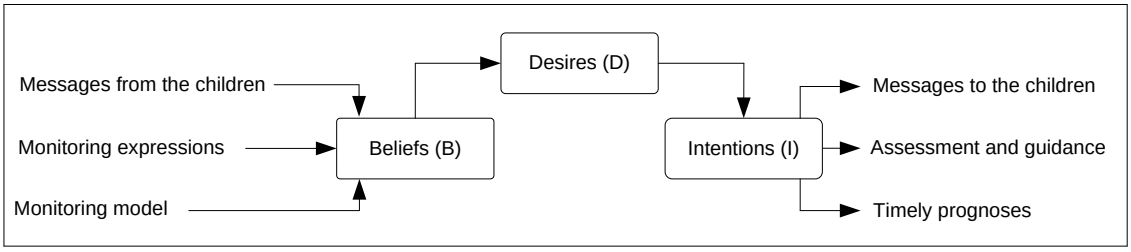


Figure 4-11: A Generic Illustration of S_MAG of Level*n*.

According to the up-to-date belief, the S_MAG reasons and produces the following actions:

- Send messages to the children.
- Provide the operator with assessment and guidance, and timely prognosis.

The desires of the S_MAG are identical to those of the Ss_MAG of levels extending from level*2* to level*n-1*. The only difference is that the S_MAG has no parent and thus it exchanges messages only with the children of the immediate lower level (level*n-1*). To avoid repetition, discussion of the belief updating, desires and intention of the S_MAG is intentionally omitted, as they are as same as those of Ss_MAGs of levels extending from level*2* to level*n-1*. It should not be forgotten, however, that in the case of the S_MAG, messages are exchanged only with the children as there is no parent.

4.6 Programming and Implementing the BDI Agents

Writing a high-level language program requires an environment in which the program's functions are compiled, exchange parameters and deliver the intended results. Similarly, BDI-agents require a platform[1] in which (a) real-time multi-agent systems are constructed; (b) agents take their perceptions and exchanged messages into a reasoning process and deliver the intended tasks.

One of the earliest implementations of a BDI-agent platform is the Procedural Reasoning System (PRS) by Georgeff and Lansky (1987). In the PRS, in order for the agent to proceed from the three mental attitudes (Belief, Desire, and Intention) to producing actions, it is programmed as a belief base and a plan library. Since its emergence, PRS has been intensively researched, improved and thus it has become the most durable (Bordini *et al.*, 2007), best-established (D'Inverno *et al.*, 1997) and best-known (Bellifemine *et al.*, 2007) agent platform.

PRS has, moreover, demonstrated an outstanding ability in achieving integral reasoning on a spectrum of the distributed applications, ranging from fault diagnosing and controlling of critical systems and air traffic management systems to managing complicated business processes, such as production task scheduling and product customisation (D'Inverno *et al*, 2004; Huerta *et al.*, 2007; Giret *et al.*, 2005).

One of the successor implementations of the PRS is the AgentSpeak platform[2] (Rao, 1996). AgentSpeak can be characterised as a language that provides a sound theoretical underpinning (formal semantic) for programming BDI-agents. Thus, its applicability in different contexts could be easily investigated. The formal semantic of AgentSpeak platform has recently been extended and implemented by the Jason platform (Bordini *et al.*, 2007). In the context of this thesis, Jason is exploited as a programming language to implement the monitoring agents of the hierarchical multi-agent system. It must also be pointed out that, in addition to the Jason programming language, agents also need to be interfaced with the monitoring model, data structures, operator interface and system controller by means of a procedural programming language[3].

In order to implement the required collaboration among agents, Jason offers three different implementation and communication infrastructures (Bordini *et al.*, 2007). The first is a

[1] In addition to a platform, the environment might also be called the framework, agent architecture or infrastructure.

[2] There is, however, a considerable number of agent platforms that have been developed and implemented. Some of them no longer exist, others have proved an outstanding success in specific fields and a few are open source for general purpose application. Therefore, it would be advisable to refer to agent platform surveys by Bordini *et al.* (2006), Fisher *et al.* (2004) and Mascardi *et al.* (2004).

[3] For further information about the reason behind using procedural language, it would be advisable to refer to Chapter Five of Bordini *et al.* (2007).

centralised infrastructure, by which agents can be installed, communicate and operate on the same host. The second is a distributed infrastructure called Simple Agent Communication Infrastructure (SACI) by which Jason agents can be distributed over a number of hosts. The first and second infrastructures exploit Knowledge Query and Manipulation Language (KQML) (Finin *et al.*, 1993) to implement informative communication among the agents.

The third infrastructure consists of Java Agent DEvelopment framework (JADE) and Intelligent Physical Agents-Agent Communication Language FIPA-ACL (FIPA, 2003). It supports the real-time monitoring and inspecting of agents and also achieves the required interoperability among agents.

4.7 Programming the Monitoring Agents

Before discussing the programming of the plan libraries of the monitoring agents, it might be helpful to illustrate the basic programming notions of the Jason programming language.

4.7.1 Jason Plan

As shown in Figure 4-12, a Jason plan consists of two main parts; the plan head and plan body. The plan head consists further of two other parts, trigger event and context and the plan body consists of a course of actions.

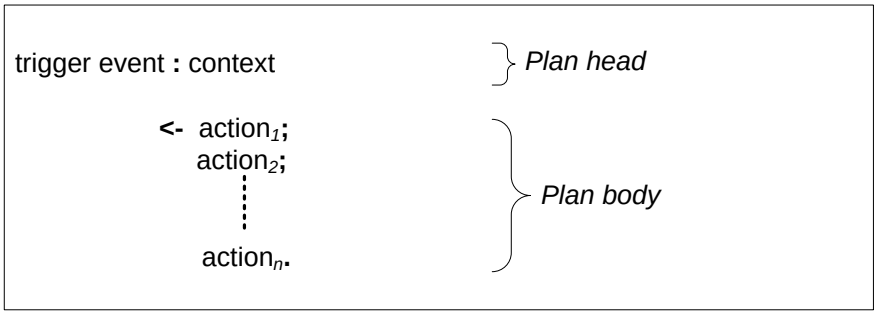


Figure 4-12: Structure and Components of Jason Plan.

Plan Body[1]

The course of actions of the plan body always starts with "<-", every action is followed by a semicolon and the last action is followed by a dot. Table 4-1 abstracts the six types of actions that may appear in the plan body. The first type is external actions, which are directly executed in the environment of the agent. Monitoring agents implement this action as perceiving the

[1] The explanation has been deliberately started with the second part of the plan (plan body) as the course of actions in this part represents a key issue in explaining the first part (plan head).

monitoring model, updating and evaluating the monitoring expressions and instructing the controller (or actuators) to apply corrective measures.

Table 4-1: Plan's Body Actions.

Action	Form
External action	action
Achievement goal	!goal
Test goal	?goal
Internal action	.action
Belief base update action	+belief, –belief or –+ belief
Computational action	X=Y+Z

The second type is the achievement goal. These actions are always preceded by an exclamation mark "!" and require one or more plan to be achieved. An example of an achievement goal will be presented in section 4.7.2; one can be seen in line 2 of plan L_1-P_1. As that goal matches the head of plan L_1-P_2, it would thus be the means-ends plan to achieve that goal.

The third type of action is the test goal. These actions are always preceded by a question mark "?" and used to retrieve information from the belief base or the agent environment. Examples of test goals will also be presented in section 4.7.2; one can be seen in line 2 of L_1-P_2 by which information that is already in the belief base is retrieved and exploited in line 3 of the same plan.

The fourth type is internal actions. This type of action is used to achieve internal actions (inside the agent itself or the multi-agent system), e.g. sending a message to another agent. Such actions are always preceded by a dot. The fifth type of action is the belief updating action; this is used to update the belief base of the agents. Updating might be achieved by:

- Adding a belief to the belief base of the agent. In such a case the belief is preceded by an addition sign "+".
- Deleting a belief from the belief base of the agent. In such a case the belief is preceded by a subtraction sign "-".
- Deleting and adding belief. In such a case the belief is preceded by subtraction and addition signs "-+".

Finally, the sixth type of action is the computational action by which an agent computes and finds the result of different mathematical processes.

Plan Head

As shown by Figure 4-12, the trigger event is the first part of the plan head which, in turn, is separated by a colon "**:**" from the second part, the context. The trigger event represents the post-condition upon which the plan is recognised as relevant to the current belief or an achievement goal from other plans in the plan library. The context represents the pre-conditions depending on which applicable plans are selected from the relevant plans. Plan context is not mandatory and its appearance depends on the need of post-conditions.

During the compilation process, plans whose trigger events match the up-to-date belief or an achievement goal are regarded as relevant plans. The contexts of those plans are evaluated in a further compilation step to arrive at one or more applicable plans. In the case of a single plan, that plan is considered as a means-ends plan and executed, while if a set of plans is identified a further compilation step is undertaken to select a means-ends plan. In that step the highest priority plan among the applicable plans is considered as means-ends. Priority is identified according to the order in which the plans appear in the plan library.

As shown earlier in this chapter, there are three different groups of monitoring agents. Consequently, the plan libraries of those agents also differ from each other. The next sections, therefore, list and explain the plan libraries of the three types of monitoring agents: (a) Ss_MAGs that are deployed over level*1*; (b) Ss_MAGs that are deployed over levels extending from level*2* to level*n-1*; (c) S_MAG that is deployed over level*n*.

In order to make a clear distinction among those libraries, an abbreviation that is composed of the deployment level and plan number is used to designate every plan. Those abbreviations will appear in the following generic form:

$$L_{NO}\text{-}P_{NO}$$

Where L_{NO}: is the serial number of the level at which the agent is deployed.
P_{NO}: is the serial number of the plan according to its appearance in the plan library of an agent.

Moreover, each line in the plans is given a number to reference the explanations.

4.7.2 Plan Library of Ss_MAGs of Level*1*

Ss_MAGs of level*1* launch their safety monitoring processes with the following initial beliefs:

```
1.  parent_name(PName).
2.  diagnosis_status(false).
3.  subsystem(SsName).
4.  start(monitoring).
```

By the above lines the Ss_MAG will have the respective beliefs: the name of the parent, diagnosis is not needed so far, name of the monitored sub-system and launching a monitoring process. For the belief of line 4, plan L_1- P_1 is the relevant, applicable and means-ends plan.

Plan L_1-P_1: as the head of this plan can be unified with the initial belief of line 4 of the above, it will be the relevant plan. Moreover, as it does not have a context and it is the only relevant plan, it is selected as an applicable and means-ends plan. The role of this plan is to launch a monitoring cycle for the initial state, which at this time is the current state.

```
1.  +start(monitoring)
2.        <- !monitoring(0).
```

By the achievement goal of line 2, Ss_MAG launches a monitoring cycle, where the position of the structure of the initial state is held in position number 0 of the states array (as declared in Figure 3-13).

Plan L_1-P_2: this plan is the only one relevant to the achievement goal of line 2 of L_1-P_1. Its role is to provide the operators with assessment and guidance, identify the active events of the current state (CState), and set up contact statuses.

```
1.  +!monitoring(CState)
2.        <- ?subsystem(SsName)
3.           feedback(CState, SsName);
4.           events(CState);
5.           ?active_events(AE);
6.           +pmessage(false);
7.           +smessage(false);
8.           +global_measurement(false);
9.           ?diagnosis_status(Need);
10.          !diagnose(Need);
11.          !observe(AE, false, false, false).
```

The test goal of line 2 retrieves the name of the monitored sub-system that is added by line 3 of the initial beliefs. The external action of line 3 provides the operators with assessment and guidance of the current conditions. According to the structure that holds the state-transition table of level*1* (Figure 3-13), this action is implemented by the following pseudo code:

```
if (action == "feedback")
  {
    Int StatePos = action.GetTerm (0);          //Get the position of the current state from the
                                                //action.
    String SubSysN = action.GetTerm (1);        //Get the name of the monitored sub-system
                                                //from the action.
```

```
   AnnounceAss(States[StatePos].Assessment, SubSysN);      //From the current state
                                                           //announce assessment for the
                                                           //monitored sub-system.
   AnnounceGuid(States[StatePos].Guidance, SubSysN);       //From the current state
 }                                                         //announce guidance for the
                                                           //monitored sub-system.
```

The external action of line 4 perceives the state-transition table to identify and add the number of the active events of the current state to the belief base. This action is implemented by the following pseudo code:

```
if (action == "events")
  {
    StatePos = action.GetTerm (0);        //Get the position of the current state from the action.
    LE = States[StatePos].LastEvent;                  //Retrieve the position of the last event
                                                      //of the current state.
    FE = States[StatePos].FirstEvent;                 //Retrieve the position of the first event
                                                      //of the current state.
    if (LE != FE)  AE = LE – FE;                  //If the state has more than one event, then
                                                  //calculate the number of those events.
    if (LE == nil || FE == nil) AE = nil;         //If the state has no exit event; while interpreting and
                                                  //loading the table the FirstEvent  and LastEvent of
                                                  //such states are instantiated with nil.
    else   AE = 1;          //The otherwise means the state has only one exit event and the first
                            //and last positions have the same value.
   AddPercept (active_events(AE ));       //Add the percept of the number of the active event to
  }                                       //the belief base.
```

The test goal of line 5 retrieves the number of the active events added by the above pseudo code. Lines 6, 7 and 8 set contact statuses by adding the respective beliefs (a) no message received from the parent *+pmessage(false)*, this is instantiated with true when a message is received from the parent, as shown in line 2 of L_1-P_7; (b) no message received from a sibling *+smessage(false)*, this is instantiated with true when a message is received from a sibling, as shown in line 2 of L_1-P_8; (c) there is a no need to ask a sibling for a global measurement *+global_measurement(false)*, this is instantiated with true when the evaluation of an expression requires such a measurement, as shown in line 3 of L_1-P_{14}.

By the test goal of line 9 and the achievement goal of line 10, the diagnostic status is retrieved and the diagnostic process is triggered (if needed), respectively. Initially, the variable *Need* is instantiated with false, as shown by line 2 of the initial belief. If the diagnosis is needed, then that variable is instantiated with true as shown by line 2 of L_1-P_{24}. For the achievement goal of line 10, L_1-P_{26} and L_1-P_{27} are relevant. By the achievement goal of line 11 a monitoring cycle is launched; for this goal there are several relevant plans: L_1-P_3, L_1-P_4, L_1-P_5 and L_1-P_6.

Plan L_1-P_3: This plan and L_1-P_4, L_1-P_5 and L_1-P_6 are relevant to the achievement goal of line 11 of the above plan. According to its context, if the number of the active events (AE) is more than or equal to zero, no message is received from the parent or the siblings and there is no need for asking a sibling for a global measurement (GM), then this plan will be one of the applicable plans. If, moreover, the contexts of the L_1-P_4, L_1-P_5 and L_1-P_6 are verified false, then this plan is the means-ends one. The role of this plan is to evaluate the expressions of the active events.

```
1.+!observe(AE, PMSG, SMSG, GM): AE >= 0 & PMSG == false & SMSG == false &GM==false
2.        <- !examine_event(AE);
3.           ?pmessage(PMSG);
4.           ?smessage(SMSG);
5.           ?global_measurement(GM);
6.           !observe(AE, PMSG, SMSG, GM).
```

Before evaluating an expression, its sensory measurements are updated and the need for a global measurement is checked. That is achieved by the achievement goal of line 2, for which L_1-P_{13} is the relevant, applicable and means-ends plan. While achieving this goal the contact statuses might be updated, the test goals of lines 3, 4 and 5 retrieve the up-to-date statuses. For the achievement goal of line 7 either this plan is the recursive means-ends plan or any of L_1- P_4, L_1- P_5 or L_1- P_6 is the means-ends plan.

Plan L_1-P_4: This plan and plans L_1-P_3, L_1-P_5 and L_1-P_6 are relevant to the achievement goals of line 11 of L_1-P_2 and line 6 of L_1-P_3. The context of this plan is verified true when (a) the number of active events (AE) is less than zero, implying achieving a monitoring cycle or the number of those events is nil, implying the current state has no exit event, e.g. in the case of a failure state (FS); (b) no message has been received from the parent or a sibling; (c) there is no need for a global measurement. If the context is true and the contexts of L_1-P_3, L_1-P_5 and L_1-P_6 are false, then this plan will be the applicable and the means-ends one.

```
1. +!observe(AE, PMSG, SMSG, GM): AE < 0 & PMSG==false & SMSG==false & GM==false
2.        <- ?active_events(AE);
3.           ?pmessage(PMSG);
4.           ?smessage(SMSG);
5.           ?global_measurement(GM);
6.            !observe(AE, PMSG, SMSG, GM).
```

The role of this plan is to launch a new monitoring cycle after evaluating the expressions of all the active events, when the current state has only one or a number of events. On the contrary, when the current state has no exit event, this plan keeps the agent idle.

To launch a new monitoring cycle the test goal of line 2 retrieves the number of active events (AE) and the test goals of lines 3, 4 and 5 retrieve the up-to-date contact status. For the achievement goal of line 6, this plan and L_1-P_3, L_1-P_4, L_1-P_5 and L_1-P_6 are relevant.

Plan L_1-P_5: This plan and L_1-P_3, L_1-P_4 and L_1-P_6 are relevant to the achievement goals of line 11 of plan L_1-P_2, line 6 of plan L_1-P_3, and line 6 of plan L_1-P_4. Its context is verified true when the given Ss_MAG receives a message from either the parent or a sibling. If the context is true and the contexts of plans L_1-P_3, L_1-P_4 and L_1-P_6 are false, then this plan would be the applicable and the means-ends one.

```
1.  +!observe(AE, PMSG, SMSG, GM): PMSG == true | SMSG == true
2.      <- +current_event (AE)
3.         ?pmessage(Parent);
4.         ?smessage(Sibling);
5.         !sender(Parent, Sibling).
```

The role of this plan is to check whether there is a message received during a monitoring cycle. Line three adds the serial number of the current active event to the belief base. By the test goals of lines 3 and 4, Ss_MAG retrieves the contact statuses, which are updateable by line 2 of L_1-P_7 and line 2 of L_1-P_8. By the achievement goal of line 5, Ss_MAG checks whether the sender is the parent or a sibling. For this goal, plans L_1-P_9 and L_1-P_{10} are relevant.

Plan L_1-P_6: This plan and L_1-P_3, L_1-P_4 and L_1-P_5 are relevant to the achievement goals of line 11 of plan L_1-P_2, line 6 of plan L_1-P_3 and line 6 of plan L_1-P_4. Its context is verified true when GM is instantiated with a true truth value, in other words, when the given Ss_MAG has already asked a sibling for a global measurement. If the context is true then it will be one of the applicable plans. If, moreover, the contexts of L_1-P_3, L_1-P_4 and L_1-P_5 are false, then this plan would be the means-ends one.

```
1.  +!observe(AE, PMSG, SMSG, GM): GM == true
2.      <- ?pmessage(Parent);
3.         ?smessage(Sibling);
4.         ?global_measurement(GM);
5.         !observe(AE, Parent, Sibling, GM).
```

Lines 2, 3 and 4 retrieve the up-to-date contact statuses. For the achievement goal of line 5, L_1-P_3, L_1-P_4, L_1-P_5 and L_1-P_6 are the relevant plans. As long as GM is true this plan will recursively be the applicable and means-ends plan and thus the Ss_MAG stays waiting until it receives a tell message.

GM value is instantiated with true by line 3 of L_1-P_{14}. When the Ss_MAG receives the needed measurement, GM is updated and instantiated with false by line 5 of L_1- P_{11}. In that case, this plan would not be the means-ends one.

Plan L_1-P_7: This plan will be the means-ends one when the given Ss_MAG receives a message[1] from its parent. Its role is to update the relevant contact status upon receiving a message from the parent.

```
1.  +!message(PState)
2.        <- -+pmessage(true);
3.           +parental_state(PState).
```

Line 2 updates the belief base of the Ss_MAG by dropping the belief *pmessage(false)* and adding the belief, *pmessage(true)*. This belief would be instantiated with the variable *PMSG* of line 3 of L_1-P_3 and line 3 of L_1-P_4, and with the variable *Parent* of line 3 of L_1-P_5 and line 2 of L_1-P_6. Line 3 adds the received parental state to the belief base. This belief is retrieved and used accordingly in line 2 and line 4 of plan L_1-P_9.

Plan L_1-P_8: This plan will be the means-ends one when the given Ss_MAG receives a message[2] from a sibling. Its role is to find out the name and the purpose of the sender to be answered accordingly and to update the relevant contact status.

```
1.  +!sibling_message(Measurement, Subject)[source(SName)]
2.        <- -+smessage(true);
3.           +sibling(Measurement, Subject, SName).
```

The measurement and the subject are obtained from the respective variables *Measurement* and *Subject* of the received message. According to whether the subject is "ask" or "tell", the *Measurement* could be either sensor identification or a measurement value. The name of the agent is obtained from *SName* of line 1. The obtained measurement, purpose and sender name are added to the belief base by line 3. Line 2 updates the belief base by dropping the belief, *smessage(false)* and adding the belief, *smessage(true)*. This belief would be instantiated with the variable *SMSG* of line 4 of L_1-P_3 and line 4 of L_1-P_4, and with the variable *Sibling* of line 4 of L_1-P_5 and line 3 of L_1-P_6.

[1] A message can be sent by the parent as shown by lines 5, 6 or 7 of plan $L_{2,n-1}$-P_5 (section 4.7.3).
[2] Such a message can be sent by a sibling as shown by line 2 of plan L_1-P_{14}.

Plan L_1-P_9: This plan and L_1-P_{10} are relevant to the achievement goal of line 5 of L_1-P_5. Its context checks whether a message is received from the parent. It will be the applicable and means-ends plan when the context is verified true and the context of L_1-P_{10} is verified false. The role of this plan is to check whether the received parental state results in verifying an occurrence of an event and triggering a state transition.

```
1.  +!sender(Parent, Sibling): Parent == true
2.        <- ?parental_state(PState);
3.           ?current_event(CE);
4.           instantiate(PState, CE);
5.           ?check_event(Occurrence, E);
6.           -+pmessage(false);
7.           !event(Occurrence, E).
```

The test goals of line 2 and line 3 retrieve the received state that has been added to the belief base by line 3 of L_1-P_7 and the current active event that has been added by line 2 of L_1-P_5. The external action of line 4 checks whether that state instantiates an expression. According to the structures declared by Figure 3-13, the pseudo code that achieves this action is as follows:

```
if (action == "instantiate")
 {
   String PState = action.GetTerm (0);  //Get the name of the parental state from the action.
   Int E = action.GetTerm (1);          //Get the serial number of the currently active event
                                        //from the action.
   Boolean Occurrence = false;
   Int AEvent = FE;                     //Hold the position of the first active event by variable
                                        //AEvent.
   while ( AEvent <= LE && ! Occurrence)      //For every active event of the current state.
    {
      if CheckEvent (Events[AEvent].ExpressionPosition, PState)      //This function checks
       {                    //whether the parental state results in a true evaluation; if so then.
         Occurrence = true;
         E = LE – AEvent;               //Calculate the serial number of the occurred event.
       }                                //the belief base of the agent.
      else AEvent = AEvent + 1;         //Otherwise, check the next active event.
    }
   AddPercept(check_event(Occurrence, E));   //Add whether an event has occurred or not
 }                                           //and the corresponding serial number of the
                                             //occurred event or current event.
```

The added percept *check_event(Occurrence, E))* is retrieved by the test goal of line 5. The variables *Occurrence* and *E* are exploited by the achievement goal of line 7. For this goal, L_1-P_{16} and L_1-P_{17} are relevant. In line 6 the belief base is updated by dropping the belief, *pmessage(true)* and adding the belief, *pmessage(fasle)*. That means the received state has been considered and the proper action is currently being taken.

Plan L_1-P_{10}: This plan and L_1-P_9 are relevant to the achievement goal of line 5 of L_1-P_5. Its context is verified true when a message is received from a sibling. It will be the applicable and

means-ends plan when its context is true and the context of L_1-P_9 is false. The role of this plan is to check whether the sender tells or asks about a global measurement.

```
1.  +!sender(Parent, Sibling): Sibling == true
2.        <- ?sibling(Measurement, Subject, SName);
3.           !tell_or_ask(Measurement, Subject, SName).
```

The test goal of line 2 retrieves the belief added by line 3 of plan L_1-P_8. The achievement goal of line 3 checks whether it is an "ask" or "tell" message. For this goal, L_1-P_{11} and L_1-P_{12} are the relevant plans.

Plan L_1-P_{11}: This plan and L_1-P_{12} are relevant to the achievement goal of line 3 of L_1-P_{10}. The context of this plan is verified true when the received message is a tell message. It will be the applicable and means-ends plan when the context is true and the context of L_1-P_{12} is false. The role of this plan is to instantiate the relevant expression with the delivered measurement and to update the contact status.

```
1.  +!tell_or_ask(Measurement, Subject, SName): Subject==tell
2.        <- ?current_event (AE);
3.           arrived_measurement(Measurement, AE);
4.           -+smessage(false);
5.           -+global_measurement(false).
```

The test goal of line 2 retrieves the serial number of the currently evaluated event among the active events; this is added as a belief in line 2 of L_1-P_5. By the external action of line 3, the corresponding expression is instantiated with the received measurement. According to the structures declared in Figure 3-13, this action is implemented by the following pseudo code:

```
if (action == "arrived_measurement")
  {
    Double ArrivedMeasurement = action.GetTerm(0);    //Get the arrived measurement from
                                                      //the action.
    Int AEvent =  LE- action.GetTerm(1);              //From the action, get the serial number of the
                                                      //current event whose expression needs a
                                                      //global measurement and calculate the actual
                                                      //position of that event.
    InstantiateExp (Events[AEvent].ExpressionPosition, ArrivedMeasurement);     //This function
  }                                                   //instantiate the intended expression with the
                                                      //arrived measurement.
```

By line 4, the belief, *smessage(true)* is dropped and the belief, *smessage(fasle)* is added; as the message has been considered and the proper action has accordingly been taken. Similarly, by line 5, Ss_MAG updates its belief base with the belief of no need to wait for an answer, see L_1-P_6.

Plan L_1-P_{12}: This plan and L_1-P_{11} are relevant to the achievement goal of line 3 of L_1-P_{10}. The context of this plan is verified true when the received message is an "ask" message. It would be the applicable and the means-ends plan when its context is verified true and the context of L_1-P_{11} is verified false. The role of this plan is to answer the received message by sending an up-to-date global measurement to the correct Ss_MAG and update the relevant contact status.

```
1.  +!tell_or_ask(Measurement, Subject, SName): Subject==ask
2.        <- retrieve(Measurement);
3.           ?asked(M);
4.           .send(SNmae, achieve, sibling_message(M, tell));
5.           -+smessage(false).
```

The external action of line 2 updates and adds the required measurement to the belief base. The pseudo code to implement this action is as follows:

```
if (action == "retrieve")
  {
    String SensorID = action.GetTerm(0);          //Get the identification of the sensor which
                                                  //provides the needed measurement.
    Double M = GetMeasurement(SensorID);          //This function brings the up-to-date
                                                  //measurement from the intended sensor.
    AddPercept (asked(M));        //Add the intended measurement to the belief base.
  }
```

Line 3 retrieves that measurement and line 4 sends a message (by the internal action *.send*) to the corresponding Ss_MAG. The name of the intended Ss_MAG (SName) is instantiated with the sibling name; this name is already added to the belief base by line 3 of plan L_1-P_8. The *M* holds the measurement and the *tell* instantiates the variable *Subject* of the relevant plan of the recipient. By line 5, the Ss_MAG updates its belief base by dropping the belief, *smessage(true)* and adding the belief, *smessage(fasle)*; as the message has been considered and the proper action has accordingly been taken.

Plan L_1-P_{13}: this plan is the only one relevant to the achievement goal of line 2 of L_1-P_3. As it has no context, it is then the only applicable and the means-ends plan for that goal. Its role is to update the buffers of the expression of the current event and check whether there is a need for a global measurement.

```
1.  +!examine_event(CE)
2.        <- check_update(CE);
3.           ?global_contact (SName, S_id, ExpStatus);
4.           !event_status(SName, S_id, ExpStatus).
```

The external action of line 2 updates and checks the current expression. According to the structures declared in Figure 3-13, the pseudo code to implement this action is as follows:

```
if (action == "check_update")
  {
    Int CEvent =  LE – action.GetTerm(0);          //From the action, get the serial number of the
                                                   //current event, whose expression may need a
                                                   //global measurement, and calculate the actual
                                                   //position of that event.
    Int ExpPos = Events[CEvent].ExpressionPosition;   //Retrieve the position of the
                                                      //expression of the current event.
    if (MeasurementChecker(ExpPos))                //This function checks whether the expression
      {                                            //needs a global measurement. If so, then
        String SiblingName = Sibling (ExpPos);     //This function finds the name of the intended
                                                   //sibling.
        String SID = SensorID (ExpPos);            //This function finds the name of the sensor
                                                   //which delivers the needed measurement.
        AddPercept ("global_contact(SiblingName, SID, true)");   //Add the corresponding
      }                                                          //percept to the belief base.
    AddPercept ("global_contact(none, none, false)");   //Otherwise, add this percept to the
    }                                                   //belief base.
```

The test goal of line 3 retrieves the results of the checking achieved by line 2. The test goal instantiates the (a) *SName* with the name of the intended sibling; (b) *S_id* with the name of the sensor that delivers the required measurement; (c) *ExpStatus* with either true or false according to the need of the contact. The achievement goal of line 4 checks what should be done accordingly. For this goal, L_1-P_{14} and L_1-P_{15} are the relevant plans.

Plan L_1-P_{14}: This plan and L_1-P_{15} are relevant to the achievement goal of line 4 of L_1-P_{13}. The context of this plan is verified true when an expression needs a global measurement. It would be the applicable and means-end plan when its context is verified true and the context of L_1-P_{15} is verified false. The role of this plan is to send an ask message to the intended sibling.

```
1.  +!event_status (SName, S_id, ExpStatus): ExpStatus == true
2.        <- .send(SName, achieve, sibling_message(S_id, ask));
3.           -+global_measurement(true).
```

The internal action of line 2 sends an *ask* message to that sibling. The content part of that message will be unified with the head of a plan that is similar to plan L_1-P_8, but in the recipient's plan library. The variables *SName* and *S_id* are instantiated with respective values, the name of the intended sibling and the name of the needed measurement and the *ask* instantiates the *Subject* of the plan of the recipient. Line 3 updates the belief base with the belief of need to wait for an answer on a global measurement.

Plan L_1-P_{15}: This plan and L_1-P_{14} are relevant to the achievement goal of line 4 of L_1-P_{13}. The context of this plan is verified true when an expression does not need a global measurement. It would be the applicable and means-ends plan when its context is verified true and the context of L_1-P_{14} is verified false. The role of this plan is to evaluate the expression of the current event.

```
1.  +!event_status(SName, S_id, ExpStatus): ExpStatus == false
2.        <- ?current_event(CE);
3.           evaluate(CE);
4.           ?evaluation(Result, CE);
5.           !event(Result, CE).
```

The test goal of line 2 retrieves the current active event; this belief has been added by line 2 of L_1-P_5. The external action of line 3 checks the evaluation result of the given expression and updates the belief base accordingly. According to the structures declared in Figure 3-13, the pseudo code to implement this action is as follows:

```
if (action == "evaluate")
  {
    Int CurrentEvent = action.GetTerm(0);          //Get the serial number of the current event
                                                   //from the action.
    Int CEvent = LE – CurrentEvent                 //Calculate the actual position of that event.
    Int ExpPos = Events[CEvent].ExpressionPosition;     //Retrieve the position of the
                                                        //expression of the current event.
    Int FT= Expressions[ExpPos].FirstToken;        //Retrieve the position of the first token of the
                                                   //expression.
    Int LT= Expressions[ExpPos].LastToken;         //Retrieve the position of the last token of the
                                                   //expression.
    Boolean EvResult = Evaluation(FT, LT);         //This function evaluates the expression
                                                   //through parsing the relevant tokens and
                                                   //buffers (Figure 3-13).
    AddPercept (evaluation(EvResult, CurrentEvent));     //Add this percept to the belief base.
  }
```

Line 4 retrieves the result of the evaluation achieved by the external action. The achievement goal of line 5 checks the occurrence of that event. For this goal, L_1-P_{16} and L_1-P_{17} are the relevant plans.

Plan L_1-P_{16}: This plan and L_1-P_{17} are relevant to the achievement goal of line 6 of L_1-P_9 and line 4 of L_1-P_{15}. Its context is verified true when the evaluation result is false. It will be the applicable and means-ends plan when its context is verified true and the context of L_1-P_{17} is verified false. The role of this plan is to keep evaluating expressions of the active events as long as none of them is verified with a true evaluation.

```
1.  + !event(Result, CE): Result == false
2.        <- ?pmessage(PMSG);
3.           ?smessage(SMSG);
```

```
4.          ?global_measurement(GM);
5.          CE = CE - 1;
6.          +current_event(CE);
7.          !observe(CE, PMSG, SMSG, GM).
```

The test goal of lines 2, 3 and 4 retrieve the contact statuses. Line 5 decreases the serial number of the current event by one to evaluate the next event. Line 6 updates the belief base with the serial number of the current event. For this goal L_1-P_3, L_1-P_4, L_1-P_5 and L_1-P_6 are the relevant plans.

Plan L_1-P_{17}: This plan and L_1-P_{16} are relevant to the achievement goal of line 7 of L_1-P_9 and in line 5 of L_1-P_{15}. Its context is verified true when the evaluation of an expression results in true. It would be the applicable and means-ends plan when its context is true and the context of L_1-P_{16} is false.

```
1.  +!event(Result, AE): Result == true
2.        <- occurred (AE);
3.           ?alarm (Alarm);
4.           !alert(Alarm);
5.           ?controlling(CStatus, CPosition);
6.           !control(CStatus, CPosition);
7.           ?diagnosing(DStatus);
8.           !diagnosis(DStatus);
9.           ?new_state(State);
10.          !transition(State).
```

The role of this plan is to perceive the state-transition table and collect the following beliefs:

a- Alarm status. This can be achieved by the test and achievement goals shown respectively by line 3 and line 4. While the test goal retrieves the alarm clause from the state-transition table, the achievement goal checks whether the occurrence is of a failure event. For the achievement goal, L_1-P_{18} and L_1-P_{19} are the relevant plans.

b- Controlling status. This can be achieved by the test and achievement goals shown respectively by line 5 and line 6. While the test goal retrieves the controlling status from the state-transition table, the achievement goal ascertains whether the controlling status is "after_diagnosis", "none", "impossible" or corrective measures are available. For the achievement goal, L_1-P_{20}, L_1-P_{21} and L_1-P_{22} are the relevant plans.

c- Diagnosis status. This can be achieved by the test and achievement goals shown respectively by line 7 and line 8. While the test goal retrieves the diagnostic status from the state-transition table, the achievement goal ascertains whether that status is "needed", "not_needed" or the underlying causes are already identified in the relevant attribute. For this achievement goal, L_1-P_{23} and L_1-P_{24} are the relevant plans.

d- New state resulting from the state transition. This can be achieved by the test and achievement goals shown respectively by line 9 and line 10. While the test goal retrieves the new state from the state-transition table, the achievement goal takes that state along with the occurred event to find out the name of the new state in order to inform the parent accordingly. For the achievement goal, L_1-P_{25} is the relevant, applicable and means-end plan.

The above role is achieved by the external action of line 2. According to the structures declared in Figure 3-13, the pseudo code to implement this action is as follows:

```
if (action == "occurred")
  {
    Int CurrentEvent = action.GetTerm(0);          //Get the serial number of the current event
                                                   //from the action.
    Int CEvent = LE – CurrentEvent;                //Calculate the actual position of that event.

    String AlarmClause = Events[CEvent].Alarm;     //Retrieve the alarm clause.

    AddPercept (alarm(AlarmClause));   //Add the retrieved clause to the belief base.

    Int CPosition = Events[CEvent].ControllingPosition;     //Retrieve the position of the
                                                            //controlling structure.
    String CStatus = CMeasures[CPosition].ControllingStatus;  //Retrieve the status of the
                                                              //corrective measures.
    AddPercept (controlling(CStatus, CPosition));   //Add the position and the status to the
                                                    //belief base.
    String DStatus = Events[CEvent].Diagnosis;      //Retrieve the diagnosis status.

    AddPercept (diagnosis(Status));                 //Add the status to the belief base.

    if (DStatus == "needed")      //If diagnosis is needed then update the symptoms of the
      {                           //diagnostic model. Assuming that the diagnostic model is a
                                  //fault tree and according to Figure 3-21 take the following
                                  //actions:
      Int DMP = Events[CEvent].DiagnosticModelPosition;     //Retrieve the position of the
                                                            //diagnostic model.
      Int FNP = Nodes[DMP].FirstNodePosition;     //Retrieve the position of the first node
                                                  //of the model.
      Int LNP = Nodes[DMP].LastNodePosition;      //Retrieve the position of the last node
                                                  //of the model.
      UpdatFTModel (FNP, LNP);      //This function updates the symptoms of the nodes.
      }
    Int NState = Events[CEvent].StatePosition;  //Retrieve the position of the structure of the
                                                //new state.
    AddPercept (new_state(NState));             //Add the position to the belief base
  }
```

The part concerned with updating the diagnostic model in the above pseudo code is slightly different if the diagnostic model is a state-machine. According to the declaration of the structures that hold the state-machines diagnostic model (Figure 3-26), that part should have the following pseudo code:

```
    if (DStatus == "needed")      //If diagnosis is needed then update the symptoms of the
      {                           //diagnostic model.
```

```
    Int DMP = Events[CEvent].DiagnosticModelPosition;      //Retrieve the position of the
                                                           //diagnostic model.
    Int FSP = StateMachines[DMP].FirstStatePosition;       //Retrieve the position of the
                                                           //first state of the model.
    Int LSP = StateMachines[DMP].LastStatePosition;        //Retrieve the position of the
                                                           //last state of the model.
    UpdatSMModel (FSP, LSP);        //This function updates the symptoms of the states.
  }
```

Plan L_1-P_{18}: This plan and L_1-P_{19} are relevant to the achievement goal of line 4 of L_1-P_{17}. Its context is verified true when the occurrence is for a failure event. It will be the applicable and means-ends plan when the context is verified true and the context of L_1-P_{19} is false. The role of this plan is to announce an alarm.

```
1.  +!alert(Alarm): Alarm \== “none”
2.          <- announce(Alarm).
```

Line 2 is an external action to announce the quoted alarm’s clause. The pseudo code to implement this action is as follows:

```
if (action == “announce”)
  {
    String AlarmClause =  action.GetTerm(0);     //Get the alarm clause from the action.
    AnnounceAlarm (AlarmClause);       //This function announces the corresponding alarm on
  }                                    //the operators’ interface.
```

Plan L_1-P_{19}: This plan and L_1-P_{18} are relevant to the achievement goal of line 4 of L_1-P_{17}. Its context is verified true if the occurrence is for a normal or corrective event. It will be the applicable and means-ends plan if the context is verified true and the context of L_1-P_{18} is verified false. As there is no need for an alarm, this plan does nothing.

```
+!alert(AE, AlarmStatus): AlarmStatus == “none”.
```

Plan L_1-P_{20}: This plan as well as L_1-P_{21} and L_1-P_{22} are relevant to the achievement goal of line 6 of plan L_1-P_{17}. Its context is verified true when fault control should be achieved after diagnosing the underlying cause. It will be the applicable and means-ends plan if the context is verified true and the contexts of the L_1-P_{21} and L_1-P_{22} are verified false. The role of this plan is to launch a fault controlling process after completion of a diagnostic process.

```
1.  +!control(CStatus, CPosition): CStatus == “after_diagnosis”
2.          <- -+measure(after_diagnosis).
```

Line 2 updates the belief base by adding a belief that implies controlling is achievable after diagnosing the underlying causes.

Plan L_1-P_{21}: This plan as well as L_1-P_{20} and L_1-P_{22} are relevant to the achievement goal of line 6 of plan L_1-P_{17}. Its context is verified true when control is not needed or is impossible. It will be the applicable and means-ends plan when the context is verified true and the contexts of L_1-P_{20} and L_1-P_{22} are verified false. The role of this plan is to update the belief base correspondingly.

```
1. +! control(CStatus, CPosition): CStatus == “none” | CStatus == “impossible”.
2.                                  <- -+measure(none).
```

Line 2 updates the belief base by adding the belief, “none”, which means that corrective measures are unavailable.

Plan L_1-P_{22}: This plan as well as L_1-P_{20} and L_1-P_{21} are relevant to the achievement goal of line 6 of plan L_1-P_{17}. Its context is verified true when corrective measures are available. It will be the applicable and means-ends plan when the context is true and the contexts of L_1-P_{20} and L_1-P_{21} are false. The role of this plan is to apply the corrective measures.

```
1.  +! control(CStatus, CPosition): CStatus \== “after_diagnosis” & CStatus \== “none” &
                                      ContStatus \== “impossible”
2.                                  <- apply_measures(CPosition).
```

Line 2 applies the available measures either by instructing the controller of the monitored system or by directly instructing the relevant actuators. According to the structures declared in Figure 3-13, the pseudo code to implement this action is as follows:

```
if (action == “apply_measures”)
  {
    Int MPosition = action.GetTerm(0);          //Get the position of the structure of the
                                                //corrective measures from the action
    Int FMT = CMeasures[MPosition].FirstCMToken;    //Retrieve the position of the first token
                                                    //in the tokens array.
    Int LMT = CMeasures[MPosition].LastCMToken;     //Retrieve the position of the last token
                                                    //in the tokens array.
    Apply(FMT, LMT);                                //This function applies the corrective
  }                                                 //measures.
```

Plan L_1-P_{23}: This plan and L_1-P_{24} are relevant to the achievement goal of line 8 of L_1-P_{17}. Its context is verified true if the underlying causes are identified in the state-transition table; the occurred failure event and the causes are in a one-to-one relationship. It will be the applicable

and means-ends plan when the context is true and the context of L_1-P_{24} is false. It is role is to add the underlying causes to the diagnostic list.

```
1. +! diagnosis(DStatus): DStatus \== “not_needed” && DStatus \== “needed”
2.                          <-  cause(DStatus);
3.                              -+diagnosis_status(false).
```

Line 2 adds the causes to the diagnostic list. The pseudo code to implement this action is as follows:

```
if (action == “cause”)
  {
    String Cause = action.GetTerm(0);          //Get the cause from the action.
    AnnounceCause (Cause);            //This function adds the cause to the diagnostics list on
  }                                   //the operators’ interface.
```

Plan L_1-P_{24}: This plan and L_1-P_{23} are relevant to the achievement goal of line 8 of L_1-P_{17}. Its context is verified true when the diagnostic process is needed. It will be the applicable and means-ends plan when it context is verified true and the context of L_1-P_{23} is verified false.

```
1.  +! diagnosis(DiagStatus): DiagStatus == “needed”
2.                              <- -+diagnosis_status(true).
```

Line 2 updates the belief base with a belief that implies diagnosis is needed. This belief is the basis for selection of the applicable and means-ends plan between L_1-P_{26} and L_1-P_{27}. These plans will be relevant to the achievement goal of line 10 of L_1-P_2.

Plan L_1-P_{25}: This plan is the only one relevant to the achievement goal of line 10 of L_1-P_{17}. Thus, it is the applicable and means-ends plan for that goal. The role of this plan is to find out the name of the new state, inform the parent about that state and launch a monitoring cycle for the active events of the new state.

```
1.  +!transition(State)
2.        <- state_name(State);
3.           ?name(StateName);
4.           ?parent_name(PName);
5.           .send(Parent_Nmae, achieve, message(StateName));
6.           !monitoring(State).
```

The external action of line 2 finds out and adds the name of the new state to the belief base. According to the structures declared in Figure 3-13, the pseudo code to implement this action is as follows:

```
if (action == "state_name")
  {
    Int NSPosition= action.GetTerm(0);   //Get the position of the new state from the action.
    String Name = States[NSPosition].StateName;          //Retrieve the name of the new state.
    AddPercept(name(Name));              //Add the name of that state to the belief base.
  }
```

The name of the new state is retrieved from the belief base by the test goal of line 3. Line 4 retrieves the parent name; the name of the parent has been added by line 1 of the initial beliefs. Line 5 informs the parent about the new state. The achievement goal of line 6 launches a monitoring cycle for the active events of the new state. The relevant, applicable and means-ends plan of this goal is L_1-P_2.

Plan L_1-P_{26}: This plan and L_1-P_{27} are relevant to the achievement goal of line 10 of L_1-P_2. Its context is verified true when a diagnostic process is needed. It is the applicable and means-ends plan when the context is verified true and the context of L_1-P_{27} is verified false. The role of this plan is to diagnose the causes of the occurred failure and check whether fault controlling is needed thereafter. This is implemented either by the diagnostic algorithm shown in Figure 3-22 (when the diagnostic model is derived from a fault tree) or by the diagnostic algorithm shown in Figure 3-27 (when the diagnostic model is derived from state-machines).

```
1.  +! diagnose(Need): Need == "true"
2.        <- ?current_event(FE);
3.           diagnosis(FE);
4.           ?measure(MStatus)
5.           !causes_control(MStatus);
6.           -+diagnosis_status(false).
```

The test goal of line 2 retrieves the verified failure event; this belief has been added by line 2 of L_1-P_5 and updated along with the monitoring cycle by line 6 of L_1-P_{16}. The external action of line 3 launches a diagnostic process and announces also the name of the underlying causes. According to the structures declared in Figure 3-13, the pseudo code to implement this action is as follows:

```
if (action == "diagnosis")
  {
    Int E = action.GetTerm(0);     //Get the serial number of the occurred event among the active
```

```
                                    //events.
   Int CEvent = LE – E;             //Calculate the actual position of the structure of that event.

   Int DMP = Events[CEvent].DiagnosticModelPosition;  //Retrieve the position of the diagnostic
                                                      //model of the occurred failure event.
   Boolean Diagnostic = Diagnosis (DMP);              //Call the diagnostic function, which is
                                                      //as shown in Figure 3-22.
   if (Diagnostic)                                    //If the function returns true.
     {
       For ( int i = 0; i < a; i++)          //For every diagnosed cause, where a is the number of
         {                                   //the diagnosed causes in the diagnostics array. As
                                             //shown in Figure 3-22 this array holds the positions of
                                             //the diagnosed nodes.
           Int NP = DiagnosticArray[i];      //Retrieve the position of a diagnosed node.

           String Cause = Nodes[NP]. Fault;  //Retrieve the fault clause enclosed by the
                                             //diagnosed node.
           AnnounceCause (Cause);            //This function adds the cause to the
         }                                   //diagnostics list on the operators' interface.
     }
 }
```

This pseudo code is different if the diagnostic mode is a state-machine. According to the declaration of the structures (Figure 3-26) and algorithm (Figure 3-27) of the diagnostic state-machines, the pseudo code is as follows:

```
if (action == "diagnosis")
  {
    Int E = action.GetTerm(0);
    Int CEvent = LE – E;
    Int DMP = Events [CEvent].DiagnosticModelPosition;
    Int EP= StateMachines[DMP]. EventPropagator;   //Retrieve the position of the
                                                   //propagator which results in the
                                                   //verified event.
    Boolean Diagnostic = Diagnosis (EP);           //Call the diagnostic function, which is
                                                   //as shown in Figure 3-27.
    if (Diagnostic)                                //If the function returns true.
      {
        For ( int i = 0; i < a; i++)          //For every diagnosed cause, where a is the number of
          {                                   //the diagnosed causes in the diagnostics array. As
                                              //shown in Figure 3-27 this array holds the positions of
                                              //the diagnosed states.
            Int SP = DiagnosticArray[i];      //Retrieve the position of a diagnosed state.

            String Cause = States[SP].Fault;  //Retrieve the fault clause enclosed by the
                                              //diagnosed states.
            AnnounceCause (Cause);            //This function adds the cause to the
          }                                   //diagnostics list on the operators' interface.
      }
  }
```

The test goal of Line 4 of the above plan retrieves and keeps a record about whether fault controlling is needed after diagnosing the causes; this belief is added by line 2 of L_1-P_{20} or line 2 of L_1-P_{21}. The achievement goal of line 5 checks that record, in order to do what is needed. For

this goal, L_1-P_{28} and L_1-P_{29} are the relevant plans. Line 7 updates the belief base to reflect that a diagnostic process is no longer necessary as it has been achieved.

Plan L_1-P_{27}: This plan and L_1-P_{26} are relevant to the achievement goal of line 10 of L_1-P_2. Its context is verified true when a diagnostic process is not needed. It is the applicable and means-ends plan when the context is true and the context of L_1-P_{26} is false. There is no action to be taken in the body of this plan.

```
+! diagnose(Need): Need == "false".
```

Plan L_1-P_{28}: This plan and L_1-P_{29} are relevant to the achievement goal of line 5 of plan L_1-P_{26}. Its context is verified true when control should be achieved after diagnosing the cause of that failure. It is the applicable and means-ends plan when the context is true and the context of L_1-P_{29} is false. The role of this plan is to apply corrective measures associated with the diagnosed underlying causes.

```
1.  +!cause_control(MStatus): MStatus == "after_diagnosis"
2.        <- apply (measures).
```

The external action of line 2 applies corrective measures associated with the underlying cause. The pseudo code to implement this action is as follows:

```
if (action == "apply")
  {
    For (Int i = 0; i < a; i++)        //For every diagnosed cause, where a is the number of
      {                                //the diagnosed causes in the diagnostics array. As
                                       //shown in Figure 3-22, this array holds the positions of the
                                       //diagnosed nodes.
        Int NP = DiagnosticsArray [i];       //Retrieve the position of the diagnosed node.

        Int MStatus = Nodes[NP]. MeasureStatus;//Retrieve the status of the corrective
                                                    //measures.
        if (MStatus != none)                 //If the status is not none.
          {
            Int FMT = Nodes[NP].FirstMeaToken;  //Retrieve the position of the first token of the
                                                //measures.
            Int LMT = Nodes[NP].LastMeaToken;  //Retrieve the position of the last token of the
                                                //measures.
            Apply(FMT, LMT);                 //This function applies the corrective measures.
          }
      }
  }
```

This pseudo code is different if the diagnostic mode is a state-machine. According to the declaration of the structures (Figure 3-26) and algorithm (Figure 3-27) of the diagnostic state-machines, the pseudo code is as follows:

```
if (action == "apply")
  {
    For (Int i = 0; i < a; i++)        //For every diagnosed cause, where a is the number of
      {                                //the diagnosed causes in the diagnostics array. As
                                       //shown in Figure 3-27 this array holds the positions of the
                                       //diagnosed states.
       Int SP = DiagnosticsArray [i];      //Retrieve the position of a diagnosed state.

       Int MStatuse = States[NP]. MeasureStatus;      //Retrieve the status of the corrective
                                                      //measures.
       if (MStatus != none)                //If the status is not none.
         {
           Int FMT = States[NP].FirstMeaToken;  //Retrieve the position of the first token of the
                                                //measures.
           Int LMT = States[NP].LastMeaToken;   //Retrieve the position of the last token of the
                                                //measures.
           Apply(FMT, LMT);                //This function applies the corrective measures.
         }
      }
  }
```

Plan L_1-P_{29}: This plan and L_1-P_{28} are relevant to the achievement goal of line 5 of L_1-P_{26}. Its context is verified true when fault control is not needed after diagnosing the causes. It will be the applicable and means-ends plan when the context is verified true and the context of L_1-P_{28} is verified false.

+!cause_control(MStatus): MStatus == "none".

There is no action to be achieved.

4.7.3 Plan Library of Ss_MAGs of Levels Extending from Level*2* to Level*n-1*

The initial beliefs of each Ss_MAG of levels extending from level*2* to level*n-1* are as follows:

1. parent(parent_name).
2. children(child1, child2, child3).
3. subsystem(SsName).
4. start(monitoring).

By lines 1, 2 and 3 the Ss_MAG has the respective beliefs, the name of the parent, names of the children and the name of the monitored sub-system. There are three assumed children whose names are as follows: child1, child2 and child3. Line 4 is a belief to monitor the assigned sub-system. For this belief, $L_{2,n-1}$-P_1 is the relevant, applicable and means-ends plan.

$L_{2,n-1}$-P_1: The role of this plan is to identify and regard the initial state as the current state.

```
1.  +start(monitoring)
2.      <- !monitoring(0).
```

The achievement goal of line 2 launches the monitoring process. For this goal, $L_{2,n-1}$-P_2 is the relevant, applicable and means-ends plan.

$L_{2,n-1}$-P_2: The role of this plan is to provide system operators with assessment and guidance on the current conditions.

```
1.  +!monitoring(CState)
2.      <- +current_state (CState);
3.          ?subsystem(SsName);
4.          feedback(CState,SsName).
```

Line 2 updates the belief base of the agent with the current state. The test goal of line 3 retrieves the name of the monitored sub-system. The external action of line 4 provides the operators with assessment and guidance (feedback) quoted from the current state. According to the structures declared in Figure 3-16, the pseudo code to implement this action is as follows:

```
if (action == "feedback")
  {
    Int SPos = action.GetTerm (0);          // Get the position of the current state from the action.
    String SubSysN = action.GetTerm (1);          //Get the name of the monitored sub-system
                                                  //from the action.
    AnnounceAss(States[SPos].Assessment, SubSysN); //From the current state announce
                                                   //assessment for the monitored
                                                   //sub-system.
    AnnounceGuid(States [SPos].Guidance, SubSysN);  //From the current state announce
  }                                                 //guidance for the monitored
                                                    //sub-system.
```

After achieving this plan the given Ss_MAG becomes idle unless it receives a message from the parent or a child, which triggers the next plan.

$L_{2,n-1}$-P_3: This plan is the relevant, applicable and means-ends one when the given Ss_MAG receives (a) a message from its parent, sent in the way shown in lines 6, 7, and 8 of $L_{2,n-1}$-P_5; or (b) a message from one of its children, sent in the way shown in line 4 of $L_{2,n-1}$-P_5. The role of this plan is to pass the received state (RState) to be checked by $L_{2,n-1}$-P_4.

```
1.  +!message(RState)
2.      <- !transition_checker(RState).
```

For the achievement goal of line 2, $L_{2,n-1}$-P_4 is the relevant, applicable and means-ends plan.

$L_{2,n-1}$-P_4: The role of this plan is to evaluate the currently active events and check whether the received state triggers a state transition.

```
1.  +!transition_checker(RState)
2.        <- ?current_state(CState);
3.           transition(CState, RState);
4.           ?check(Trans, State);
5.           !is_transition(Trans, State).
```

The test goal of line 2 retrieves the current state; this belief has been added by line 2 of $L_{2,n-1}$-P_2. The external action of line 3 evaluates the expressions of the active events according to the received state. According to the structures declared in Figure 3-16, the pseudo code to implement this action is as follows:

```
if (action == "transition")
  {
    Int CSPos = action.GetTerm (0);       //Get the position of the current state from the action.

    String RSName = action.GetTerm (1);        //Get the name of the received state from the
                                               //action.
    Boolean Occurrence = false;

    Int LE = States[StatePos].LastEvent;       //Retrieve the position of the last event of the
                                               //current state.
    Int FE = States[StatePos].FirstEvent;      //retrieve the position of the first event of the
                                               // current state.
    if (LE != nil && FE != nil)                //If the current state has a number of the active
      {                                        //events.
        while (FE <= LE && ! Occurrence)       //For every event, achieve the following:
          {
            Int EP = Events[FE].ExpressionPosition;     //Retrieve the position of the structure
                                                        //of the relevant expression.
            Int FT = Expression[EP].FirstToken;         //Retrieve the position of the first token
                                                        //of that expression.
            Int LT = Expression[EP].LastToken;          //Retrieve the position of the last token
                                                        //of that expression.
            Occurrence = Evaluate(FT, LT, RSName);      //For those tokens, this function
                                                        //checks whether the received state
                                                        //triggers a state transition.
            FE = FE+ 1;
          }
        if (Occurence)                         //If that state triggers a transition.
          {
            CSPos = Events[FE-1].StatePosition;         //Retrieve the position of the new
                                                        //state.
            AddPercept(check(true, CSPos));             //Add true as transition has occurred
                                                        //and the position of the structure of
                                                        //the new state to the belief base.
            String SN = States [CSPos].StateName;       //Retrieve the name of the new state.
            AddPercept(new_state(SN));                  //Add that name to the belief base.
          }
```

```
        else AddPercept(check(false, CSPos));          //Otherwise, add false as transition has
    }                                                  //not occurred and the position of the
}                                                      //structure of the current state.
```

Line 4 retrieves the result of the checking achieved by the external action. If a transition has occurred then the values of the *Trans* and *State* are instantiated with true and the name of the new state, respectively. Otherwise, those values are instantiated with false and the name of the current state, respectively. The achievement goal of line 5 determines what to do in either case. For this goal $L_{2,n-1}$-P_5 and $L_{2,n-1}$-P_6 are the relevant plans.

$L_{2,n-1}$-P_5: This plan and $L_{2,n-1}$-P_6 are relevant to the achievement goal of line 5 of $L_{2,n-1}$-P_4. Its context is verified true when the received message triggers a state transition. It will be the applicable and means-ends plan when its context is verified true and the context of $L_{2,n-1}$-P_6 is verified false. The role of this plan is to communicate the new state to the parent and children.

```
1.  +!is_transition(Trans, State): Trans == true
2.        <- ?new_state(SName);
3.           ?parent(PName);
4.            .send(Pname, achieve, message(SName));
5.            ?children(C1, C2, C3);
6.            .send(C1, achieve, message(SName));
7.            .send(C2, achieve, message(SName));
8.            .send(C3, achieve, message(SName));
9.            !monitoring(State).
```

Line 2 retrieves the name of the new state (added by the above pseudo code). Line 3 retrieves the name of its parent and accordingly line 4 informs the parent about the new state by sending a message. Similarly, line 5 retrieves the names of the children and informs them about the new state by sending a message to each of them, as shown by lines 6, 7 and 8. The achievement goal of line 9 launches the monitoring to the new state. For this achievement goal, plan $L_{2,n-1}$-P_2 is the only relevant plan.

$L_{2,n-1}$-P_6: This plan and $L_{2,n-1}$-P_5 are relevant to the achievement goal of line 5 of $L_{2,n-1}$-P_4. Its context is verified true when the received state does not trigger a state transition. It will be the applicable and means-ends plan when its context is verified true and the context of $L_{2,n-1}$-P_5 is verified false. This plan has no action to be achieved.

```
+!is_transition(Trans, State): Trans == false.
```

4.7.4 Plan Library of S_MAG

The plan library of the S_MAG is quite similar to those of Ss_MAG of levels extending from level*2* to level*n-1*. However, the only difference is that, as the S_MAG does not have a parent, it exchanges messages with children only.

Plans L_n-P_1, L_n-P_2, L_n-P_3, L_n-P_4 and L_n-P_6 of the S_MAG are as the same as the respective plans $L_{2,n-1}$-P_1, $L_{2,n-1}$-P_2, $L_{2,n-1}$-P_3, $L_{2,n-1}$-P_4 and $L_{2,n-1}$-P_6 of Ss_MAGs of levels extending from level*2* to level*n-1*. The difference can be reflected in the initial belief and plan L_n-P_5 as follows:

The initial beliefs of the S_MAG are:

```
1.    children(child1, child2, child3).
2.    system(SName).
3.    start(monitoring).
```

Line 1 and line 2 add the respective beliefs, names of the children and the name of the monitored system. There are three assumed children whose names are as follows: child1, child2 and child3. Line 3 is a belief of monitoring the assigned system.

L_n-P_5: By this plan, the S_MAG informs its children about any new state.

```
1.  +!is_transition(Trans, State): Trans == true
2.      <- ?NewState(SName);
3.         ?children(C1, C2, C3);
4.         .send(C1, achieve, message(SName));
5.         .send(C2, achieve, message(SName));
6.         .send(C3, achieve, message(SName));
7.         !monitoring(State).
```

4.8 Reasoning and Logical Omniscience Problem of the Monitoring Agents

The logical omniscience problem is an epistemic problem that may limit the ability of agents to deal with conditions that may fall beyond their reasoning ability (Halpern and Pucella, 2011; Vardi, 1986).

This problem has been addressed to some extent during the design and implementation of the monitoring agents. Monitoring agents have been provided with collaboration protocols that can assure effective integration among their monitoring models and sharing of sensory measurements. They are also programmed to deliver the safety tasks on observations that are acknowledged by their monitoring models.

Care has been taken to define techniques that can guarantee as much as possible the completeness of the model, e.g. HAZOP among techniques by which investigation of all possible deviations from normal behaviour is possible. Agents can reason to some extent about uncertainty using the three value logic employed in the evaluation of the truth value of monitored events. However, more sophisticated techniques, such as Bayesian inference, should be incorporated in the future to improve omniscience.

The reasoning of the monitoring agents is illustrated in Figure 4-13 which gives the interpretation steps[1] of the Jason agent. The steps show how the agents reason on their plan libraries according to their perceptions and received messages. The reasoning steps are exemplified by the plan libraries discussed above.

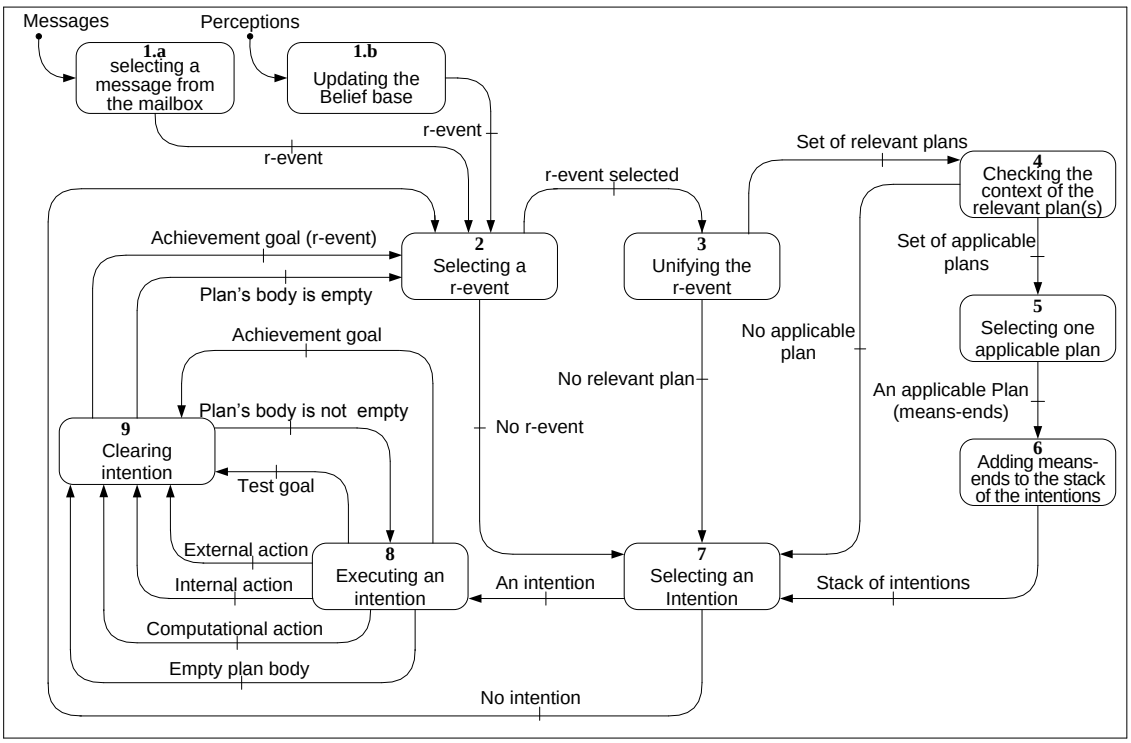


Figure 4-13: State Transitions of the Jason Interpreter During a Reasoning Cycle.

Step 1: Selecting a message from the mailbox (1.a) or updating the belief base (1.b)

The reasoning cycle is launched upon either (a) receiving a message (step 1.a), e.g. plan $L_{2,n-1}$-P_3 is triggered upon receiving a message and thus a reasoning cycle is launched; (b) updating the belief base by perceiving the environment (step 1.b), e.g. line 3 of plan L_1-P_{15} by which a reasoning cycle is launched after perceiving the occurrence of an event.

[1]Those steps are illustrated according to the Jason compiler. Figure 4-13 and the associated explanations are drawn from Vieira *et al.* (2007) and Jason interpreter and formal semantic stated respectively in chapters four and ten of Bordini *et al.* (2007).

Received messages are held by a mailbox and by an interpretation function called the message selection function a received message is selected to be a reasoning event (r-event). Similarly, every perceived belief is held by the belief base and by two interpretation functions, the belief update and belief revision functions; the up-to-date belief is selected as r-event.

Apart from the messages and perceptions, the agent may have an already-provided belief that can also trigger a reasoning cycle, e.g. the initial belief of Ss_MAG of level*1* (section 4.7.2).

Step 2: Selecting an r-event

In this step, an r-event is selected from those presented by the previous step and achievement goals that appear in the body of an under-execution plan[1]. For this purpose, Jason interpreter is provided with an interpretation function called the event selection function. This function is customisable; it can be modified to change the priorities for selecting an r-event. In the original implementation, every r-event has a priority based on which it is taken through a reasoning cycle. The highest priority is given to the r-events of the received messages, followed by those of the up-to-date belief and the lowest priority is given to the r-events of the achievement goals.

If there is no r-event and the intention stack[2] is not empty then the top intention in that stack is selected to be executed; see the state transition from step 2 to step 7 and from there to step 8. If there is no r-event and the intention stack is empty, then the agent resumes step 2 and keeps idle until it receives a message or perceives a new belief; see the state transition from step 2 to step 7 and from there to step 2 again. The idle situation has been exemplified during the programming process of the Ss_MAGs of levels extending from level*2* to level*n-1*.

Step 3: Unifying the r-event

By this step the selected r-event is unified with all trigger events of the plans' heads in the plan library. Plans with trigger events that can be unified with the r-event are selected as relevant plans, by means of an interpretation function called the unifying function. In the original design of this function, if there is no plan relevant to the r-event, then that r-event is simply discarded and step 7 is triggered to execute the intention that appears at the top of the intentions stack. This is shown by the state transition from step 3 to step 7.

[1] Further details are provided in step 9.

[2] Jason interpreter is provided with a stack data structure to hold the intentions whose executions require recursive execution of other intentions. Further explanations of this stack will be given in step 6 and step 7 of the compilation steps.

In the design and implementation of the monitoring agents, careful attention has been paid to providing corresponding plans for every potential r-event. This has also been demonstrated in discussing the plan libraries; section 4.7.2 and section 4.7.3.

Step 4: Checking the context of the relevant plans

To achieve this step the interpreter is provided with a function called the context checking function, whose role is to check the context of the relevant plans (result from step 3) and identify the applicable plans. Consider, for example, plans L_1-P_{11} and L_1-P_{12}; after unifying their heads with the achievement goal of line 3 of plan L_1-P_{10} they have been selected as relevant plans, their contexts are further checked and thus one of them becomes the applicable and means-ends plan.

If there is no applicable plan then the selected r-event is discarded and either (a) a new reasoning cycle is launched by selecting another r-event; such a case may occur when the stack of intentions is empty; see the transition from step 4 to step 7 and from there to step 2; (b) executing the intention that appears at the top of the intentions stack when the stack of the intentions is not empty, see the state transition from step 4 to step 7 and from there to step 8. If this step results, on the contrary, in more than one applicable plan, then the next interpretation step selects only the applicable plan, as means-ends, to be executed.

Step 5: Selecting one applicable plan

This step is achieved by an interpretation function called the applicable plan selection function and results in selecting the means-ends plan. However, this step is needed only when there is more than one applicable plan for the same r-event. During the design and programming process of the monitoring agents it was unnecessary to write more than one plan for the same r-event. Hence, this step would not be applicable in the interpretation of the monitoring agents.

Step 6: Adding the means-ends plan to the set of the intention.

In this step, the means-ends plan is pushed into the intentions stack to be executed.

Step 7: Selecting an intention

This step is achieved by an interpretation function called the intention selection function. The function selects the first means-ends plan that appears at top of the stack to be executed in the next step. If the intention stack is empty then a new reasoning cycle is launched, as shown by the state transition from step 7 to step 2.

Step 8: Executing an intention

In this step the selected intention is executed. As the intention is implemented as a plan, actions that appear in the plan body are executed one by one. If the action is an achievement goal then it is removed from the plan body and regarded as an r-event, for which there will be a new reasoning cycle and a means-ends plan, as shown by the state transition from step 8 to step 9 and from there to step 2. That plan will, in turn, be pushed into and appear at the top of the intention stack. Consider, for example, the achievement goal of line 2 of plan L_1-P_3; to execute that goal, plan L_1-P_3 is suspended and plan L_1-P_{13} is pushed into the top of the intention stack. After implementing L_1-P_{13} the execution of L_1-P_3 is resumed.

If the action is a test goal[1], external action, internal action or computational action, then it is executed and removed from the plan body. After every execution, the plan body is checked to verify whether there are further actions to be taken. If so, then actions are also executed until the body is empty.

Step 9: Clearing intention

In this step every executed action is removed from the plan body. After removal of an action, the plan body is checked for the existence of more actions. If there remain other actions, then step 8 is re-launched until the end of the plan body. On the contrary, if the plan body is empty, then it is removed from the top of the intention stack, i.e. clearing intention, and a new reasoning cycle is launched. A new reasoning cycle is also launched when the action is an achievement goal; see the transition from step 9 to step 2.

[1] Some test goals are similar to achievement goals and require one or more plans to be executed. Such a case was not needed during the programming of the monitoring agents and thus it is not included, for the sake of brevity.

4.9 Summary

This chapter has developed the multi-agent system; the second constituent of the proposed monitor. In this context, the appropriateness of BDI agents as monitoring agents, deployment approach, collaboration protocols, design and implementation of the monitoring agents have been discussed.

Thus far, the approach and the two constituents of the proposed monitor have been developed. In the following two chapters, the monitor is applied to two different case studies. In Chapter Five, the monitor is applied to an aircraft fuel system, while in Chapter Six it is applied to an aircraft brake system.

Chapter Five

Distributed On-line Safety Monitor Based on HiP-HOPS and a Case Study of an Aircraft Fuel System

5.1 Introduction

This chapter explains and demonstrates how the distributed monitoring model can be derived from the HiP-HOPS safety assessment model. It also demonstrates application of the monitor to a case study, which was performed on a simulated model of an aircraft fuel system (AFS), and evaluates the effectiveness of the monitoring tasks delivered by the monitor.

5.2 Aircraft Fuel System (AFS)

Figure 5-1 illustrates the physical configuration of the model AFS. According to the illustration, the AFS feeds the port and starboard engines of a two-engine aircraft with fuel and it consists of:

- Seven identical-capacity tanks to store fuel, each tank coupled with a level sensor to measure the fuel levels. The tanks are deployed in such a way as to maintain the best balance of fuel across the aircraft body. Two tanks are positioned in each wing, left and right, and three tanks are positioned longitudinally across the fuselage. To avoid any imbalance across the aircraft body, fuel level in the seven tanks should be even or symmetrical across the two axes.
- Eleven fuel pumps each coupled with a speed sensor (rotary decoder). They can pump fuel in two different directions and at a variable rate to support the demanded thrust and maintain an even level among the seven tanks. The speed sensors may deliver positive or negative measurements; each indicates a different flow direction.
- Twelve valves each coupled with a position sensor. They collaborate with pumps to implement Active Fault-Tolerant Controlling (AFTC). These valves allow fuel flow through certain paths during normal conditions and isolate faulty components to tolerate faults.
- Twelve flow meters to measure fuel volume flows across the fuel system. Their measurements could be negative or positive, indicating both flow directions.
- Two jettison points each consisting of a valve, flow meter and pump. They dump fuel into the atmosphere during an in-flight emergency, to reduce the weight of the plane so that emergency landing can be achieved more safely.
- Fuel pipes to connect the tanks, valves, flow meters and pumps together and link them to the refuelling point, two engines and two jettison points.

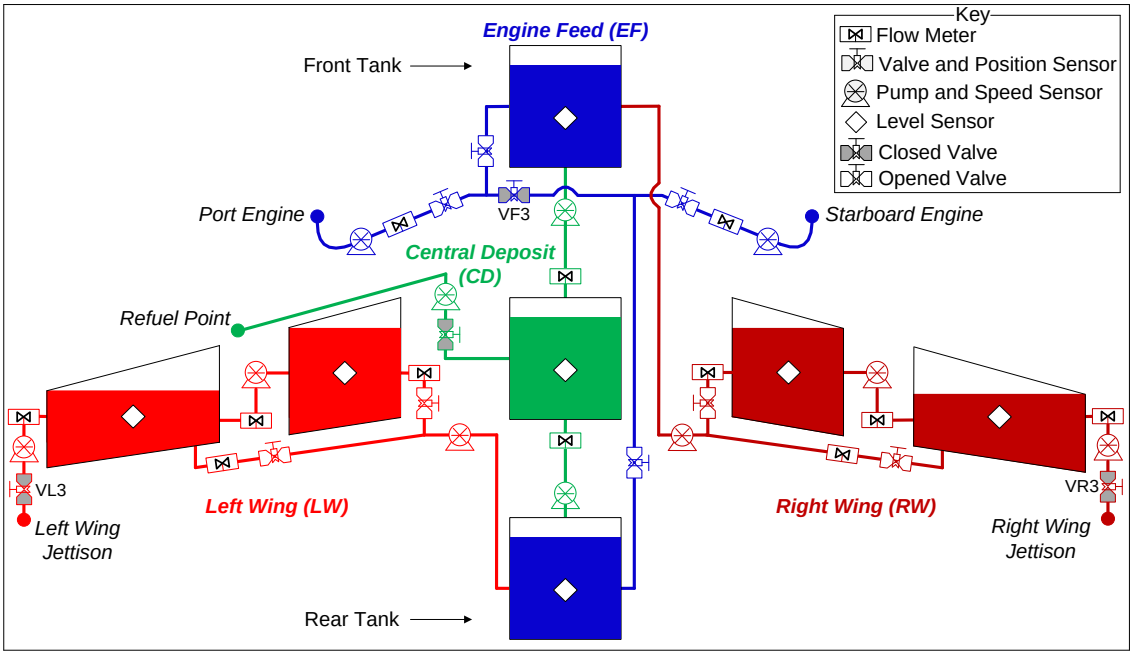


Figure 5-1: Illustration of the Aircraft Fuel System (AFS).

The fuel tanks along with the other basic components are arranged in four sub-systems: Engine Feed (EF) sub-system, Central Deposit (CD) sub-system, Left Wing (LW) sub-system and Right Wing (RW) sub-system. Each sub-system is indicated by a different colour; blue, green, red and brown, denoting the EF, CD, LW and RW, respectively.

The typical functionality of the fuel system can be defined as fuel storage and distribution across two different modes; refuelling and consumption. During the refuelling mode, the fuel system evenly distributes fuel injected from the refuel point into the seven tanks. On the contrary, during the consumption mode, the system evenly draws fuel from the seven tanks to feed the two engines at a variable flow rate.

A Fuel System Control Unit (FSCU), which is a computerised controlling sub-system, is employed to deliver the functionality of the fuel system. The detailed role of the FSCU is as follows:

- During the consumption mode, FSCU applies a feedback-control algorithm to draw the demanded fuel rate (R) of each engine. To achieve even drawing and avoid violating central gravity criteria, FSCU obtains measurements from the flow meters and level sensors, regulates the speed of the pumps and puts the valves in the correct closed or open positions. In normal conditions, valve VF3 is closed and thus the port engine is fed from the front tank and the starboard engine is fed from the rear tank. The two jettison valves, VL3 and VR3, are also closed as long as jettison is not needed.
- During the refuelling mode, FSCU applies another feedback-control algorithm to assure even distribution of the fuel injected from the refuelling point to the seven tanks. In addition to

affecting the body balance, injecting extra or uneven fuel may result in rupturing a tank. To avoid this, FSCU uses the flow meters and level sensors to measure the distribution, regulate the speed of the pumps and control the positions of the valves.

- Transmitting alarms to the cockpit and measurements to other computers (of the aircraft) concerning the FSCU status. According to the delivered alarms, pilots could command the corresponding corrective measures to avoid exacerbation. For example, when a fuel leak occurs in a tank of the LW or RW sub-system[1], the pilot could apply a certain procedure to command the FSCU to isolate the affected tank, close the relevant valves and jettison the isolated fuel.

5.2.1 Hierarchical Architecture of the Aircraft Fuel System

The hierarchy of the fuel system consists of three architectural levels: system level (level*2*), sub-system level (level*1*) and basic components level (level*0*). Figure 5-2 illustrates the system level as a Simulink block. The system has six inputs: refuel point, two auto-throttle lines to convey the demand fuel rates of the starboard and port engines, pilot commands line to deliver the flying instructions and two redundant power lines (power1 and power2). It has also five outputs: alarm line, port engine feed line, left wing jettison line, right wing jettison line and starboard engine feed.

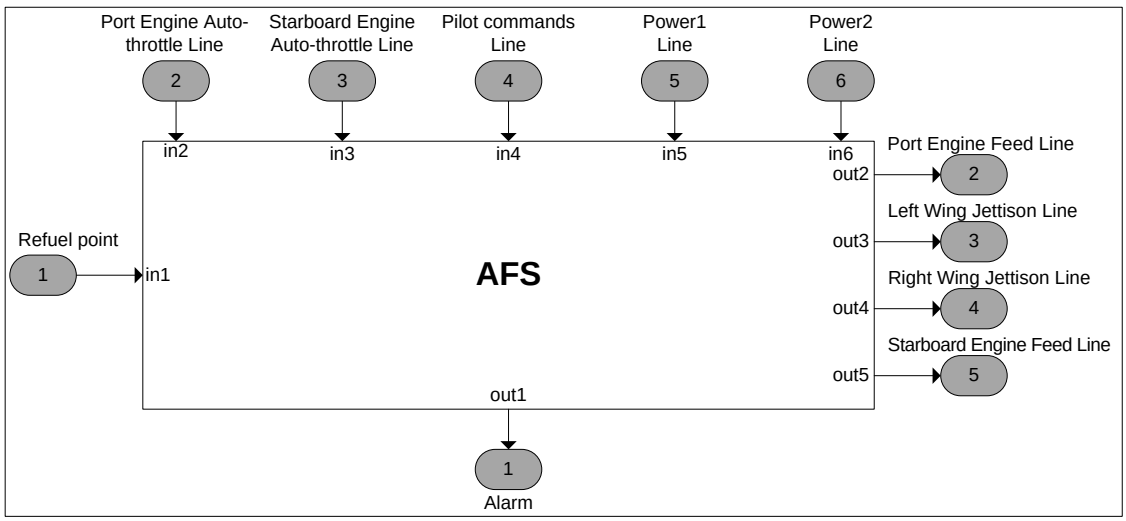


Figure 5-2: The System Level of the Hierarchical Architecture of the AFS.

Figure 5-3 illustrates level*1*of the fuel system as a Simulink model. At this level, the system appears as five sub-systems, each of which appears as a block. The sub-systems have inputs and

[1] Tanks of these sub-systems are susceptible to a structural leak due to the possibility of a bird strike (Langton, 2009).

outputs as mutual interfaces and interactions with other external devices related to the fuel system.

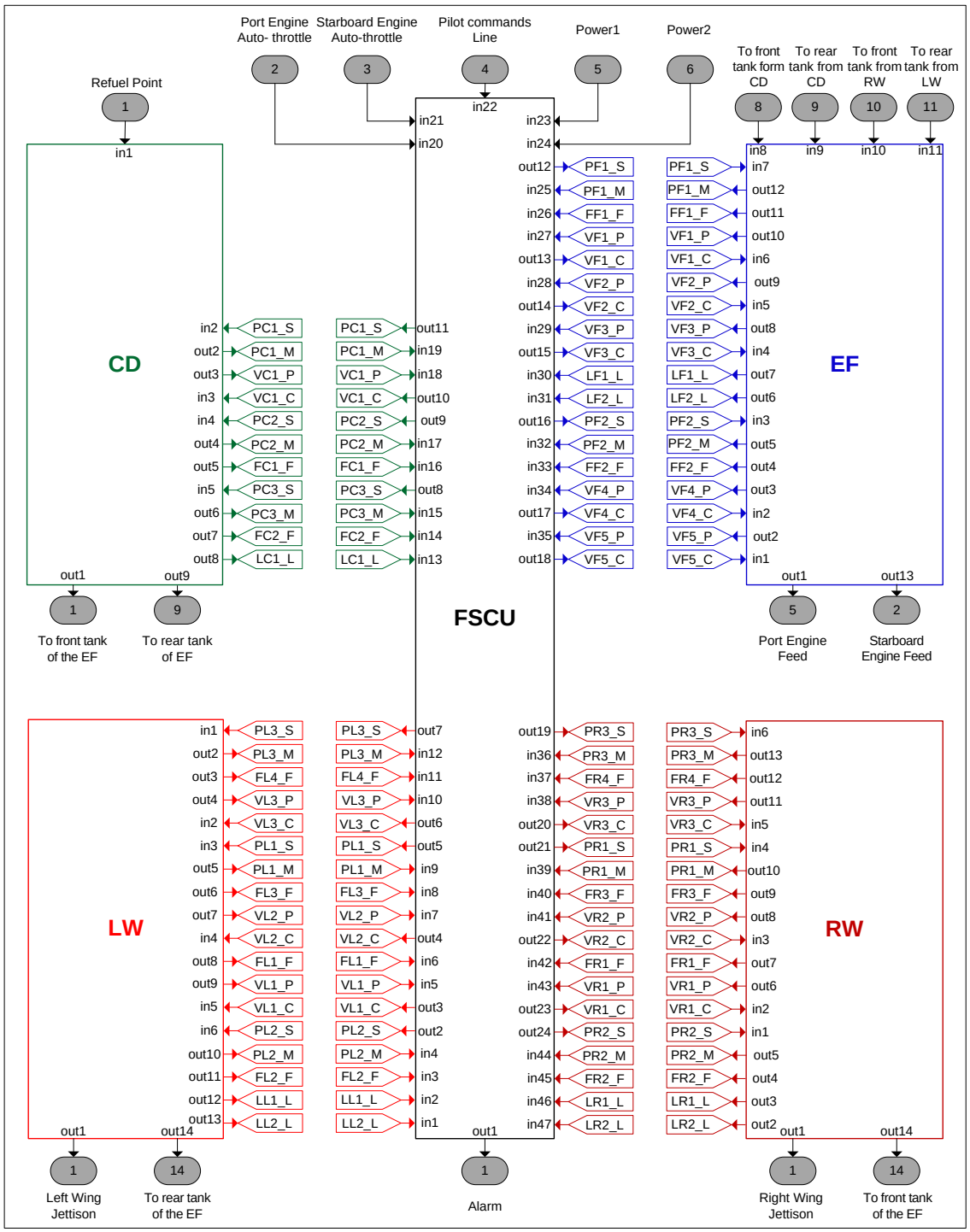


Figure 5-3: Level*1* of the Hierarchical Architecture of the AFS as a Simulink Model**.**

Figure 5-3 also shows how the FSCU and the four sub-systems interface. The four sub-systems, engine feed (EF), central deposit (CD), left wing (LW) and right wing (RW), deliver sensory measurements to the fuel system control unite (FSCU), which in turn instructs them by actuating commands. They also deliver fuel to each other until it reaches the starboard and port

engines. Figure 5-4 abstracts, as E-BNF notation, the grammar used to name both sensory measurements and actuating commands.

interface	::= sensory_measurement | actuating_command;
sensory_measurement	::= basic_component_initial, sub-system_initial, number, "_", measurement_initial;
basic_component_initial	::= "P" | "V" | "F" | "L";
sub-system_initial	::= "E" | "C" | "L" | "R";
measurement_initial	::= "P" | "L" | "F" | "M";
number	::= digit | digit;
digit	::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
actuating_command	::= basic_component_initial, sub-system_initial, number, "_", command_initial;
command_initial	::= "C" | "S";

where

P	::= An initial of a pump;
V	::= An initial of a valve;
F	::= When it appears as the first letter in the interface name, it is then an initial for the flow meter. Otherwise, it is an initial of a flow rate measurement;
L	::= When it appears as the first letter in the interface name, it is then an initial of the level sensor. Its appearance as the second letter in the interface name means it is an initial of the left wing subsystem. Otherwise it is an initial of a fuel level measurement;
E	::= An initial of the engine feed sub-system;
C	::= When it appears as the second letter in the interface name, it is then an initial of the central deposit sub-system. Otherwise, it is an initial of actuating commands either to open or close a valve;
R	::= An initial of the right wing sub-system;
P	::= A measurement of a valve position, which is either open and equals "1" or closed and equals "0";
M	::= A speed measurement of a pump;
S	::= An initial of actuating commands either to increase or drop the speed of a pump;

Figure 5-4: E-BNF of the Sensory Measurements and Actuating Commands of the AFS.

The interface is either a sensory measurement or an actuating interface. Every sensory measurement consists of a two-part name separated by an underscore. The first part consists of the initials of a basic component and a sub-system and a number, while the second part is the initial of a measurement. Similarly, every actuating command consists of a two-part name separated by an underscore. The initials of a basic component and a sub-system and a number compose the first part and a command initial constitutes the second part. The syntax of the initials is also explained in the E-BNF. The letters L, F, P and C may appear more than once in the same interface and they are accordingly defined according to the positions in which they appear.

Figure 5-5 illustrates the basic components level (level*0*) of the fuel system as a Simulink model. In this illustration, the basic components of the FSCU have been omitted for the sake of brevity and also because the FSCU is a reliable component and not susceptible to failure. The fuel system is typically provided with primary and backup FSCUs and each is powered by redundant power lines (power1 and power2). Despite the omission, the interfaces between the FSCU and the four sub-systems appear in the illustration.

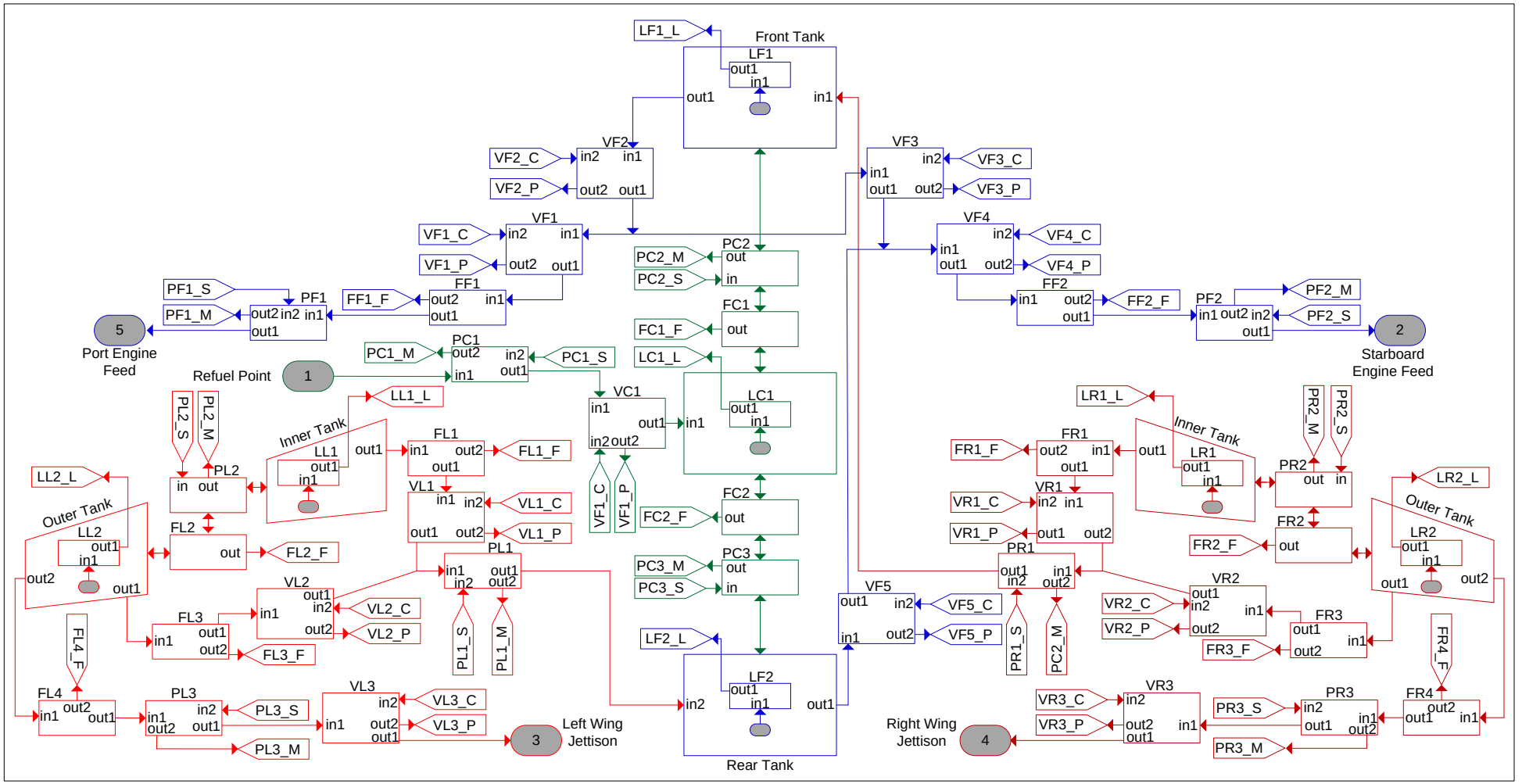


Figure 5-5: Level*0* of the Hierarchical Architecture of the AFS as a Simulink Model.

5.2.2 Operational Behaviour of the Aircraft Fuel System

Over the flight phases, the behaviour of the fuel system can be abstracted in two modes; the refuelling and consumption modes. While the refuelling mode applies during the pre-flying phase, the consumption mode applies in the taxiing, take-off, climbing, cruising, approaching and landing phases.

During the consumption mode[1], the FSCU receives demands to deliver variable fuel rate from the auto-throttles of the two engines; the rate is always 2R. The FSCU, in turn, applies controlling instructions that deliver the intended rate and maintain an even level across the seven tanks. Figure 5-6 illustrates the flow directions and flow rates under the normal conditions of the consumption mode.

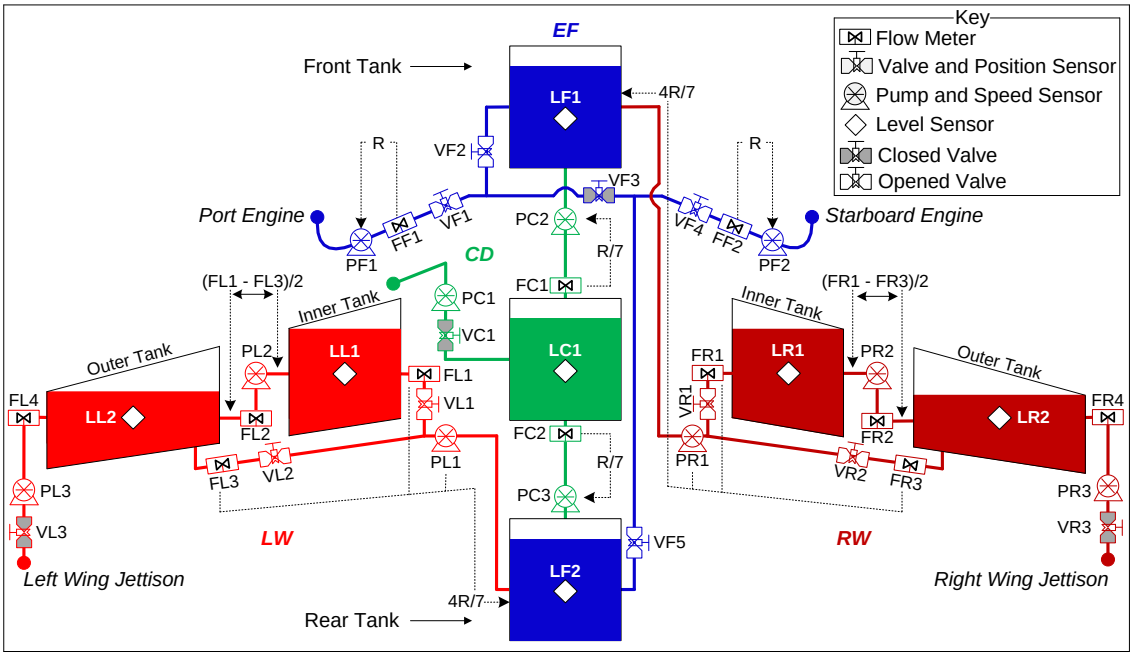


Figure 5-6: Configuration, Flow Directions and Rates of the Consumption mode of the AFS.

Table 5-1, moreover, details the contribution and flow rates to and from each tank. Together, Figure 5-6 and Table 5-1 show how the pumps and flow meters work in different closed-control loops to draw fuel at the corresponding rate from each tank. Two loops can be seen in the LW sub-system; the first between pump PL1 and the two flow meters, FL3 and FL1, and the second between pump PL2 and flow meter FL2. While the first loop draws fuel from the inner and outer tanks equal to the LW portion (4R/7), the second maintains an even fuel level between those tanks. If the fuel level of the inner tank (LL1_L) is more than that of the outer tank

[1] Taking the importance of convenient illustration as well as brevity into consideration, the effectiveness of the proposed monitor is demonstrated and examined during the consumption mode of the AFS.

(LL2_L) then PL2 should draw the difference from the inner tank to the outer tank and vice versa. Two similar loops can also be seen in the RW sub-system.

There are also two control loops in the EF sub-system, the first between pump PF2 and flow meter FF2 and the second between pump PF1 and flow meter FF1. In both loops, the speed of the pumps is regulated according to the fuel volume measured by the flow meters to draw fuel equal to R demanded by each engine. Similarly, there are two loops in the CD sub-system, the first between pump PC2 and flow meter FC1 and the second between pump PC3 and flow meter FC2. These together draw fuel equal to the flow portion of the CD tank, which equals 2R/7 (R/7 for each loop).

Table 5-1: Detailed Flow Rate of each Fuel Tank of the AFS.

Tank	**Tank contribution**	**Inlet**		**Outlet**	
		Rate	**From**	**Rate**	**To**
Front	2R/7	4R/7	RW Tanks	R	Port Engine
		R/7	CD Tank		
Rear	2R/7	4R/7	LW Tanks	R	Starboard Engine
		R/7	CD Tank		
CD	2R/7	None	None	R/7	Front Tank
				R/7	Rear Tank
LW inner	2R/7	(LL2_L − LL1_L)/2 *When* LL2_L > LL1_L	Outer of LW	(LL1_L − LL2_L)/2 *When* LL1_L > LL2_L	Outer of LW
				2R/7	Rear Tank
LW outer	2R/7	(LL1_L − LL2_L)/2 *When* LL1_L > LL2_L	Inner of LW	(LL2_L − LL1_L)/2 *When* LL2_L > LL1_L	Inner of LW
				2R/7	Rear Tank
RW inner	2R7	(LR2_L − LR1_L)/2 *When* LR2_L > LR1_L	Outer of RW	(LR1_L − LR2_L)/2 *When* LR1_L > LR2_L	Outer of RW
				2R/7	Front Tank
RW outer	2R/7	(LR1_L − LR2_L)/2 *When* LR1_L > LR2_L	Inner of RW	(LR2_L − LR1_L)/2 *When* LR2_L > LR1_L	Inner of RW
				2R/7	Front Tank

The maximum deliverable rate of the entire fuel system is in the region of two litres per minute (2 L/M); each engine may demand a maximum of 1 L/M. The rate varies according to the

demanded thrust of the engines. Every rate change is followed by an unsteady interval in which fuel flow and tank levels fluctuate.

Figure 5-7 illustrates the approximate intervals for bringing the entire fuel system into steadiness after changing the flow rates. The illustration considers thrust increasing for achieving a step climb[1] during the cruising phase and also decreasing the thrust for achieving efficient cruising. It is assumed that the current flow rate is 40%, the step climb requires another 40% and when the desired altitude is reached the thrust is reduced to 30%.

Whenever the rate changes, an interval of maximum 6 seconds fluctuation follows. Over the first three seconds the flow rates fluctuate outside the demanded rate, in the next three seconds the fluctuation is close to the demanded range, within a range of less than ± 3% and thereafter steadiness is reached. The illustration also shows that the same fluctuation occurs with both the increase and decrease of the flow rate.

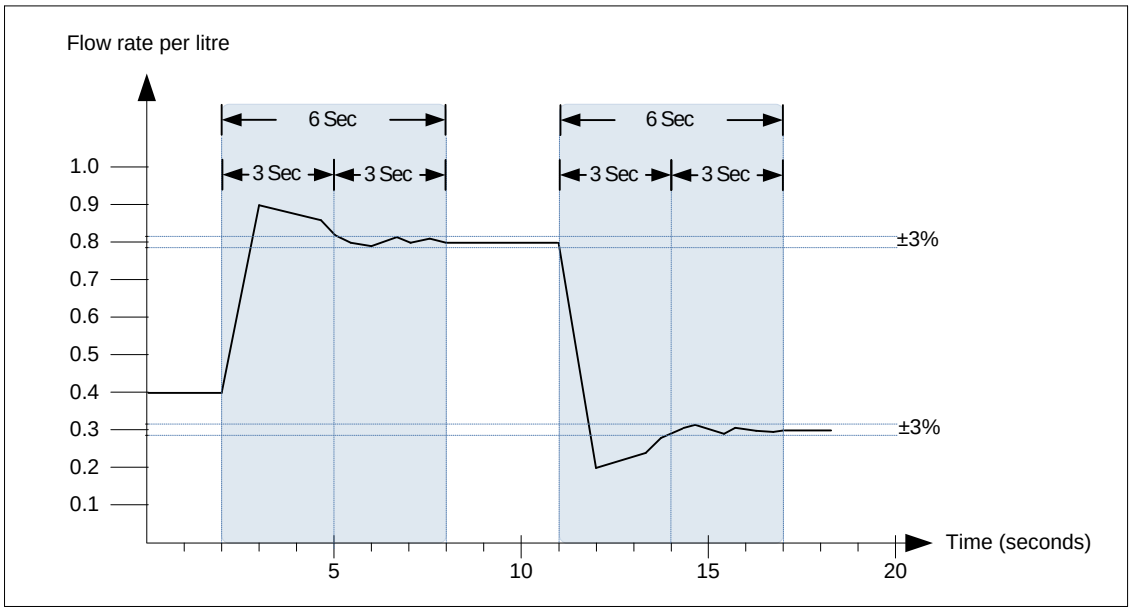


Figure 5-7: Fluctuation Follows Changing the Flow Rate of the AFS.

Over the different flow rates and from full to empty tanks, levels across the seven tanks would be approximately even, although small variation may result from potential bias and accumulative inaccuracy of the sensors. Figure 5-8 illustrates a typical pattern of accumulative discrepancy between the levels of the front and rear tanks across five different rates. The total discrepancy is approximately 6%.

[1] Step climb is a flying action taken during the cruise phase to gain the best altitude at which the aircraft becomes lighter and fuel consumption is, consequently, reduced.

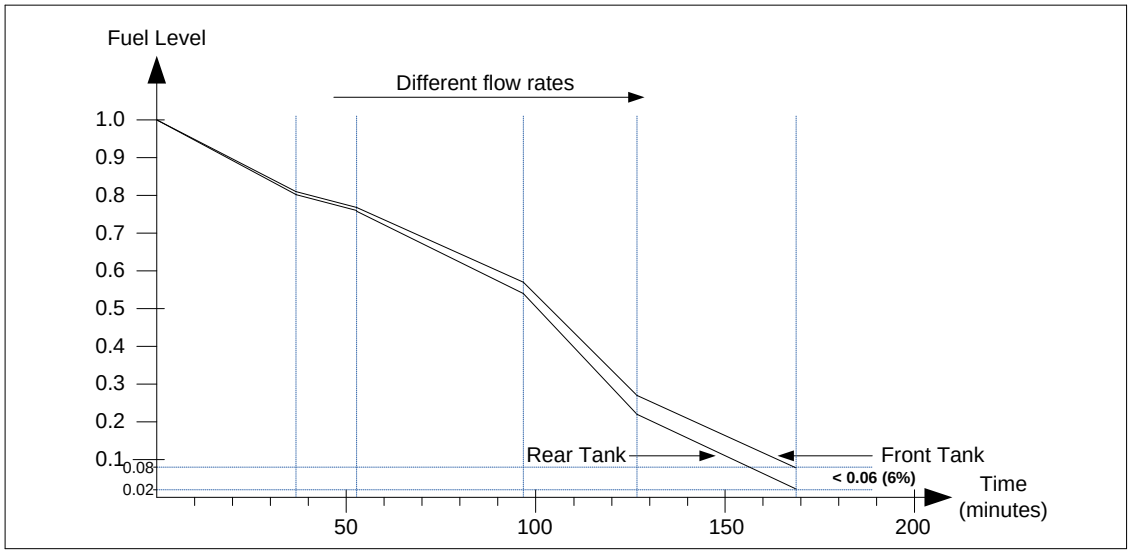


Figure 5-8: Discrepancy of Fuel Levels of Two Tanks from Full to Empty Status of AFS.

5.3 Monitoring Agents Deployment

Five monitoring agents are deployed to monitor the fuel system. One agent monitors the entire fuel system, termed aircraft fuel system monitoring agent (AFS_MAG). The other four are deployed to monitor the four sub-systems and they are termed engine feed monitoring agent (EF_MAG), central deposit monitoring agent (CD_MAG), left wing monitoring agent (LW_MAG) and right wing monitoring agent (RW_MAG). Figure 5-9 shows the hierarchical deployment and the collaboration protocols[1] of the deployed agents.

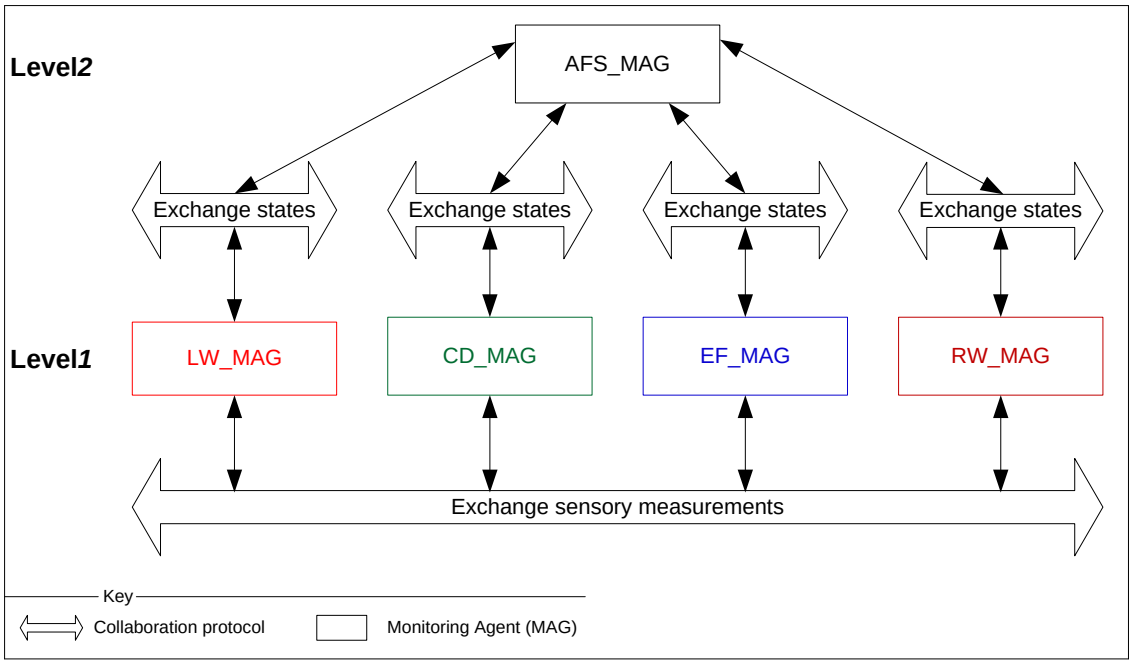


Figure 5-9: The Hierarchical Deployment Model of the Multi-agent System over the AFS.

[1] The figure is illustrated according to the Prometheus methodology for developing intelligent agent systems (Padgham and Winikoff, 2004).

5.4 HiP-HOPS Model and the Distributed Monitoring Model

This section develops HiP-HOPS models of the fuel system and its sub-systems. From those models and according to the derivation and formalisation approach developed in section 3.6, the monitoring knowledge is then derived and formalised as a distributed monitoring model. The full model is too large to be described here. However, excerpts of the model are given in the following sub-sections to enable description of monitoring experiments.

5.4.1 HiP-HOPS Model and the Monitoring Model of the Aircraft Fuel System (AFS)

Figure 5-10 shows an excerpt of the state-machine of fuel system. The behaviour of the system is shown as seven different states. Five belong to the consumption mode: one error-free state, AFS_CM_EFS; three permanent degraded states, AFS_CM_PDS1, AFS_CM_PDS2 and AFS_CM_PDS3, and a failure state, AFS_CM_FS. The sixth and seventh are a permanent degraded state (AFS_RM_PDS3) and failure state (AFS_RM_FS) that belong to the refuel mode. Transitions among the states are triggered by the states of the four sub-systems.

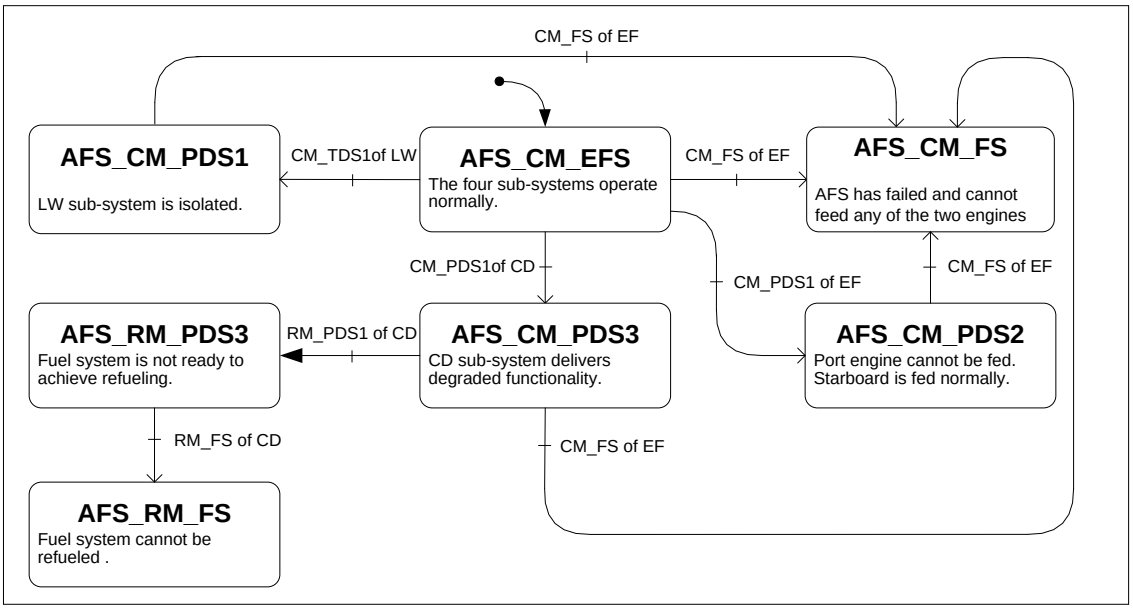


Figure 5-10: An Excerpt of the State-machine of the AFS (Top Level State-machine).

According to the developed derivation and formalisation approach, the derived state-transition table of the AFS is as shown by Table 5-2. In Table 5-2 every trigger event is formalised as a monitoring expression, as follows:

- Events that represent exit from state AFS_CM_EFS are:

  CM_FS of EF → EF_CM_FS == true;
  CM_TDS1 of LW → LW_CM_TDS1 == true;

CM_PDS1 of EF → EF_CM_PDS1 == true;

CM_PDS1 of CD → CD_CM_PDS1 == true;

- The event that represents exit from state AFS_CM_PDS1 is:

  CM_FS of EF → EF_CM_FS == true;

- The event that represents exit from state AFS_CM_PDS2 is:

  CM_FS of EF → EF_CM_FS == true;

- The event that represents exit from state AFS_CM_PDS3 are:

  RM_PDS1 of CD → CD_RM_PDS1 == true;
  CM_FS of EF → EF_CM_FS == true;

- The event that represents exit from state AFS_RM_PDS3 is:

  RM_FS of CD → CD_RM_FS == true;

Table 5-2: State-transition Table of the AFS.

CURRENT STATE	CONDITIONS	EVENT	NEW STATE
AFS_CM_EFS	**Assessment**: the four sub-systems operate normally. **Guidance**: none.	EF_CM_FS == true	AFS_CM_FS
		LW_CM_TDS1 == true	AFS_CM_PDS1
		EF_CM_PDS1 == true	AFS_CM_PDS2
		CD_CM_PDS1 == true	AFS_CM_PDS3
AFS_CM_PDS1	**Assessment**: LW sub-system is isolated. **Guidance**: none.	EF_CM_FS == true	AFS_CM_FS
AFS_CM_PDS2	**Assessment**: port engine cannot be fed, whereas starboard engine is feeding normally **Guidance**: none.	EF_CM_FS == true	AFS_CM_FS
AFS_CM_PDS3	**Assessment**: CD sub-system delivers degraded functionality. **Guidance**: none.	CD_RM_PDS1 == true	AFS_RM_PDS3
AFS_RM_PDS3	**Assessment**: fuel system is not ready to achieve refuelling. **Guidance**: none.	CD_RM_FS == true	AFS_RM_FS
AFS_CM_FS	**Assessment**: AFS has failed and cannot feed any of the two engines. **Guidance**: none	none	none

5.4.2 HiP-HOPS Model and the Monitoring Model of the Engine Feed (EF) Sub-system

Figure 5-11 illustrates an excerpt of the state-machine of the EF sub-system. The behaviour of the EF is shown as six different states, all belonging to the consumption mode. States are as follows: (a) two error-free states EF_CM_EFS1 and EF_CM_EFS2; (b) two temporary degraded or failure states EF_CM_TDFS1 and EF_CM_TDFS2; (c) a permanent degraded state EFS_CM_PDS1; (d) a failure state EFS_CM_FS. Some states are triggered by the state transitions of the AFS (CM_PDS1 of AFS).

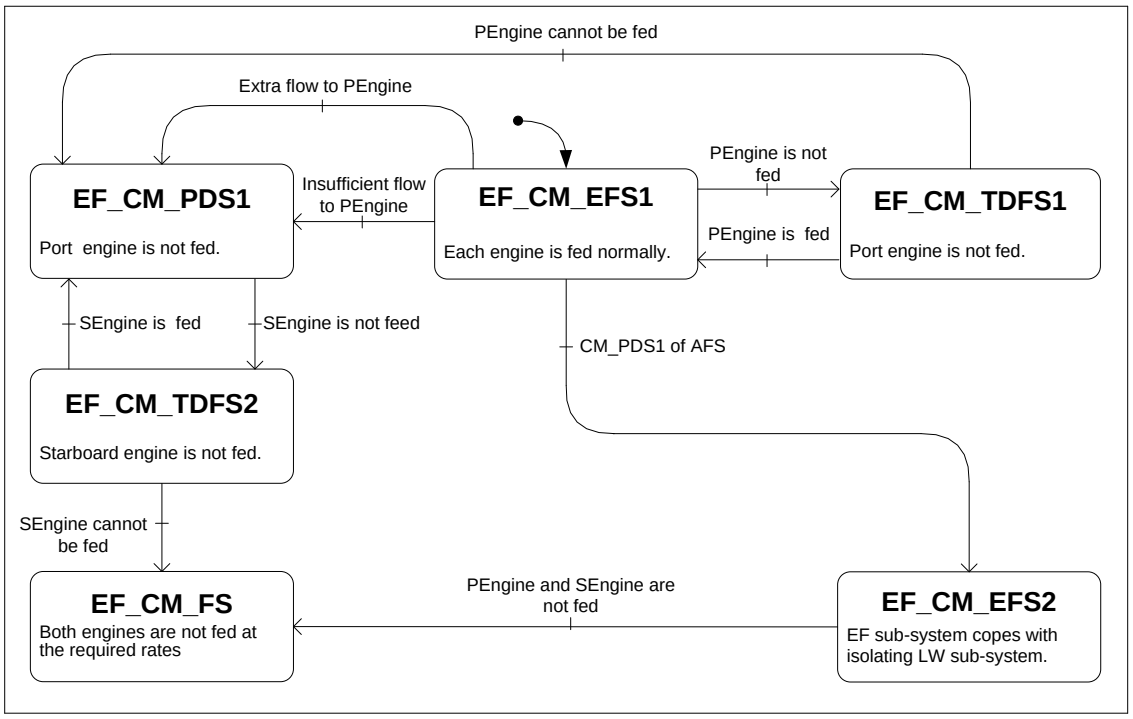


Figure 5-11: An Excerpt of the State-machine of the EF Sub-system.

The state-transition table of the EF is as shown in Table 5-3. Every trigger event is formalised as a monitoring expression as follows:

- Events that represent exit from state EF_CM_EFS1 are:

  PEngine is not fed[1] → T(|FF1_F| < 0.03, 4 sec);
  Extra flow to PEngine[2] → T(FF1_F > R + 0.03, 6 sec);
  Insufficient flow to PEngine → T(FF1_F < R – 0.03, 6 sec);
  CM_PDS1 of AFS → AFS_CM_PDS1 == true;

[1] It is assumed that the flow meter FF1 has a possible bias as a maximum range of - 0.029 to + 0.029. Accordingly, if the delivered measurement is less than 0.03 then that represents no flow. The same bias is assumed applicable for the other flow meters. One should not forget, though, that sensors of such systems have auto-calibration ability which makes bias less likely but also possible with few measurements.

[2] This threshold has been defined according to Figure 5-7.

- Events that represent exit from state EF_CM_TDFS1 are:

  PEngine cannot be fed → T(|FF1_F| < 0.03, 4 sec);
  PEngine is fed → T(|FF1_F – R| < 0.03, 4 sec);

- The event that represents exit from state EF_CM_PDS1 is:

  SEngine is not fed → T(|FF2_F| < 0.03, 4 sec);

- The event that represents exit from state EF_CM_EFS2 is:

  PEngine and SEngine are not fed → T(|FF1_F| < 0.03 AND |FF2_F| < 0.03 , 4 sec);

- Events that represent exits from state EF_CM_TDFS2 are:

  SEngine cannot be fed → T(|FF2_F| < 0.03, 4 sec);
  SEngine is fed → T(|FF2_F – R| < 0.03, 4 sec);

Table 5-3: State-transition Table of the EF Sub-system.

CURRENT STATE	CONDITIONS	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
EF_CM_EFS1	**Assessment:** each engines is fed normally. **Guidance:** none	**T(|**FF1_F**| <** 0.03, 4 sec);	port engine is not fed	after_diagnosis	needed	EF_CM_TDFS1
		T(FF1_F > R + 0.03, 6 sec);	port engine fed with extra rate	- PF1_S = 0; - VF1_C = 0;	not_needed	EF_CM_PDS1
		T(FF1_F < R – 0.03, 6 sec);	port engine is fed with insufficient rate.	- PF1_S = 0; - VF1_C = 0;	not_needed	EF_CM_PDS1
		AFS_CM_PDS1 **==** true;	none	none	not_needed	EF_CM_EFS2
EF_CM_TDFS1	**Assessment:** port engine is not fed and recovery is in progress. **Guidance:** watch for further feedback.	**T(|**FF1_F**| <** 0.03, 4 sec);	feeding port engine cannot be recovered.	- PF1_S = 0; - VF1_C = 0; - VF2_C = 0;	not_needed	EF_CM_PDS1
		T(|FF1_F - R**| <** 0.03, 4 sec);	none	none	not_needed	EF_CM_EFS1
EF_CM_PDS1	**Assessment:** port engine is not fed. **Guidance:** none.	**T(|**FF2_F**| <** 0.03, 4 sec);	starboard engine is not fed	after_diagnosis	needed	EF_CM_TDFS2
EF_CM_EFS2	**Assessment:** EF sub-system copes with isolating LW sub-system. **Guidance: n**one.	**T**(|FF1_F| < 0.03 **AND** |FF2_F| < 0.03 , 4 sec);	Both engines are not fed	impossible	needed	EF_CM_FS
EF_CM_TDFS2	**Assessment:** starboard engine is not fed and recovery is in progress. **Guidance:** watch for further feedback.	**T(|**FF2_F**| <** 0.03, 4 sec);	feeding starboard engine cannot be recovered.	impssible	not_needed	EF_CM_FS
		T(|FF2_F - R**| <** 0.03, 4 sec);	none	none	not_needed	EF_CM_PDS1
EF_CM_FS	**Assessment:** both engines cannot be fed. **Guidance:** none.	none	none	none	not_needed	none

Figure 5-12 shows the fault tree of the failure event "PEngine is not fed". According to the developed grammar to derive a diagnostic model from a fault tree (Figure 3-18), Figure 5-13 shows the diagnostic model derived from the fault tree of Figure 5-12. In Figure 5-13 the top node encloses the expression of the failure event that is monitored at level*1* (EF sub-system), which is formalised in the Table 5-3 as T(|FF1_F| < 0.03, 4 sec).

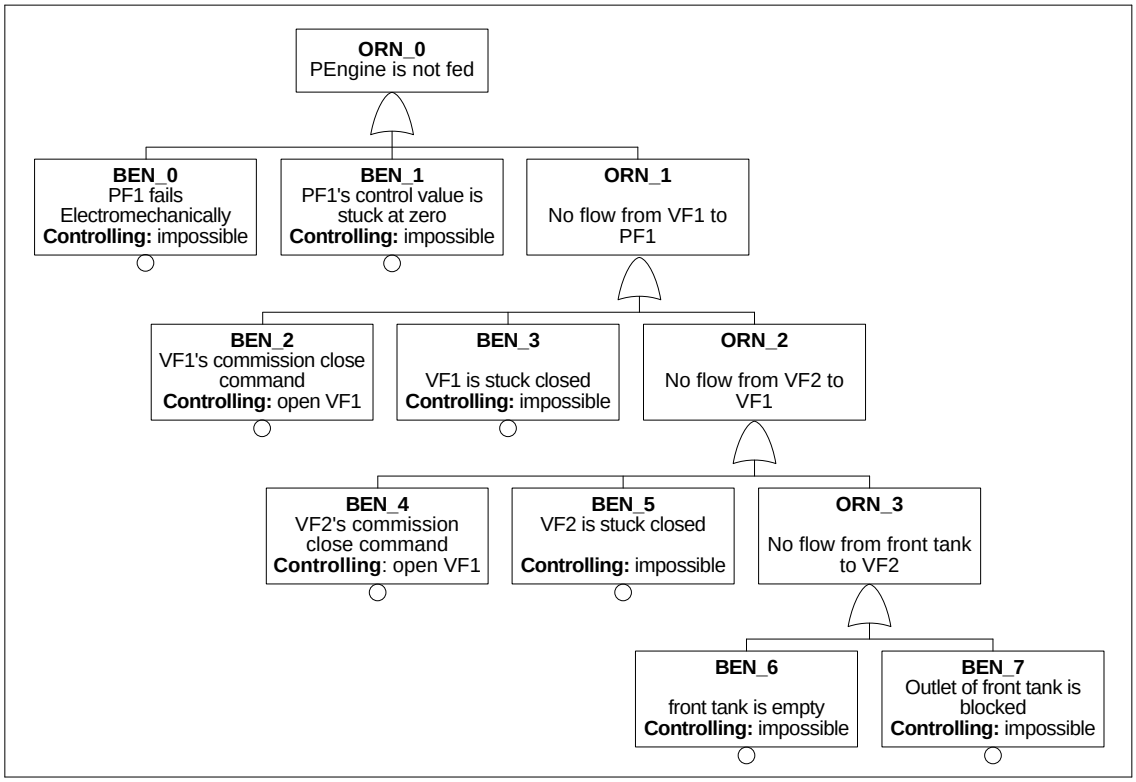


Figure 5-12: A Fault Tree of Failure Event "PEngine is not fed".

In Figure 5-13 symptoms of the fault tree (Figure 5-12) have been formalised as monitoring expressions as follows:

- The monitoring expression of the symptoms "PF1 fails Electromechanically" and "PF1's control value is stuck at zero" is:

$$|PF1\_M| <= 20\ Rpm^{1}$$

- The monitoring expression of the symptoms "VF1's commission close command" and "VF1 is stuck closed" is:

$$VF1\_P == 0$$

- The monitoring expression of the symptoms "VF2's commission close command" and "VF2 is stuck closed" is:

[1] 20 Rpm is the assumed possible bias of the speed sensor of the fuel pumps.

VF2_P == 0

- The monitoring expression of the symptom “outlet of the front tank is blocked” is:

(VF2_P ==1) AND (VF1_P==1) AND (PF1_M > 20 Rpm)

- The monitoring expression of the symptom “front tank is empty” is:

LF1_L < 0.1

NodeName: ORN_0.
Symptom: T(|FF1_F**|** < 0.03, 4 sec).
ChildName: BEN_0.

NodeName: BEN_0.
Symptom: |PF1_M**|** <= 20.
Fault: PF1 fails electromechanically.
Controlling: none.
Sibling: BEN_1.

NodeName: BEN_1.
Symptom: |PF1_M**| <=** 20.
Fault: PF1's control value is stuck at zero.
Controlling: none.
Sibling: ORN_1.

NodeName: ORN_1.
Symptom: none.
Child: BEN_2.
Sibling: none.

NodeName: BEN_2.
Symptom: VF1_P **==** 0.
Fault: VF1's commission close command.
Controlling: VF1_C = 1.
Sibling: BEN_3.

NodeName: BEN_3.
Symptom: VF1_P **==** 0.
Fault: VF1 is stuck closed.
Controlling: none.
Sibling: ORN_2.

NodeName: ORN_2.
Symptom: none.
Child: BEN_4.
Sibling: none.

NodeName: BEN_4.
Symptom:VF2_P **==** 0.
Fault: VF2's commission close command.
Controlling: VF2_C = 1.
Sibling: BEN_5.

NodeName: BEN_5.
Symptom: VF2_P == 0.
Fault: VF2 is stuck closed.
Controlling: none.
Sibling: ORN_3.

NodeName: ORN_3.
Symptom: none.
Child: BEN_6.
Sibling: none.

NodeName: BEN_6.
Symptom:(VF2_P ==1) **AND** (VF1_P==1) **AND** (PF1_M > 20).
Fault: front tank outlet is blocked.
Controlling: none.
Sibling: BEN_7.

NodeName: BEN_7.
Symptom: LF1_L **<** 0.1.
Fault: front tank is empty.
Controlling: none.
Sibling: none.

Figure 5-13: Formal Diagnostic Form of Fault Tree Shown by Figure 5-12.

5.4.3 HiP-HOPS Model and the Monitoring Model of the Central Deposit (CD) Sub-system

Figure 5-14 illustrates an excerpt of the state-machine of the CD sub-system. The figure shows seven different states as follows:

- Five states belong to consumption mode: three error-free states CD_CM_EFS1, CD_CM_EFS2 and CD_CM_EFS3, a permanent degraded state CD_CM_PDS1 and a failure state CD_CM_FS.
- Two states belong to refuelling mode a permanent degraded state CD_RM_PDS and a failure state CD_RM_FS.

The incorporation of states of refuelling mode in this excerpt facilitates some aspects of the experimental demonstration and evaluation of timely prognosis; as will be seen in section 5.5.3.2.

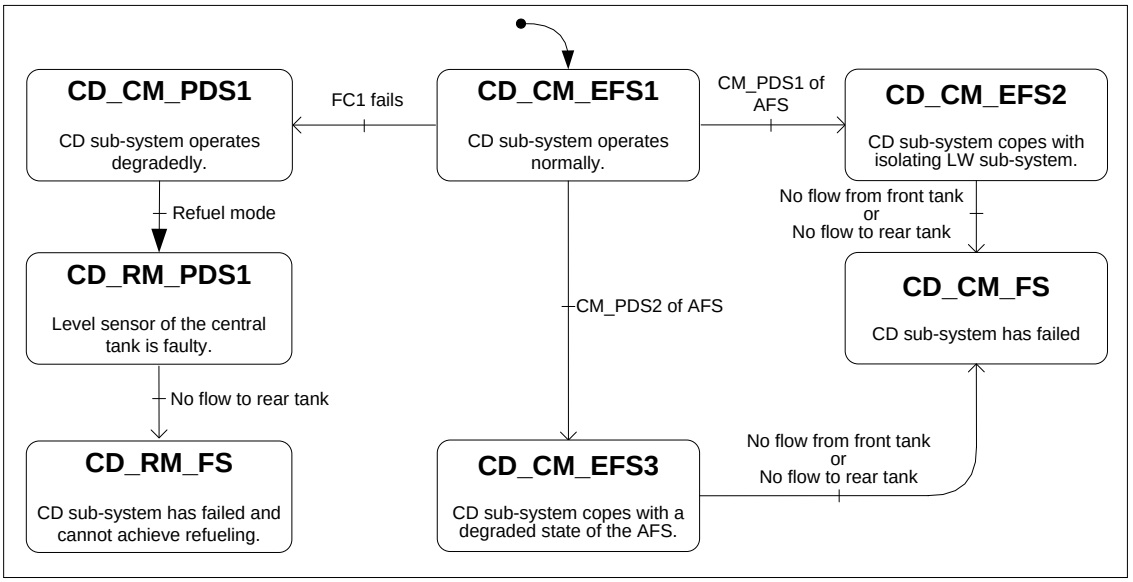


Figure 5-14: An Excerpt of State-machine of the EF Sub-system.

Table 5-4 shows the state-transition table derived from Figure 5-14. Every trigger event is formalised as a monitoring expression as follows:

- Events that represent exit from state CD_CM_EFS1 are:

FC1 fails[1] →(T(FC1_F > R/7 + 0.03, 6 sec) AND T(FC1_F > FC2_F + 0.03, 6 sec))
OR
(T(FC1_F < R/7 - 0.03, 6 sec) AND T(FC1_F < FC2_F - 0.03, 6 sec));

[1] R/7 is the volume of fuel that should flow through flow meter FC1, see also Table 5-1.

CM_PDS1 of AFS → AFS_CM_PDS1 == true;

CM_PDS2 of AFS → AFS_CM_PDS2 == true;

- The event that represents exit from state CD_CM_PDS1 is:

  Refuel mode → VC1_P == 1 AND VF1_P == 0 AND VF4_P == 0;

- The event that represents exit from state CD_RM_PDS1 is:

  No flow to rear tank → T(|FC2_F| < 0.03, 4 sec);

- The event that represents exit from state CD_CM_EFS2 is:

  No flow from front tank or no flow to rear tank → T(|FC1_F| < 0.03 OR |FC2_F| < 0.03, 4 sec);

- The event that represents exit from state CD_CM_EFS3 is:

  No flow from front tank or no flow to rear tank → T(|FC1_F| < 0.03 OR |FC2_F| < 0.03, 4 sec);

Table 5-4: State-transition Table of the CD Sub-system.

CURRENT STATE	CONDITIONS	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
CD_CM_EFS1	**Assessment:** CD sub-system operates normally. **Guidance:** none	(**T**(FC1_F > R/7 + 0.03, 6 sec) **AND** **T**(FC1_F > FC2_F + 0.03, 6 sec)) **OR** (**T**(FC1_F < R/7 - 0.03, 6 sec) **AND** **T**(FC1_F < FC2_F - 0.03, 6 sec));	CD sub-system has a sensory failure.	- FC1_F= FC2_F;	Sensor FC1 has failed.	CD_CM_PDS1
		AFS_CM_PDS1 **==** true;	none	- FC1_F = -R/5; - FC2_F = 3R/5;	not_needed	CD_CM_EFS2
		AFS_CM_PDS2 **==** true;	none	- FC1_F = -3R/7; - FC2_F = 4R/7;	not_needed	CD_CM_EFS3
CD_CM_PDS1	**Assessment:** CD sub-system operates degradedly. **Guidance:** none.	VC1_P **==** 1 **AND** VF1_P **==** 0 **AND** VF4_P **==** 0;	none	not_needed	not_needed	CD_RM_PDS1
CD_RM_PDS1	**Assessment:** CD sub-system operates degradedly. **Guidance:** flow meter FC1 must be replaced.	**T**(|FC2_F| < 0.03, 4 sec);	rear tank is not refueling.	- PC1_S = 0; - VF1_C = 0;	needed	CD_RM_FS
CD_CM_EFS2	**Assessment:** CD sub-system copes with isolating LW sub-system. **Guidance:** none.	**T**(|FC1_F| < 0.03 **OR** |FC2_F| < 0.03, 4 sec);	abnormal flow from the central tank.	- PC2_S = 0; - PC3_S = 0;	needed	CD_CM_FS
CD_CM_EFS3	**Assessment:** CD sub-system copes with a degraded state of the AFS **Guidance:** none.	**T**(|FC1_F| < 0.03 **OR** |FC2_F| < 0.03, 4 sec);	abnormal flow from the central tank.	- PC2_S = 0; - PC3_S = 0;	needed	CD_CM_FS

5.4.4 HiP-HOPS Model and the Monitoring Model of Left Wing (LW) Sub-system

Figure 5-15 illustrates an excerpt of the state-machine of the LW sub-system. The figure shows four different states, all of which belong to the consumption mode and are as follows:

- Two error-free states LW_CM_EFS1 and LW_CM_EFS2.
- A temporary degraded state LW_CM_TDS1.
- A failure state LW_CM_FS2.

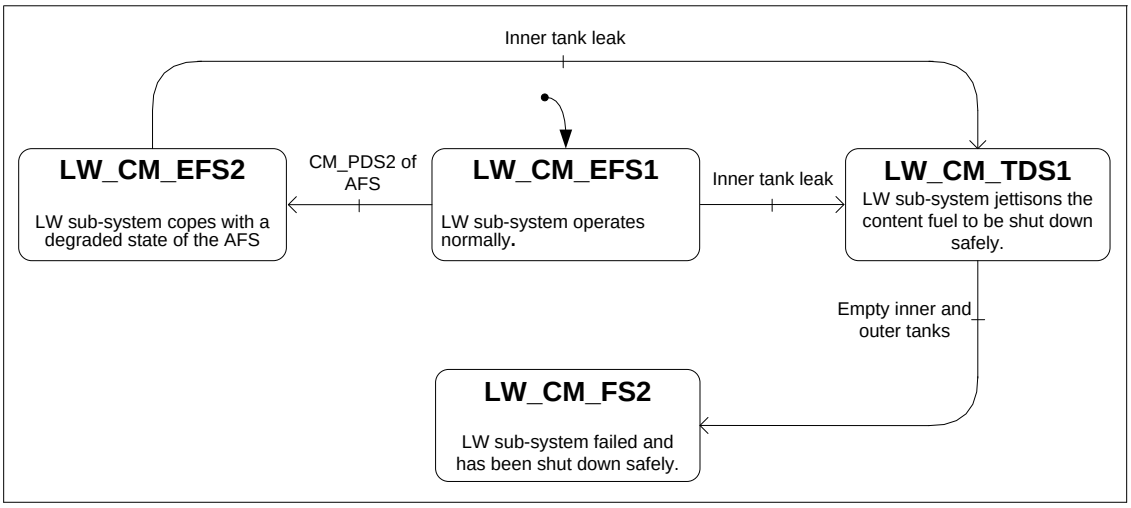


Figure 5-15: An Excerpt of State-machine of the LW Sub-system.

Table 5-5 shows the state-transition table derived from Figure 5-15. Every trigger event is formalised as a monitoring expression as follows:

- Events that represent exits from state LW_CM_EFS1 are:

  CM_PDS2 of AFS → AFS_CM_PDS2 == true;
  Inner tank leak[1] → V(LL1_L, 5 sec) > I(FL1_F + FL2_F, 5 sec) + 0.06;

- The event that represents exit from state LW_CM_EFS2 is:

  Inner tank leak → V(LL1_L, 5 sec) > I(FL1_F + FL2_F, 5 sec) + 0.06;

- The event that represents exit from state LW_CM_TDS1 is:

  Inner and outer tanks are empty → LL1_L < 0.1 AND LL2_L < 0.1;

[1] Where 0.06 is the accumulative bias of the two sensors: FL1 and FL2.

Table 5-5: State-transition Table of the LW Sub-system.

CURRENT STATE	CONDITIONS	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
LW_CM_EFS1	**Assessment:** LW sub-system operates normally. **Guidance:** none	AFS_CM_PDS2 == true;	none	- FL3_F = R/7; - FL1_F = R/7;	not_needed	LW_CM_EFS2
		V(LL1_L, 5 sec) > I(FL1_F + FL2_F, 5 sec) + 0.06;	inner tank of LW sub-system is leaky	- PL1_S = 0; - VL1_C = 0; - VL2_C = 0; - VL3_C = 1; - FL2_F = - 0.285; - FL4_F = 0.571;	leak in the inner tank of LW.	LW_CM_TDS1
LW_CM_EFS2	**Assessment:** LW sub-system copes with a degraded state of the AFS. **Guidance:** none	V(LL1_L, 5 sec) > I(FL1_F + FL2_F, 5 sec) + 0.06;	inner tank of LW sub-system is leaky	- PL1_S = 0; - VL1_C = 0; - VL2_C = 0; - VL3_C = 1; - FL2_F = - 0.285; - FL4_F = 0.571;	leak in the inner tank of LW.	LW_CM_TDS1
LW_CM_TDS1	**Assessment:** LW sub-system jettisons the content fuel to be shut down safely. **Guidance:** none	LL1_L **<** 0.1 **AND** LL2_L **<** 0.1;	none	- VL3_C = 0; - PL3_S = 0;	not_needed	LW_CM_FS2
LW_CM_FS2	**Assessment:** LW sub-system failed and has been shut down safely. **Guidance:** none	none	none	none	none	none

5.4.5 HiP-HOPS Model and the Monitoring Model of the Right Wing (RW) Sub-system

Figure 5-16 illustrates an excerpt of the state-machine of the LW sub-system. The figure shows five different states, all of which belong to the consumption mode and are as follows:

- Three error-free states RW_CM_EFS1, RW_CM_EFS2 and RW_CM_EFS3.
- A temporary degraded state RW_CM_TDS1.
- A failure state RW_CM_FS2.

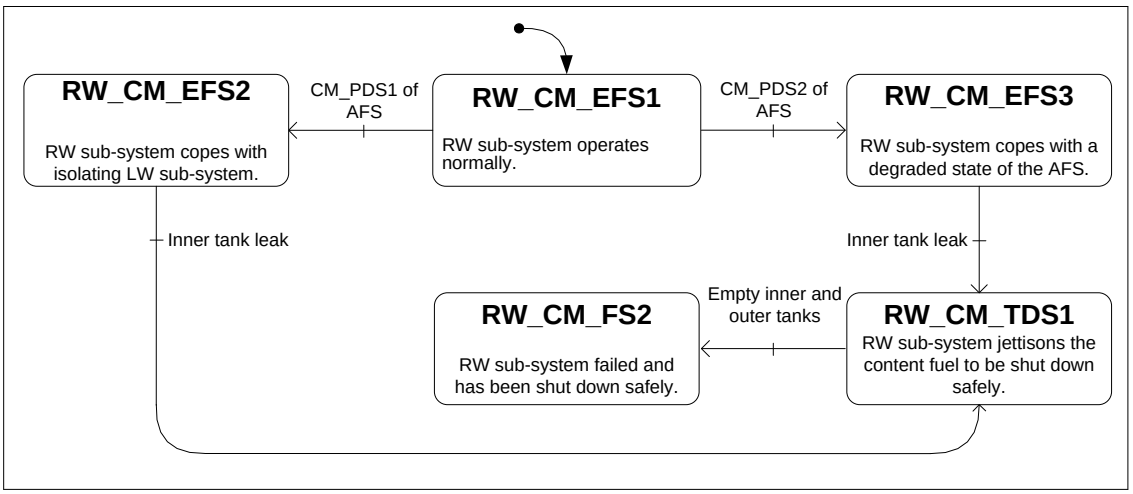


Figure 5-16: An Excerpt of State-machine of the RW Sub-system.

Table 5-6 shows the formal state-transition table derived from Figure 5-16. Every trigger event is formalised as a monitoring expression as follows:

- Events that represent exits from state RW_CM_EFS1 are:

  CM_PDS1 of AFS → AFS_CM_PDS1 == true;
  CM_PDS2 of AFS → AFS_CM_PDS2 == true;

- The event that represents exit from state RW_CM_EFS2 is:

  Inner tank leak → V(LR1_L, 5 sec) > I(FR1_F + FR2_F, 5 sec) + 0.06;

- The event that represents exit from state RW_CM_TDS1 is:

  Inner and outer tanks are empty → LR1_L < 0.1 AND LR2_L < 0.1;

Table 5-6: State-transition Table of the RW Sub-system.

CURRENT STATE	CONDITIONS	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
RW_CM_EFS1	**Assessment:** RW sub-system operates normally. **Guidance:** none	AFS_CM_PDS1 == true;	none	- FR1_F = 2R/5; - FR3_F = 2R/5;	not_needed	RW_CM_EFS2
		AFS_CM_PDS2 == true;	none	- FR1_F = R/7; - FR3_F = R/7;	not_needed	RW_CM_EFS3
RW_CM_EFS2	**Assessment:** RW sub-system copes with isolating LW sub-system. **Guidance:** none	V(LR1_L, 5 sec) > I(FR1_F + FR2_F, 5 sec) + 0.06;	inner tank of RW sub-system is leaky	- PR1_S = 0; - VR1_C = 0; - VR2_C = 0; - VR3_C = 1; - FR2_F =- 0.285; - FR4_F = 0.571;	not_needed	RW_CM_TDS1
RW_CM_EFS3	**Assessment:** RW sub-system copes with a degraded state of the AFS. **Guidance:** none	V(LR1_L, 5 sec) > I(FR1_F + FR2_F, 5 sec) + 0.06;	inner tank of RW sub-system is leaky	- PR1_S = 0; - VR1_C = 0; - VR2_C = 0; - VR3_C = 1; - FR2_F =- 0.285; - FR4_F = 0.571;	not_needed	RW_CM_TDS1
RW_CM_TDS1	**Assessment:** RW sub-system jettisons the content fuel to be shut down safely. **Guidance:** none	LR1_L < 0.1 **AND** LR2_L < 0.1;	none	none	not_needed	RW_CM_FS2
RW_CM_FS2	**Assessment:** RW sub-system failed and has been shut down safely. **Guidance:** none	none	none	none	none	none

5.5 Experiment

In order to test the monitor and evaluate its ability to deliver the intended tasks, a simulator has been developed to:

- simulate the normal conditions of every sub-system and the entire aircraft fuel system in both steady and variable feeding rates;

- inject possible faults of the basic components and simulate their effects on the behaviour of the sub-systems and the entire fuel system;
- simulate conditions following application of corrective measures.

During the consumption mode and under normal conditions, the simulator draws fuel from the seven tanks according to the demanded rate (R) of each engine, as shown by Figure 5-6 and Table 5-1. For example, to maintain the normal inlet and outlet of the front tank, the simulator achieves the following scenario: (a) drawing fuel from the CD tank equal to R/7 and from the RW deposit[1] equal to 4R/7 to be added to the front tank; (b) drawing fuel equal to R from the front tank[2]; R is the rate demanded by the port engine.

To simulate different demanded flow rates of the two engines, the simulator has the ability to generate variable demanded rates, e.g. increasing the thrust of port engine results in increasing the speed of pump PF1, as FF1 and PF1 are in the same closed-control loop. As that speed is positively correlated with the measurements of flow meters FF1_F, FR1_F, FR3_F and FC1_F, then the speed of other relevant pumps is increased to draw the corresponding portions of the new rate (from the RW deposit and CD tank). Further information on the calculation of these portions can be found in Table 5-1.

Abnormal conditions, an interruption of the flow to the port engine, for instance, can be generated by injecting a fault of pump PF1 or inadvertent closure of valve VF1 or valve VF2. After injecting a fault, the simulator stops drawing fuel from the front tank while maintaining the same flow from the CD tank and RW deposit. As such, level sensor LF1 will indicate that the fuel level of the front tank is increasing abnormally. Furthermore, the measurement of flow meter (FF1_F) will be close to zero, indicating that the port engine is not being fed. From either of these sensors, the agent EF_MAG can detect and deal with the injected fault.

Whether the fault is injected by a fault of pump PF1 or inadvertent closure of valve VF1 or valve VF2, the simulator would also provide the relevant symptoms with corresponding sensory measurements. When fuel is interrupted by injecting a fault of PF1, the generated measurements of the speed sensor of pump PF1 range between -20 and 20 Rpm. In a different case, when the fault is injected as a commission close command of valve VF1 or valve VF2, the simulator continues to generate a normal range of speed measurements of PF1, generates a closed value (0) for the faulty valve and maintains the other valve open (1).

To simulate application of corrective measures, the simulator can also be instructed by the agents to apply measures and generate the corresponding measurements. For example,

[1] Deposit means the volume of fuel in the inner and outer tanks.
[2] This is calculated by taking the total of 2R/7 of the front tank itself, 4R/7 delivered by the RW deposit and R/7 delivered by the CD sub-system.

commission close command of valve VF2 can be controlled by instructing the simulator to reopen VF2.

To facilitate the evaluation of the response time and the prompt delivery of the monitoring tasks, each monitoring agent is provided with an ability to count (a) the time that is taken to achieve a monitoring cycle; (b) the interval that falls between the injection of a fault and announcing the corresponding alarm; (c) the interval that falls between the injection of a fault and diagnosing the underlying causes; (d) the interval that falls between the fault injection and fault controlling.

Table 5-7 lists faults that are injected by the developed simulator to demonstrate the ability of the monitor to deliver the intended monitoring tasks and also to evaluate the effectiveness of those tasks. The table shows a number of possible faults of every component of the fuel system. The injection of these faults results in verifying the occurrence of different failure events and simulating different failure scenario, such as fuel flow interrupting, fuel level imbalance and fuel leak.

Table 5-7: Faults that are injected by the Developed Simulator.

Component	Injected Faults	Description
Valve	Commission close command	A valve is closed inadvertently and can be instructed to open.
	Commission open command	A valve is opened inadvertently and can be instructed to close.
	Stuck between open and close	A valve sticks half open and cannot be controlled.
	Stuck closed	A valve sticks closed and does not respond to open instructions.
	Stuck open	A valve sticks open and does not respond to close instructions.
Tank	Structural leak.	Tanks of the fuel system are susceptible to structural leak. Controlling such failure can be achieved by isolating the affected deposit and jettisoning the isolated fuel to the air.
Sensor	Permanent functional failure of a sensor.	A sensor delivers permanent spurious measurements that lie out of the expected range or delivers non-linear drift or biased measurements.
Pump	Non-stop	A pump does not respond to stop instructions.
	Non-start	A pump does not respond to start instructions.
	Opposite pumping	A pump functions reversely.
	Non-regulate	A pump does not respond to variable speed instructions.
	Fail and stop	A pump stops pumping fuel because the control value sticks at zero or due to an electromechanical failure.
	Speed sensor bias	A speed sensor biases and delivers inaccurate measurements.

Simulations of three different failure scenarios have been selected to be discussed herein: interrupting fuel flow to the port engine “PEngine is not Fed”, structural leak of the inner tank

of the LW sub-system and sensory failure (flow meter FC1 of the CD sub-system). Along with simulating these scenarios the delivery of the monitoring tasks is also evaluated based on the effectiveness of (a) detecting, diagnosing and controlling faults; (b) the design of the operators' interface and also the announced alarm information. In the experimental context, spurious sensory measurements, which may result as consequence of white Gaussian noise and behavioural transitions, have also been simulated to test the capacity of the developed filtration technique (see section 3.6.1.2).

5.5.1 First Injected Fault: Interrupting Fuel Flow to the Port Engine

The occurrence of "PEngine is not Fed" can be verified by the following time expression:

$$| FF1\_F| < 0.03 \text{ for } 4 \text{ sec}$$

In Table 5-3, this expression has been presented as T(|FF1_F| < 0.03, 4 sec). Once the monitoring agent EF_MAG evaluates this expression with true, it perceives the state-transition table (Table 5-3) and achieves the following procedure:

- From the relevant ALARM attribute, agent FE_MAG quotes the statement "port engine is not fed" and alarms the pilot.
- From the relevant CONTROLLING attribute, agent FE_MAG checks the possibility of controlling that event. As the controlling depends on the underlying cause, that attribute accordingly tells the EF_MAG to achieve a diagnostic process by traversing the relevant fault tree ("after_diagnosis").
- From the relevant DIAGNOSIS attribute, agent FE_MAG verifies the need for a diagnostic process and updates the symptoms of the diagnostic model, Figure 5-13.
- From the relevant NEW STATE attribute, agent FE_MAG transits to the new state, which is the temporary degraded or failure state EF_CM_TDFS1. From this state the pilot is provided with the assessment, "port engine is not fed and recovery is in progress" and guidance, "watch for further feedback".
- Agent EF_MAG also communicates the current state to the parent agent (AFS_MAG). The state does not trigger a state transition in the state-transition table of the AFS_MAG.

Since a diagnostic process is needed, agent EF_MAG retrieves the position of the top node of the relevant fault tree and launches a diagnostic process before launching a monitoring cycle for the new state EF_CM_TDFS1, as shown by plan L_1-P_{17} and plan L_1-P_2 of section 4.7.2. By traversing the relevant diagnostic model (Figure 5-13) the underlying cause was diagnosed and the required corrective measures were taken. Assuming that the cause is "PF1 fails electromechanically" or "PF1's control value stuck at zero", whether the cause is the former or

the latter fault, controlling is not possible and thus agent EF_MAG perceives Table 5-3 and achieves the following procedure:

- Launches a monitoring cycle to the active events of state EF_CM_TDFS1.
- During this cycle the occurrence of T(|FF1_F| < 0.03, 4 sec) is verified consequently.
- From the relevant ALARM attribute, agent EF_MAG quotes and annunciates an alarm of "feeding port engine cannot be recovered".
- From the relevant CONTROLLING attribute, agent EF_MAG takes the following actions: switching pump PF1 off and closing valves VF1 and VF2.
- As the diagnostic process appears not to be needed with this event, agent EF_MAG moves accordingly to the NEW STATE attribute, identifies and transits to a new state which is EF_CM_PDS1. From this state the pilot is provided with assessment as "port engine is not fed" and guidance, "none".
- Agent EF_MAG also communicates EF_CM_PDS1 to the parent agent (AFS_MAG).

Feeding only one engine (starboard engine) requires changing the operational structure of the entire fuel system to maintain an even level across the seven tanks. Accordingly, the above procedure is not enough to control the fault; controlling these conditions requires global collaboration among the remaining three sub-systems: LW, RW, and CD.

Once agent AFS_MAG receives a message conveying state EF_CM_PDS1, it perceives the state-transition table (Table 5-2) and achieves the following procedure:

- While the current state is AFS_CM_EFS, the received state results in verifying the occurrence of EF_CM_PDS1 == true.
- From the relevant NEW STATE attribute, agent AFS_MAG transits to the new state which is the permanent degraded state AFS_CM_PDS2. From this state the pilot is provided with assessment as "port engine cannot be fed, whereas starboard engine is feeding normally" and guidance, "none".
- Agent AFS_MAG also communicates state AFS_CM_PDS2 to the child agents: CD_MAG, LW_MAG and LW_MAG.

Upon receiving messages conveying that state, each child agent achieves a certain fault controlling procedure to draw the corresponding flow rates and also transits to a new state. State transition and controlling procedures are as follows:

CD_MAG perceives the state-transition table (Table 5-4) and achieves the following procedure:

- While the current state is CD_CM_EFS1, the received state results in verifying the occurrence of AFC_CM_PDS2 == true.

- As the relevant ALARM attribute holds “none”, no alarm is thus announced.
- From the relevant CONTROLLING attribute, agent CD_MAG applies the following flow rates: FC1_F = -3R/7 and FC2_F = 4R/7.
- As the relevant DIAGNOSIS attribute holds “not_needed”, a diagnostic process is not launched.
- From the NEW STATE attribute, agent CD_MAG transits to the new state which is another error-free state CD_CM_EFS3. From this state the pilot is provided with assessment, “CD sub-system copes with a degraded state of the AFS” and guidance, “none”.

LW_MAG perceives the state-transition table (Table 5-5) and achieves the following procedure:

- While the current state is LW_CM_EFS1, the received state results in verifying the occurrence of AFC_CM_PDS2 == true.
- As the relevant ALARM attribute holds “none”, no alarm is thus announced.
- From the relevant CONTROLLING attribute, agent LW_MAG applies the following flow rates; FL3_F = R/7 and FL1_F = R/7.
- As the relevant DIAGNOSIS attribute holds “not_needed”, a diagnostic process is not launched.
- From the relevant NEW STATE attribute, agent LW_MAG transits to the new state which is another error-free state LW_CM_EFS2. From this state the pilot is provided with assessment, “LW sub-system copes with a degraded state of the AFS” and guidance “none”.

RW_MAG perceives the state-transition table (Table 5-6) and achieves the following procedure:

- While the current state is RW_CM_EFS1, the received state results in verifying the occurrence of AFC_CM_PDS2 == true.
- As the relevant ALARM attribute holds “none”, no alarm is thus announced.
- From the relevant CONTROLLING attribute, agent RW_MAG achieves the following flow rates: FR1_F = R/7 and FR3_F = R/7.
- As the relevant DIAGNOSIS attribute holds “not_needed”, a diagnostic process is not launched.
- From the NEW STATE attribute, agent RW_MAG transits to the new state which is another error-free state RW_CM_EFS3. From this state the pilot is provided with assessment, “RW sub-system copes with a degraded state of the AFS”, and guidance, “none”.

After achieving all the above procedures the fuel system would appear in a new operational structure to control event “port engine is not fed”. The new structure and the associated flow rates are as shown in Figure 5-17.

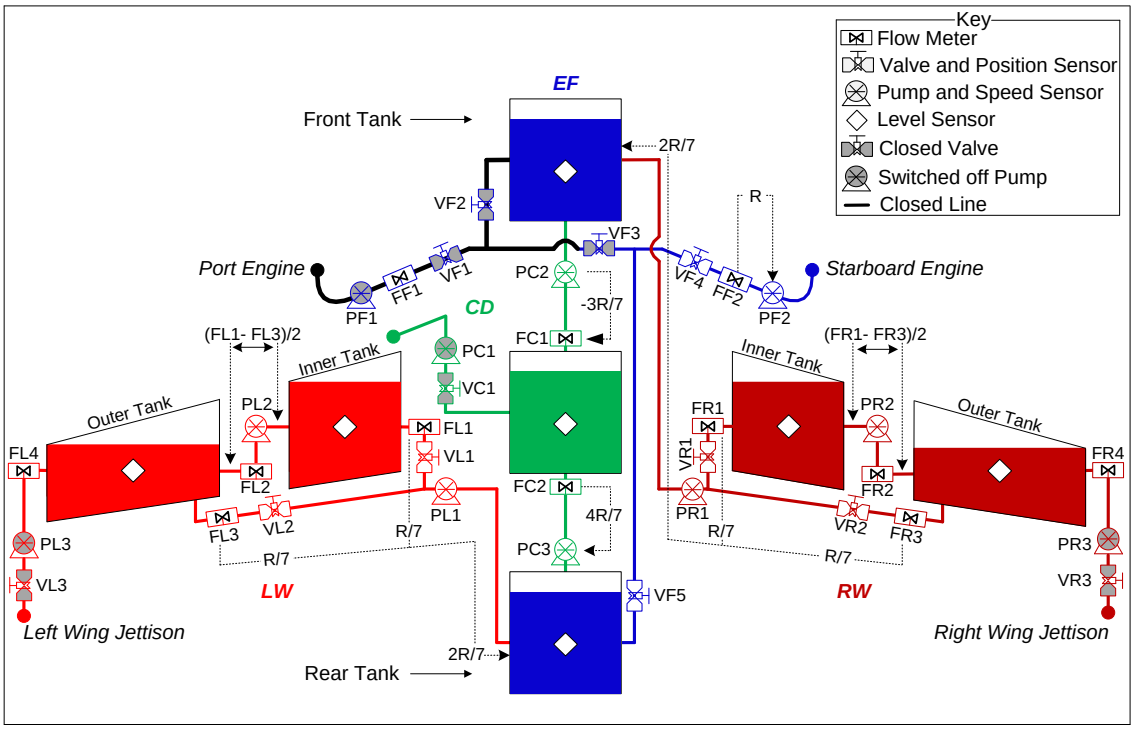


Figure 5-17: The Operational Structure of the AFS after Tolerating Failure; "PEngine is not fed".

The procedures and configuration will be different if the diagnosed cause is "VF1's commission close command". As shown in Figure 5-13, this cause is associated with a corrective measure to reopen valve VF1. Accordingly, agent EF_MAG instructs the fuel system control unit (FSCU) to reopen valve VF1 and goes back to launch a monitoring cycle of the active events of state EF_CM_TDFS1 (Table 5-3). If event T(|FF1_F| - R < 0.03, 4 sec) is verified true then agent EF_MAG resumes the error-free state EF_CM_EFS1, as that signifies the success of the applied measure in rectifying the conditions. From that state the operators would be provided with assessment and guidance that confirm the resumption of normal conditions. However, if that measure fails to rectify the conditions then event T(|FF1_F| < 0.03, 4 sec) will be verified true. In this case, agent EF_MAG achieves the corresponding state transition and repeats the above procedure; the procedure that is achieved when the PF1 is diagnosed as the underlying cause. The monitoring agents AFS_MAG, CD_MAG, LW_MAG and RW_MAG also achieve the same procedures.

5.5.1.1 Fault Detection and Diagnosis of the Occurrence of "PEngine is not fed"

Figure 5-18 shows the response of agent FE_MAG to detecting the failure, "PEngine is not fed". After injecting the fault, the total time that is taken to detect and annunciate alarm is close to 5 seconds. One second is taken to propagate the error while four seconds are taken to evaluate the relevant expression. After detecting the failure and alarming the operators, a diagnostic process is launched. The time taken to diagnose the underlying causes is 5.5 seconds.

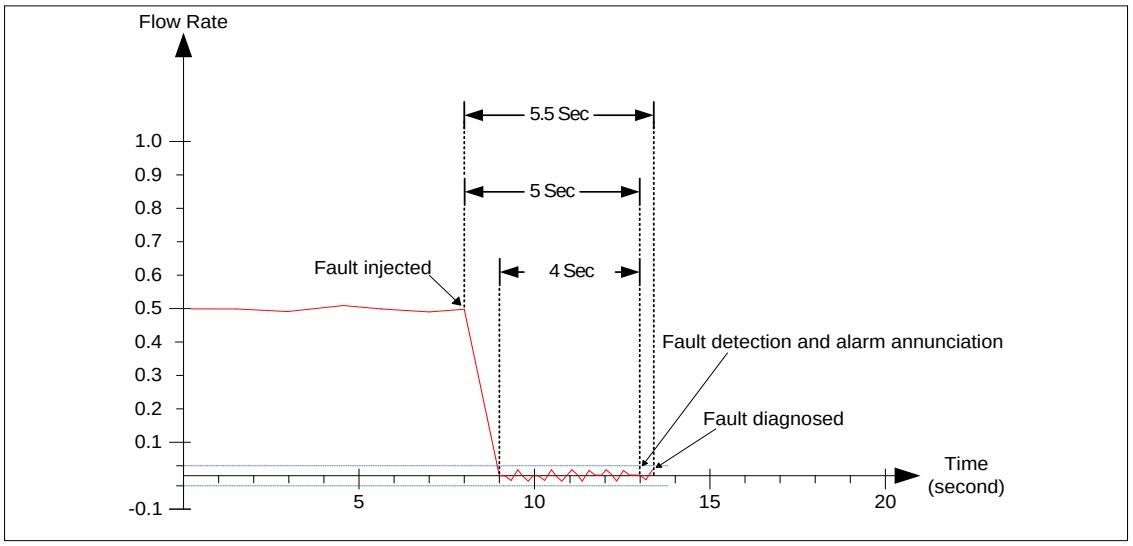


Figure 5-18: Monitor's Response in Detecting and Diagnosing "PEngine is not fed".

It can be seen how the monitoring agent of level*1* (EF_MAG) has detected and diagnosed a fault in an interval of 5.5 seconds. A large part of this interval is a result of verifying the event from a time expression that requires 4 seconds to filter the involved sensory measurements. In practice, the filtering interval can be eliminated and faster detection and diagnosis response can be obtained. However, such elimination allows the verification of the event from spurious measurements that do not reflect the actual conditions. Consequently, a false alarm is announced, wrong state transitions are achieved and incorrect and hazardous corrective measures are also applied. Therefore, despite the short delay added by the filtering interval, formalising time expressions to filter spurious measurements is necessary to avoid misleading the monitor and to ensure the delivery of effective monitoring tasks.

To evaluate the ability of the monitor to detect and diagnose multiple faults, the failure "PEngine is not fed" has also been simulated by the injection of electromechanical failure of pump PF1 and commission close command of valve VF2. The formalised monitoring expression has shown an ability to detect the faults and the diagnostic model and algorithm (Figure 3-22) have proved their effectiveness to diagnose the underlying causes.

5.5.1.2 Alarm Annunciation on the Occurrence of "PEngine is not Fed"

After verifying the occurrence of "PEngine is not fed", alarm, assessment, guidance and diagnostics are presented on the operators' interface as shown in Figure 5-19. It can be seen how the pilot is provided with an alarm on the occurrence of the failure event and assessment and guidance on the given conditions in simple and non-conflicting information. One can also see that the pilot is advised to watch for further feedback as fault controlling is in progress.

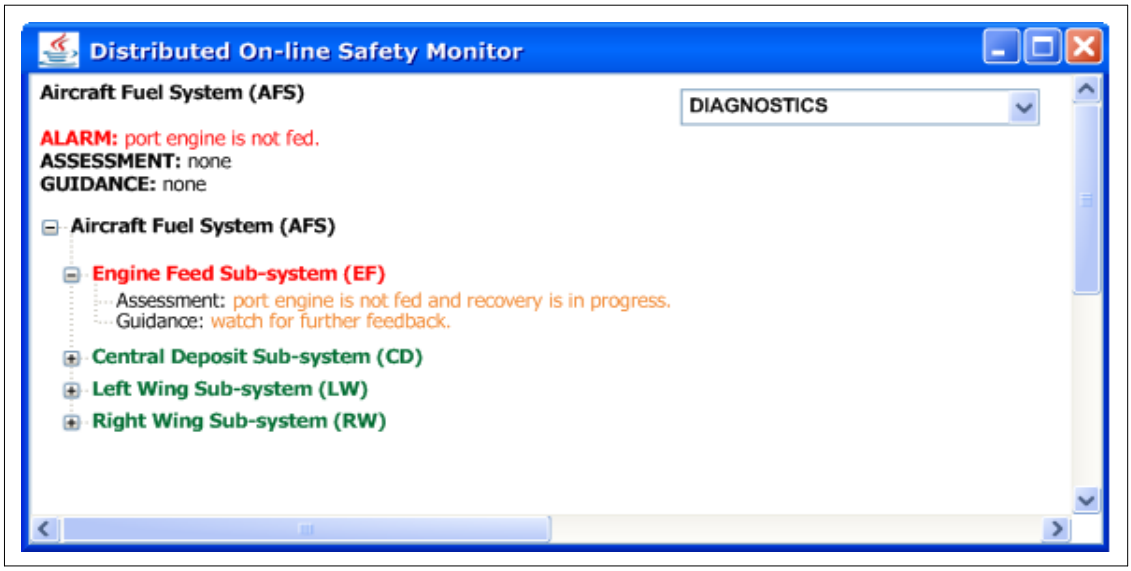


Figure 5-19: Operators' Interface after the Occurrence of "PEngine is not fed".

Such announced information seems enough to understand the actual conditions of the system. The announced information does not imply any need to involve actions that should be taken by the pilot, the case that can assure the avoidance of an incorrect interfering from the pilot and the exacerbation of the current emergency conditions. It can also be seen that the access of the diagnostics list is optional to abstract the presentation and avoid overwhelming the pilot.

5.5.1.3 Controlling the Occurrence of "PEngine is not fed"

The failure "PEngine is not fed" is controlled depending on the underlying cause. In the case in which "VF1's commission close command" is the underlying cause, a corrective measure is applied and normal conditions are resumed. Figure 5-20 illustrates the time that is taken to resume normal conditions after taking the corrective measure.

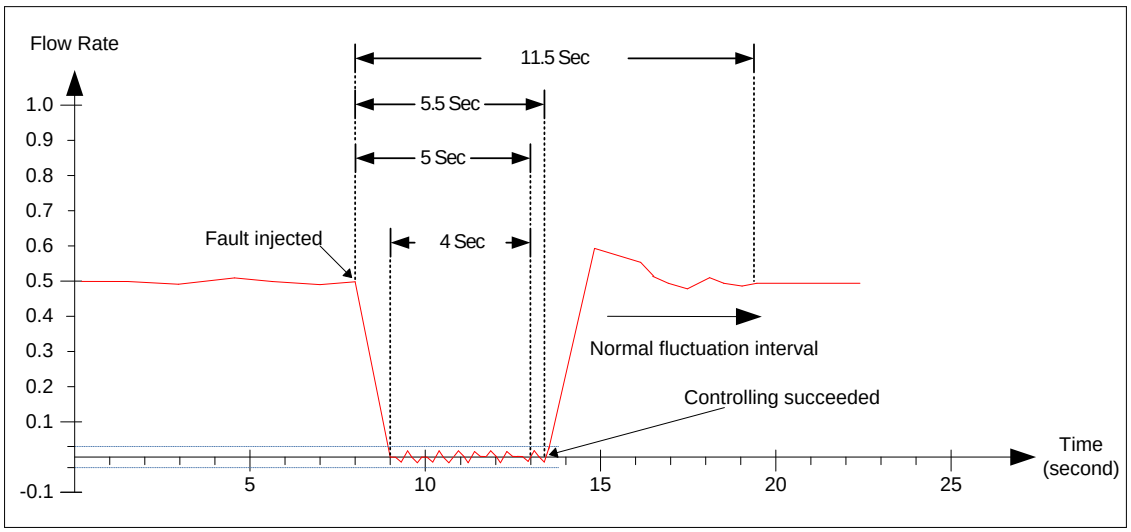


Figure 5-20: Monitor's Response in Controlling "PEngine is not fed".

The measures are taken after 5.5 seconds from detecting the fault and after five more seconds the success of the corrective measures is confirmed to the operators. The total time that is taken to bring the conditions into a steadiness is 11.5 seconds, including 6-second normal fluctuation (see also Figure 5-7). After controlling that fault, the operator's interface that is shown in Figure 5-20 is updated accordingly and appears as shown in Figure 5-21.

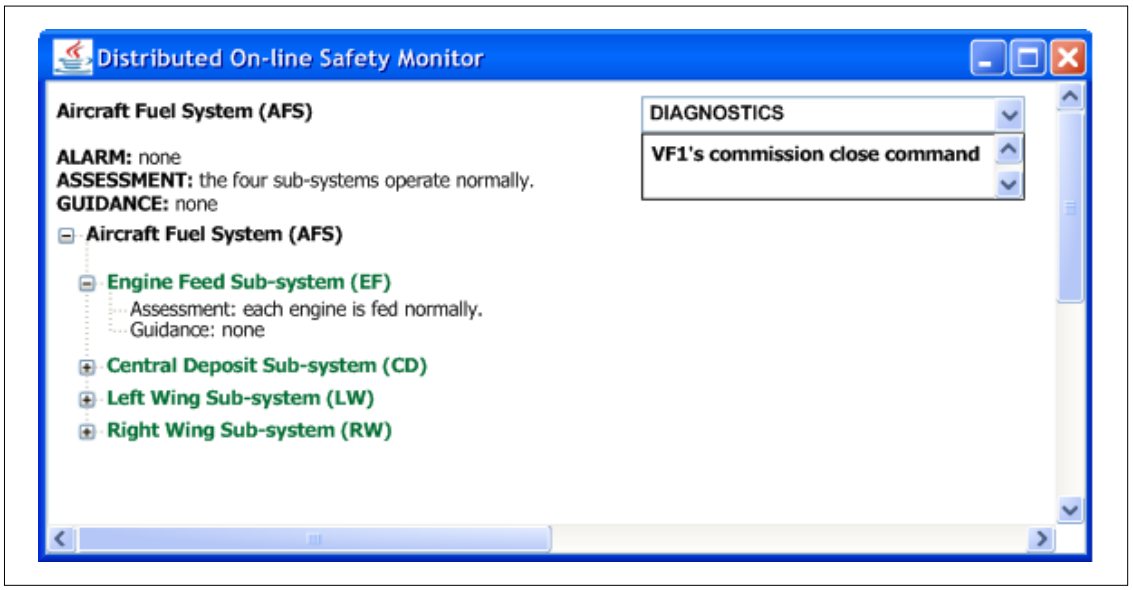


Figure 5-21: Operators' Interface after Controlling "PEngine is not fed".

It can be seen that the monitor has controlled the failure and rectified the operational conditions, which in turn are confirmed to the pilot by the updatable interface. Supported by the early fault detection and diagnosis, the incorporation of the corrective measures in the monitoring model and the updatable user interface, the fault controlling task is delivered effectively and controls the conditions automatically without adding extra load on the pilot.

5.5.2 Second Injected Fault: Structural Fuel Leak

Structural leak is among faults that are simulated during the demonstrative experiment. The fault has been injected in the inner tank of the LW sub-system. It is detected by the monitoring agent LW_MAG when the following expression is verified true:

$$\left(LL1\_L(T-5) - LL1\_L(T)\right) > \int_{T-5}^{T} \left(FL1\_F(t) + FL2\_F(t)\right) dt + 0.06$$

Where $\left(LL1\_L(T-5) - LL1\_L(T)\right)$: is the reduction of fuel level in the inner tank over an interval extending from T-5 in the past to current time T.

$\int_{T-5}^{T}(FL1\_F(t) + FL2\_F(t))\, dt$: is the total amount of fuel that has been (a) drawn from the inner tank by pump PL1 over an interval extending from T-5 in the past to current time T; (b) drawn or added by pump PL2 over the same interval. The interval is defined as 5 seconds as the shortest time to detect the structural leak.

0.06: is the maximum allowable discrepancy between the two above observations in normal conditions.

In Table 5-5 the above expression is presented as follows:

V(LL1_L, 5 sec) > I(FL1_F + FL2_F, 5 sec) + 0.06

Accordingly, once agent LW_MAG evaluates this expression with true, it perceives the state-transition table (Table 5-5) and launches the following procedure:

- From the relevant ALARM attribute, agent LW_MAG quotes and announces "inner tank of LW sub-system is leaky".
- From the relevant CONTROLLING attribute, agent LW_MAG achieves the following: (a) switch pump PL1 off; (b) close valves VL1 and VL2; (c) open valve VL3; (d) draw fuel from inner tank to outer tank equal to[1] -0.285 (FL2_F = -0.285); (e) switch pump PL3 on and draw fuel through jettison line equal to 0. 571 litre (FL4_F = 0.571).
- From the relevant DIAGNOSIS attribute, agent LW_MAG quotes "leak in the inner tank of the LW", as this event and its cause are in a one-to-one relationship.
- From the relevant NEW STATE attribute, agent LW_MAG transits to the new state, which is the permanent degraded state LW_CM_TDS1. From this state the pilot is provided with assessment, "LW subsystem jettisons the content fuel to be shut down safely" and guidance, "none".
- Agent LW_MAG communicates state LW_CM_TDS1 to the parent agent (AFS_MAG).

As shown in the above procedure, the deposit of LW sub-system is isolated and the two engines are currently fed by the EF, CD and RW sub-systems. To assure even fuel level among the remaining five tanks, these conditions cannot be controlled locally by the actions taken in the above procedure; they rather require collaboration among the other three sub-systems. Thus, changing the operational structure of the entire fuel system and applying new flow rates is necessary to tolerate the leak.

Once fuel system monitoring agent AFS_MAG receives a message that conveys state EF_CM_PDS1, it perceives the state-transition table (Table 5-2) and launches the following procedure:

[1] The negative sign is to instruct pump PL2 to draw fuel from the inner tank to the outer tank.

- While the current state is the AFS_CM_EFS, the received state results in verifying the occurrence of LW_CM_TDS1 == true.
- From the relevant NEW STATE attribute, agent AFS_MAG transits to the new state which is the permanent degraded state AFS_CM_PDS1. From this state the pilot is provided with assessment, “LW sub-system is isolated” and guidance, “none”.
- Agent AFS_MAG also communicates AFS_CM_PDS1 to the child agents; EF_MAG, CD_MAG, and RW_MAG.

When the child agents receive messages conveying that state, each of them achieves a certain fault controlling procedure to draw the corresponding flow rates and state transition. State transition and controlling procedures are as follows:

EF_MAG perceives the state-transition table (Table 5-3) and launches the following procedure:

- While the current state is EF_CM_EFS, the received state results in verifying the occurrence of AFS_CM_PDS1 == true.
- As the relevant ALARM attribute holds “none”, no alarm is thus announced.
- As the relevant CONTROLLING attribute holds “none”, no action is taken.
- As the relevant DIAGNOSIS attribute holds “not_needed”, then diagnosis is not launched.
- From the relevant NEW STATE attribute, agent EF_MAG identifies and transits to the new state, which is another error-free state EF_CM_EFS2. From this state the pilot is provided with assessment, “EF sub-system copes with isolating LW sub-system” and guidance, “none”.

Similarly, CD_MAG perceives the state-transition table (Table 5-4) and launches the following procedure:

- While the current state is CD_CM_EFS, the received state results in verifying the occurrence of AFS_CM_PDS1 == true.
- As the relevant ALARM attribute holds “none”, no alarm is thus announced.
- From the relevant CONTROLLING attribute, agent CD_MAG achieves the following rates: FC1_F = -R/5 and FC2_F = 3R/5.
- As the relevant DIAGNOSIS attribute holds “not_needed”, then diagnosis is not launched.
- From the relevant NEW STATE attribute, agent CD_MAG transits to the new state, which is another error-free state CD_CM_EFS2. From this state the pilot is provided with assessment, “CD sub-system copes with isolating LW sub-system” and guidance, “none”.

RW_MAG perceives the state-transition table (Table 5-6) and launches the following procedure:

- While the current state is RW_CM_EFS, the received state results in verifying the occurrence of AFS_CM_PDS1 == true.
- As the relevant ALARM attribute holds “none”, no alarm is thus announced.
- From the relevant CONTROLLING attribute, agent RW_MAG achieves the following rates FR1_F = 2R/5 and FR3_F = 2R/5.
- As the relevant DIAGNOSIS attribute holds “not_needed”, then diagnosis is not launched.
- From the relevant NEW STATE attribute, agent RW_MAG transits to the new state, which is another error-free state RW_CM_EFS2. From this state the pilot is provided with assessment, “RW sub-system copes with isolating LW sub-system” and guidance, “none”.

After achieving all the above procedures, the operational structure and the flow rates of the fuel system appear as shown by Figure 5-22.

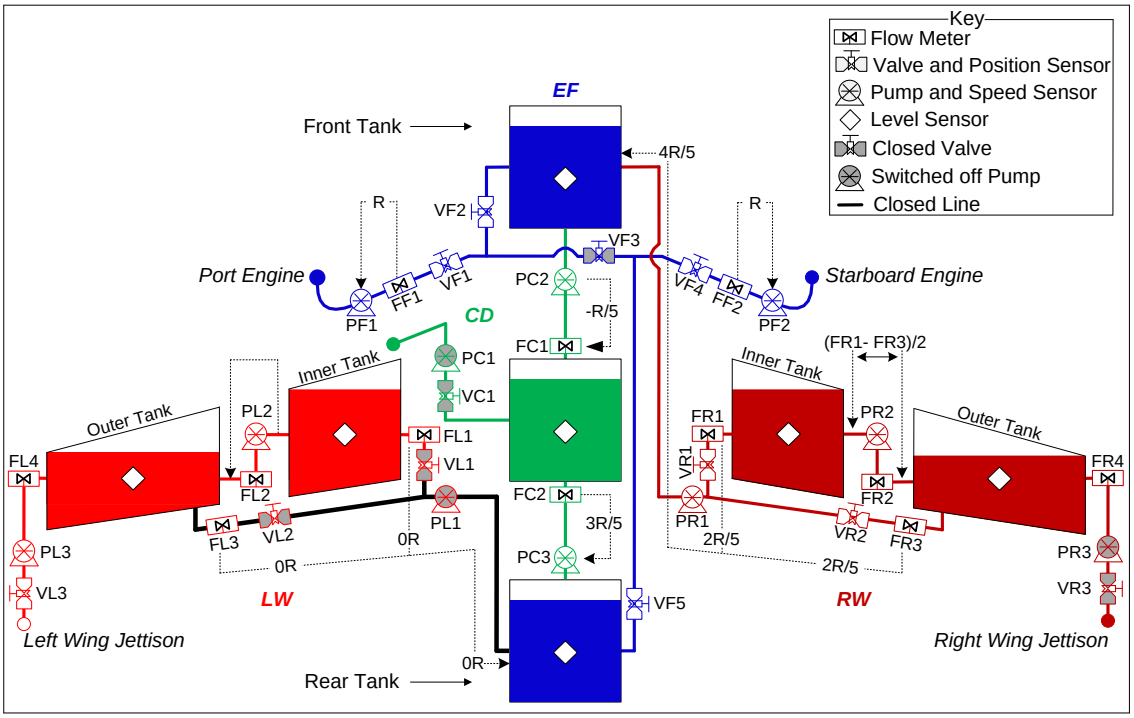


Figure 5-22: The Structure of the AFS after Tolerating “Leak in the inner tank of LW Sub-system”.

5.5.2.1 Fault Detection and Diagnosis of the Structural Leak

Figure 5-23 illustrates the response of the monitoring agent LW_MAG in detecting and diagnosing “Fuel leak in the inner tank of LW sub-system”. An interval of 5 seconds is taken by agent LW_MAG to detect and announce the corresponding alarm. This interval is required to collect the required measurements and evaluate the relevant expression. As this fault does not need a diagnostic process (i.e. the fault and its cause are in a one-to-one relationship) there is no extra time to diagnose the underlying cause and the 5 seconds is the interval to detect and diagnose the fault.

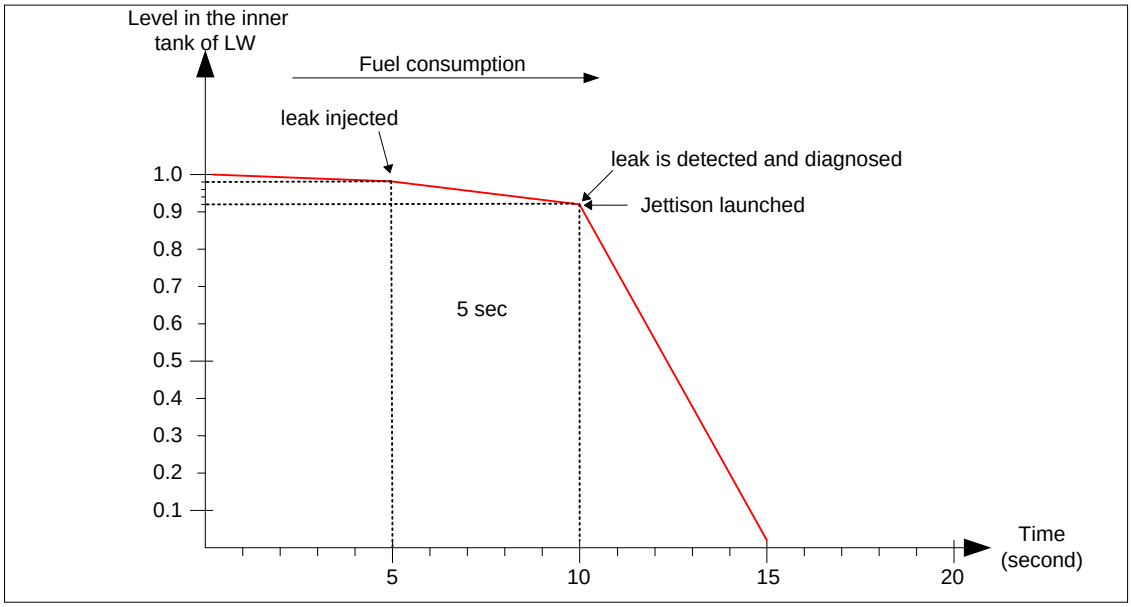


Figure 5-23: Monitor's Response in Detecting "Leak in the inner tank of LW Sub-system".

The fault has been detected after a discrepancy of 0.05 is verified between the reduction of the fuel level in the tank and the volume of the fuel flown out of the tank. Thereafter, a process of isolating and jettisoning the fuel of the left wing deposit is launched. Typically, fuel leak is a serious hazard that may result in fire or imbalance among the tanks (Langton *et al.*, 2009). Therefore, the specification of 5 seconds to collect the required sensory measurements and verify the occurrence of the event results effectively in detecting and diagnosing the fault before leakage of a large volume of fuel, causing level imbalance (and/or fire).

5.5.2.2 Alarm Annunciation on the Occurrence of Structural Leak

Alarm on the occurrence of the structural leak event as well as the conditions of the fuel system and its sub-systems are presented on the operators' interface as shown in Figure 5-24.

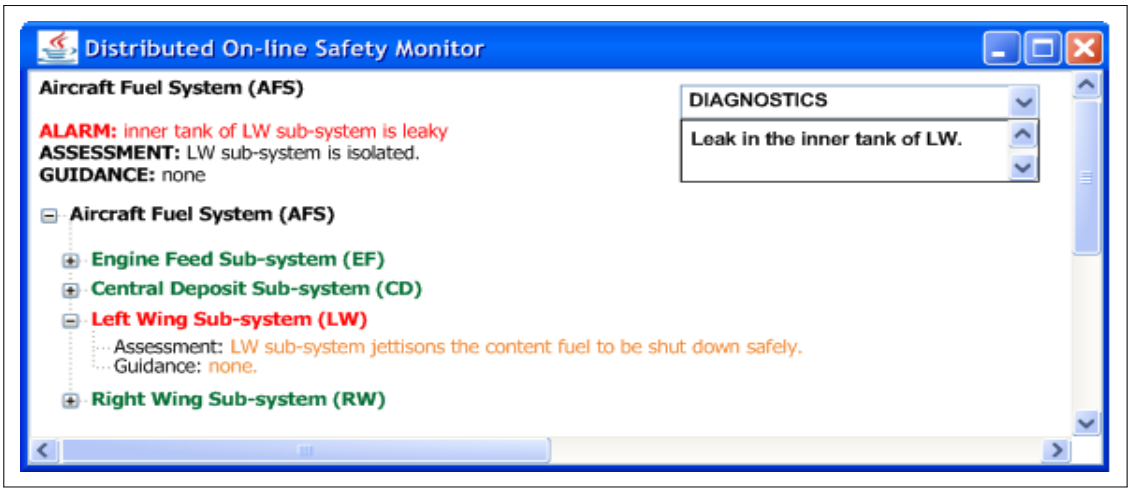


Figure 5-24: Operators' Interface after the Occurrence of "Leak in the inner tank of LW".

It can be seen that the interface provides the pilot with multi-level assessment according to the hierarchy of the fuel system. The interface also alarms the pilots on the occurrence of the fault, shows the diagnosed underlying causes and does not ask the operator to achieve any action. In this case, the presented information seems enough to understand the actual conditions of the fuel system.

5.5.2.3 Controlling the Fault of Structural Leak

Immediately after alarming the pilot on the occurrence of the leak, agent LW_MAG launches a local fault controlling process and communicates its state to the parent agent (AFS_MAG), which similarly communicates its state to the child agents (EF_MAG, CD_MAG and RW_MAG). Globally, the child agents collaborate and change their local rates to cope with the conditions.

After emptying the inner and outer tanks of the LW sub-system, agent LW_MAG achieves the following procedure:

- As the current state is LW_CM_TDS1, event (LL1_L < 0.1 AND LL2_L < 0.1) is verified.
- As the relevant ALARM attribute holds "none", no alarm is thus announced.
- From the relevant CONTROLLING attribute, agent LW_MAG achieves the following actions: close valve VL3 and switch pump PL3 off, as the inner and outer tanks are empty now and further jettison is not needed.
- Transits to state LW_CM_FS1, and thus the pilot is provided with assessment, "LW sub-system fails and has been shut down safely", and guidance, "none". This state is also communicated to the parent (AFS_MAG), but it does not trigger a state transition.

Thereafter, the operators' interface appears as shown in Figure 5-25. It can be seen that the monitor has controlled the failure by isolating and jettisoning the fuel of the affected sub-system. The evolutionary conditions of controlling task are confirmed to the pilot by the updatable interface, where alarm and alerting colours have been removed.

Similar to the controlling task that is delivered on the occurrence of "PEngine is not fed", the early fault detection and diagnosis, the incorporation of the corrective measures in the monitoring model and the updatable user interface have also supported the automatic controlling of the fuel leak and without adding extra load on the pilot.

[1] This state is also communicated to the parent (AFS_MAG), but it does not trigger a state transition.

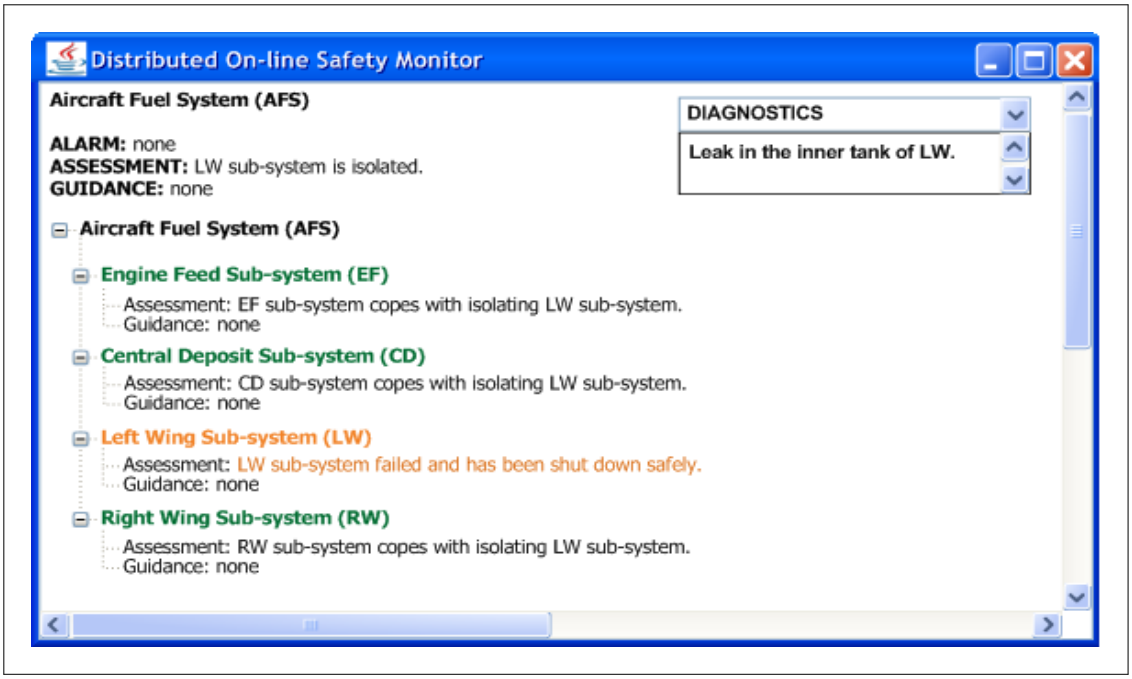


Figure 5-25: Operators' Interface after Controlling "Leak in the inner tank of LW Sub-system".

5.5.3 Third Injected Fault: Sensory Failure

A fault of the flow meter FC1 of the CD tank has been generated and experimentally injected by the simulator. The fault is detected when the evaluation of the following expression results in true:

(FC1_F > R/7 + 0.03 for 6 sec AND FC1_F > FC2_F + 0.03 for 6 sec)

OR

(FC1_F < R/7 - 0.03 for 6 sec AND FC1_F < FC2_F - 0.03 for 6 sec)

Where FC1_F > R/7 + 0.03 for 6 sec: is verified true when flow meter FC1 delivers measurements that are greater than the half of the demanded portion of the central tank (as the tank has two outlets) plus the possible bias over 6 seconds.

FC1_F > FC2_F + 0.03 for 6 sec: is verified true when flow meter FC1 delivers measurements that are greater than those are delivered by flow meter FC2 plus the possible bias over 6 seconds.

FC1_F < R/7 - 0.03 for 6 sec: is verified true when flow meter FC1 delivers measurements that are less than the half of the demanded portion of the central tank minus the possible bias over 6 seconds.

FC1_F < FC2_F - 0.03 for 6 sec: is verified true when flow meter FC1 delivers measurements that are less than those are delivered by flow meter FC2 minus the possible bias over 6 seconds.

In Table 5-4, the above expression is presented as follows:

(T(FC1_F > R/7 + 0.03, 6 sec) AND T(FC1_F > FC2_F + 0.03, 6 sec))
OR
(T(FC1_F < R/7 - 0.03, 6 sec) AND T(FC1_F < FC2_F - 0.03, 6 sec))

Once the monitoring agent CD_MAG verifies the occurrence of the sensory failure event, it perceives the state-transition table (Table 5-4) and launches the following procedure:

- From the relevant ALARM attribute, agent CD_MAG quotes and announces the alarm, "CD sub-system has a sensory failure".
- From the relevant CONTROLLING attribute, agent CD_MAG instructs the fuel system control unit (FSCU) to ignore measurements delivered by flow meter FC1 and depend alternatively on those delivered by flow meter FC2.
- From the relevant DIAGNOSIS attribute, agent CD_MAG quotes "Sensor FC1 has failed" and announces it as the diagnosed underlying cause.
- From the relevant NEW STATE attribute, agent CD_MAG transits to the new state, which is the permanent degraded state CD_CM_PDS1. From this state the pilot is provided with assessment, "CD sub-system operates degradedly" and guidance, "none".
- Agent CD_MAG also communicates the current state CD_CM_PDS1 to the parent (AFS_MAG).

When the agent AFS_MAG receives a message that conveys state CD_CM_PDS1, it perceives the state-transition table (Table 5-2) and achieves the following procedure:

- While the current state is AFS_CM_EFS, the received state results in verifying the occurrence of CD_CM_PDS1 == true.
- From the relevant NEW STATE attribute, agent AFS_MAG transits to the new state which is the permanent degraded state AFS_CM_PDS3. From this state the pilot is provided with assessment, "CD sub-system delivers degraded functionality" and guidance, "none".
- Agent AFS_MAG communicates state AFS_CM_PDS3 to the child agents: EF_MAG, LW_MAG and RW_MAG. As this state does not instantiate any active events of the children, no state transition is triggered and they do not take any action.

5.5.3.1 Fault Detection and Diagnosis of a Sensory Failure

As flow meter FC1 sticks delivering zero as permanent spurious measurements, the fuel level in the front and CD tanks has been affected. Figure 5-26 shows the response of the monitor in detecting and diagnosing a sensory failure. Agent CD_MAG detects the occurrence of the

relevant event and annunciates alarm after a maximum of 6 seconds. This interval is required to collect sensory measurements and evaluate the monitoring expression. As the fault and its underlying cause are in a one-to-one relationship, the underlying cause is diagnosed immediately, as identified by the relevant DIAGNOSIS cell of Table 5-4.

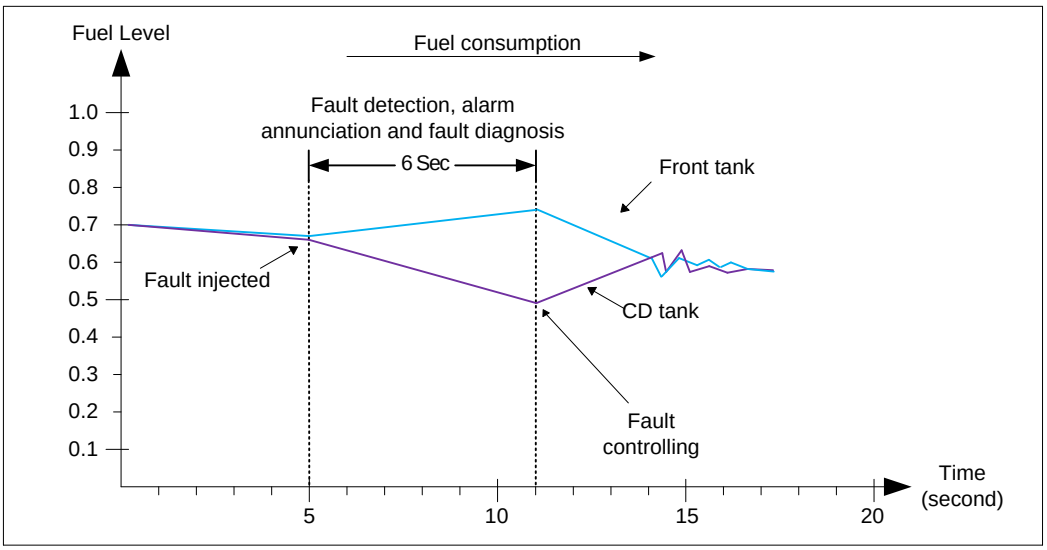


Figure 5-26: Monitor's Response in Detecting a Sensory Failure.

It can be seen that the occurrence of the event is verified when flow meter FC1 provides spurious measurements over an interval of 6 seconds. Measurements are verified spurious as they are compared with identical measurements of flow meter FC2 and also with possible measurements that can be provided by flow meter FC1. Until detecting the injected fault of flow meter FC1, a maximum discrepancy of 0.12 occurs between the intended and actual fuel level of the CD tank and front tank. This discrepancy falls in the allowable imbalance among the tanks of the fuel system and it can be corrected readily by the fuel system control unit (FSCU) (Langton *et al.*, 2009). Therefore, it can be said that in the case of a sensory failure, the monitor is able to deliver fault detection and diagnosis tasks.

The monitor has also shown ability to detect and diagnose multiple independent sensory faults[1], e.g. it detects and diagnoses the injection of two simultaneous faults of a pump speed sensor and a flow meter. However, the monitor shows limitation in detecting multiple dependent sensory faults. When simultaneous sensor faults of flow meter FC1 and flow meter FC2 were injected the monitor was unable to detect and diagnose the faults. The monitor ignored the sensory failure event and announced alarm on the occurrence of imbalance among the rear, front and CD tanks and diagnosed no cause for the anomalous conditions. This issue represents a clear

[1] Faults of more than one sensor that are not involved in validating the measurements of each other are called multiple independent sensory faults. On the contrary, faults of more than one sensor that are involved in validating the measurements of each other are called multiple dependent sensory faults.

limitation of the proposed technique to validate sensory failures and presents an area for future research.

5.5.3.2 Alarm Annunciation on the Occurrence of the Sensory Failure

During the process of formalisation of the relevant expression, the interval (6 seconds) of collecting historical sensory measurements was defined with consideration to achieving effective detection in a short possible time. Accordingly, an alarm on the fault is released immediately after verifying the expression. As before, on the occurrence of the sensory failure the monitor is also able to present well-organised alarm information that helps the pilots to understand the actual conditions of the fuel system.

To demonstrate and evaluate the ability of the monitor to deliver timely prognosis, let us assume that after controlling the fault, the aircraft has landed and during the pre-flying phase the refuelling mode is launched. This mode is triggered when the following expression is verified true:

VC1_P == 1 AND VF1_P == 0 AND VF4_P == 0;

Then agent CD_MAG perceives the state-transition table (Table 5-4) and achieves the following procedure:

- Execute the event on the table.
- As the ALARM attribute holds “none”, no alarm is thus announced.
- As the relevant CONTROLLING attribute holds “none”, no action is taken.
- As the relevant DIAGNOSIS attribute holds “not_needed”, then diagnosis is not launched.
- From the relevant NEW STATE attribute, agent CD_MAG transits to the permanent degraded state of the refuelling mode CD_RM_PDS1. From this state the pilot is provided with prognosis of assessment, “CD sub-system has a sensory failure” and guidance, “Flow meter FC1 must be replaced”.
- Agent CD_MAG also communicates the current state CD_RM_PDS1 to the parent agent (AFS_MAG).

When agent AFS_MAG receives a message conveying state CD_RM_PDS1, it perceives the state-transition table (Table 5-2) and achieves the following procedure:

- While the current state is AFS_CM_PDS3, the received state results in verifying the occurrence of CD_RM_PDS1 == true.
- From the relevant NEW STATE attribute, agent AFS_MAG transits to the new state, which is the permanent degraded state of the refuelling mode AFS_RM_PDS3. From this state the

pilot is provided with prognosis of assessment, “fuel system is not ready to achieve refuelling” and guidance, “none”.

- Agent AFS_MAG communicates state AFS_RM_PDS3 to the children: EF_MAG, LW_MAG and RW_MAG. As this state does not instantiate any active events of the children, no state transition will be triggered and they do not take any action.

This prognosis would appear on the operator interface as shown by Figure 5-27. It can be seen how the monitor avoids overwhelming the pilot with extra alarm information and provides timely prognosis according to the evolutionary behaviour of the fuel system.

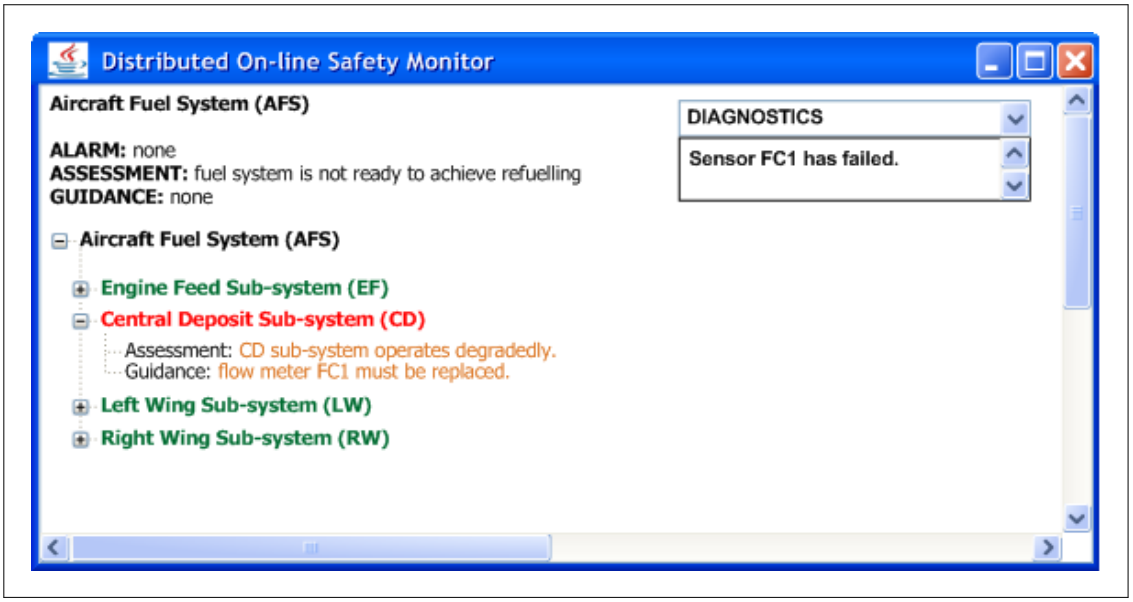


Figure 5-27: Operators’ Interface Showing Timely Prognosis on the Sensory Failure.

5.5.3.3 Fault Controlling of the Sensory Failure

As shown by Figure 5-26, after isolating flow meter FC1 and depending alternatively on measurements delivered by flow meter FC2, an interval of 6 seconds elapses before starting to bring the flow rates and the fuel levels of the CD and front tanks into the correct balance (see also Figure 5-7).

Therefore, it can be said that supported by its ability to achieve early detection and diagnoses of the sensory failure, the monitor is able to isolate the faulty sensor and measure the monitored conditions from alternative sensory measurements.

5.6 Summary

This chapter has demonstrated the possibility of deriving the distributed monitoring model from the HiP-HOPS safety assessment model. It has developed also a distributed monitor for an aircraft fuel system. For experimental purposes, a simulator that can virtualise the normal and abnormal conditions of the fuel system has been developed.

The developed monitor demonstrates an ability to deliver the intended monitoring tasks. Moreover, an evaluation of those tasks, which is based on different simulated failure scenarios, has revealed the ability of the monitor to deliver:

- Prompt fault detection and diagnosis.
- Effective alarm annunciation that is presented as well-organised alarm and multi-level assessment, guidance and timely prognosis.
- Control the injected faults automatically and without overloading or misleading the operators during emergency conditions.

The next chapter demonstrates the possibility of deriving the distributed monitoring model from the AADL safety assessment model. The chapter looks also at the ability of the monitor to deliver effective safety tasks by means of another case study; an aircraft brake system.

Chapter Six

Distributed On-line Safety Monitor Based on AADL and a Case Study of an Aircraft Brake System

6.1 Introduction

This chapter demonstrates the possibility of deriving the distributed monitoring model from the AADL safety assessment model. It also demonstrates application of the monitor to a case study, which was performed on a simulated model of an aircraft brake system (ABS), and evaluates the effectiveness of the monitoring tasks delivered by the monitor.

6.2 Aircraft Brake System (ABS)

The main functions of the ABS[1] are to slow down the aircraft during the taxiing and landing phases and achieve safe retardation in the case of rejected take-off. The brake function of the ABS is supported by anti-skid and an optional selection between auto-brake[2] in which the pilot pre-arms the rate of deceleration before the landing phase and manual brake in which the pilot applies the brake manually by depressing two pedals.

The secondary functions of the ABS are to (a) prevent unintended motion when the aircraft is parked (parking brake); (b) steer and directionally control the aircraft; (c) stop the rotation of the wheels while the gears are retracted.

Typically, an aircraft's brake systems are fitted on the main landing gears and not on the nose gear. The Boeing 757-200/300, for example, has two identical brake systems, right-side wheel brake and left-side wheel brake, each of which applies brake on four wheels.

Figure 6-1 illustrates the physical configuration of a hypothetical ABS. The ABS consists of:

- Pipes to deliver hydraulic pressure from three pressure lines normal, alternative and accumulative, through to the ABS's components and from there to the eight wheels. Pressure on those lines should not be less than 1300 PSI[3]. The lines are identical and redundant to improve the reliability of the ABS. Initially, ABS is pressured by the normal line. Should it fail, it is isolated and the alternative line is activated. Should the alternative line fail, it too is isolated and the accumulative line activated.

[1] Specifications of the ABS are obtained from ARP 4761 (1996), Sharvia (2010), Joshi and Heimdahl (2005 and 2007) and Biggles Software (2002).
[2] During the IIIb landing which is an auto-pilot controlled landing.
[3] PSI is an acronym for Pound-force Per Square Inch, a unit used to measure pressure.

- Two hydraulic fuses: left-side hydraulic fuse and right-side hydraulic fuse. The fuses are safety components and their role is to protect the two wheel nodes against a catastrophic failure, when the pressure on operative line exceeds the allowable limit. They release the extra pressure and allow the acceptable pressure to pass to the wheels.
- Three pressure sensors N_PS, A_PS and AC_PS to measure the pressure of the respective lines: normal, alternative and accumulative.
- Seven control valves, each coupled with a position sensor. Six valves are arranged to isolate and activate the three lines as follows: valves LN_CV and RN_CV activate and isolate the normal line, valves LA_CV and RA_CV activate and isolate the alternative line and valves LAC_CV and RAC_CV activate and isolate the accumulative line. The seventh valve (A_CV) is fitted between the alternative line and the accumulator; it allows pressure from the alternative line to be accumulated in the accumulator.
- Two auto-brake valves LN_ABV and RN_ABV, each coupled with in and out pressure sensors. These valves are able to apply automatic brake and manual brake. They are fitted only on the normal line and thus the optional selection between manual and auto-brake is applicable when this line is operative.
- Four meter valves, each coupled with in and out pressure sensors. Valves LA_MV and RA_MV are fitted on the alternative line and valves LAC_MV and RAC_MV are fitted on the accumulative. They regulate pressure passing to the wheels according to the brake rate demanded by the pedals (pedal position).
- Four anti-skid valves, each coupled with in and out pressure sensors. Valves LN_ASV and RN_ASV are fitted on the normal line and valves LA_ASV and RA_ASV are fitted on the alternative line. They release excessive pressure applied to the wheels and avoid potential skidding. The accumulative line is not fitted with these valves, so the anti-skid function is not applicable when this line is operative.
- Brake release and anti-skid return line. It bleeds off the excessive pressure that is cut by the anti-skid valves and pressure that is not needed after applying the brake. It also supports the parking brake, which is applied by fully depressing the two brake pedals, pulling the parking brake handle up and releasing the two pedals. To release the parking brake, the two brake pedals are fully depressed until the parking brake handle releases. Brake pressure and parking brake pressure are released by the anti-skid valves of the normal and alternative lines and by the meter valves of the accumulative line.

As shown in Figure 6-1, the valves and pressure lines of the ABS are arranged in two sub-systems: Right-side Wheel Brake (RWB) and Left-side Wheel Brake (LWB). A Braking System Control Unit (BSCU), which is a computerised controller, is also employed to control the entire ABS. The role of the BSCU can be detailed as follows:

- Interfacing the ABS with the pilot by (a) executing the flying instructions and demanded brake rates; (b) switching between the manual and auto brake and among the three pressure lines; (c) transmitting alarms and measurements to the cockpit.
- Controlling the anti-skid valves according to the required brake and the status of the system.
- Interfacing the ABS with other computers that concern the BCSU status.

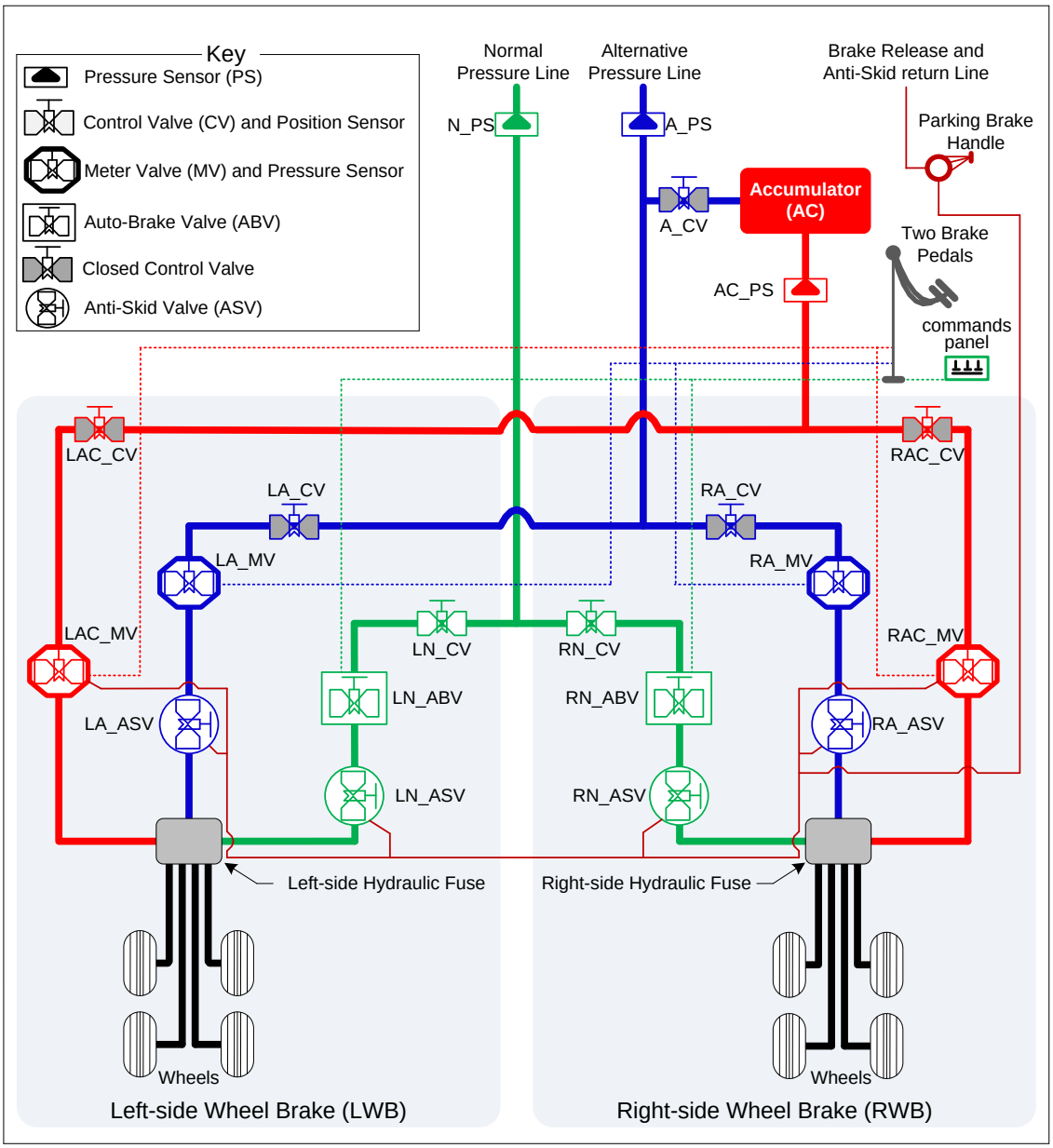


Figure 6-1: Illustration of the Aircraft Brake System.

6.2.1 Hierarchical Model of the ABS

The hierarchy of the ABS consists of three levels: system level (level*2*), sub-systems level (level*1*) and basic components level (level*0*). The Simulink block shown in Figure 6-2 illustrates the system level.

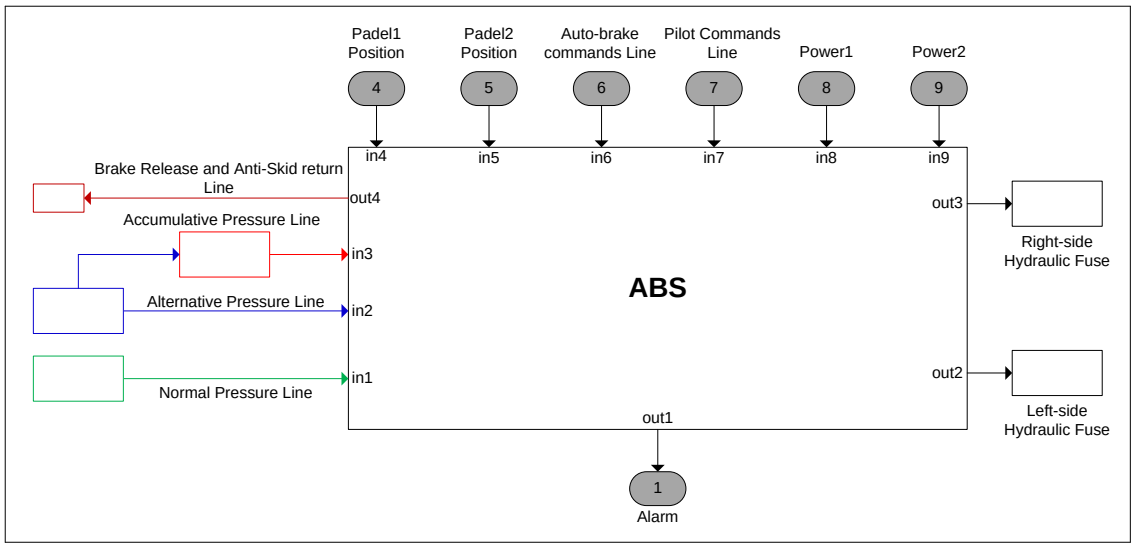


Figure 6-2: System Level of the Hierarchical Architecture of the ABS.

The ABS has nine inputs, as follows:

- Three pressure lines: normal, alternative and accumulative.
- Two pedal positions to deliver the demanded brake rate from the two pedals at the flight deck to the ABS.
- Two command lines: the auto-brake command line and pilot command line, to allow the pilot to pre-arm the brake commands and switch among the three pressure lines, respectively.
- Two redundant power lines (power1 and power2) to supply the BSCU with power; should power1 fail, power2 takes over.

The ABS has also four outputs, as follows:

- Two brake lines to the left-side and right-side landing gears, through the left-side hydraulic fuse and right-side hydraulic fuse, respectively.
- An alarm line, to communicate system status to the flight deck.
- The brake releasing and anti-skid return line.

Figure 6-3 illustrates the sub-system level (level*1*) of the ABS as a Simulink model. It shows that the ABS consists of three sub-systems: LWB, RWB and BSCU. The illustration shows how the BSCU and the LWB and RWB interface each other and interact with external devices. The LWB and RWB deliver sensory measurements to the BSCU, which in turn instructs the LWB and RWB by actuating commands.

The LWB and RWB also deliver brake pressure to the fuses and from there to the wheels of the landing gears. The BSCU receives instructions from the pedals and command lines and power from two lines and delivers alarms to the flight deck.

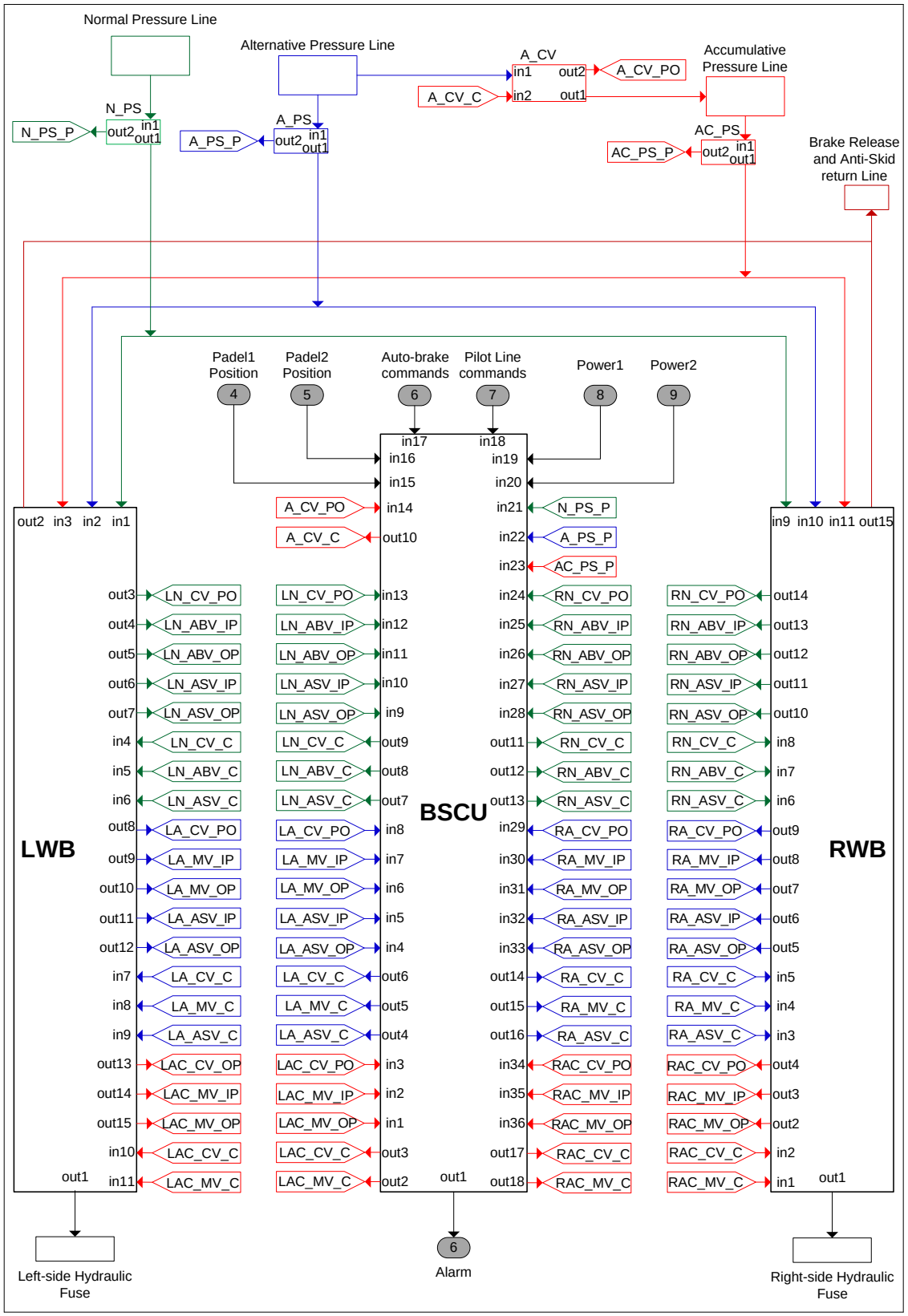


Figure 6-3: Level*1* of the Hierarchical Architecture of the ABS as a Simulink Model.

Figure 6-4 provides the syntactical grammar, as E-BNF notation, that is used to name the interfaces between the BSCU and the LWB and RWB. An interface is either a sensory measurement or an actuating command. A sensory measurement, in turn, consists of a three-part

name separated by two underscores: side and line initials, basic components initials and measurement initial. Similarly, every actuating command consists of a three-part name separated by two underscores: side and line initials, basic components initials and command initial. The syntax of the initials is also explained throughout the grammar.

interface	::= sensory_measurement | actuating_command;
sensory_measurement	::= side_initial line_initial, "_", basic_component_initial, "_", measurement_initial;
side_initial	::= "L" | "R";
line_initial	::= "N" | "A" | "AC";
basic_component_initial	::= "CV" | "MV" | "ABV" | "ASV" | "PS";
measurement_initial	::= "IP" | "OP" | "P" | "PO";
actuating_command	::= side_initial line_initial, "_", basic_component_initial, "_", command_initial;
command_initial	::= "C";
Where	
L	::= Left;
R	::= Right;
N	::= Normal Line;
A	::= Alternative Line;
AC	::= Accumulative Line;
CV	::= Control Valve;
MV	::= Meter Valve;
ABV	::= Auto-Brake Valve;
ASV	::= Anti-Skid Valve;
PS	::= Pressure Sensor;
IP	::= Input Pressure;
OP	::= Output Pressure;
P	::= Pressure;
PO	::= Position of a Control Valve (opened or closed);
C	::= Command;

Figure 6-4: E-BNF of the Sensory Measurements and Actuating Commands of the ABS.

Figure 6-5 illustrates the basic components level (level*0*) of the ABS as a Simulink model. In this illustration, the basic components of the BSCU have been omitted for the sake of brevity and also because the BSCU is a reliable component. The ABS is typically provided with primary and backup FSCUs, each of which is powered by redundant power lines (power1 and power2). Despite this omission, the interfaces between the BSCU and the LWB and RWB are still shown in the illustration.

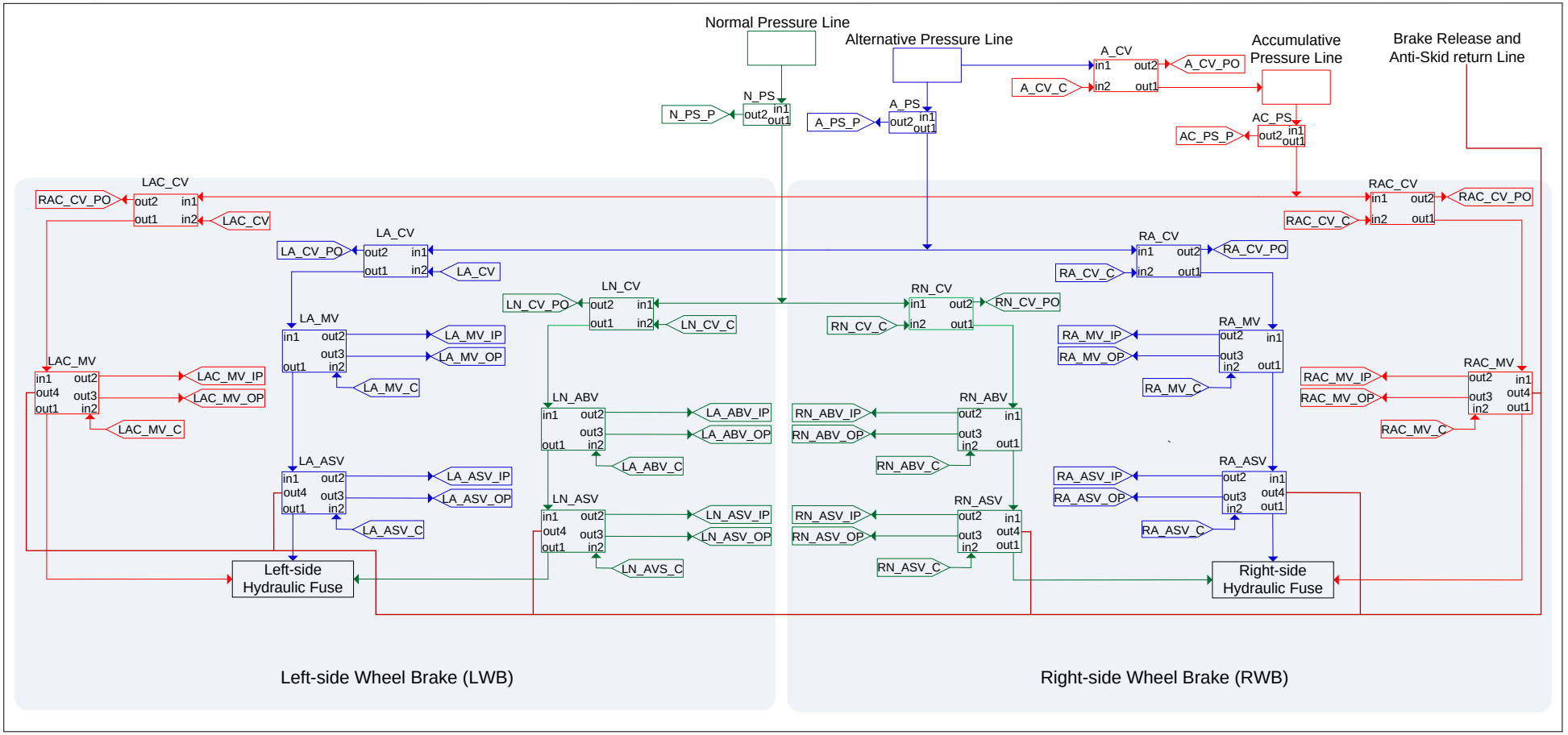


Figure 6-5: Level*0* of the Hierarchical Architecture of the ABS as a Simulink Model.

6.2.2 Operational Behaviour of the ABS

The behaviour of the ABS can be abstracted in three modes: normal, alternative and accumulative. Table 6-1 shows the functions provided in each mode. In normal mode, switching between the auto-brake and manual brake is optional and the anti-skid and parking brake functions are applicable. The hydraulic pressure in this mode is provided by the normal line.

In the alternative mode, the manual brake, anti-skid and parking functions are available and the auto-brake function is not applicable. The hydraulic pressure in this mode is provided by the alternative line. Finally, in the accumulative mode, the manual brake and parking brake are available and both anti-skid and auto-brake functions are not applicable. The hydraulic pressure in this mode is provided by the accumulative line.

Table 6-1: Functions Provided with each of the Three Pressure Lines of the ABS.

Mode	Deliverable functions
Normal	1- Auto-Brake. 2- Manual-Brake. 3- Anti-skid. 4- Parking Brake.
Alternative	1- Manual-Brake. 2- Anti-skid. 3- Parking Brake.
Accumulative	1- Manual Brake. 2- Parking Brake.

During flying the hydraulic pressure at the lines is monitored by the BSCU. Once the pressure drops below 1700 PSI, a warning light “BRAKE SOURCE” illuminates and the pilot is advised by a message of the Engine Indication and Crew Alerting System (EICAS) to switch to another brake line.

Across the taxiing, take-off, climbing, approaching and landing phases, hydraulic pressure may fluctuate over a normal range of 3400 to 3800 PSI; this is due to the other usages of hydraulic pressure. During the cruising phase, pressure may drop to fluctuate over another normal range of 2700 to 3100 PSI because of thermal effects at cruise altitude (low ambient temperature).

Figure 6-6 illustrates the behaviour of the pressure across the flying phases. The green shaded areas represent the normal fluctuations, while the red shaded area represents the abnormal pressure that may result due to failure of the pressure systems[1] of the aircraft.

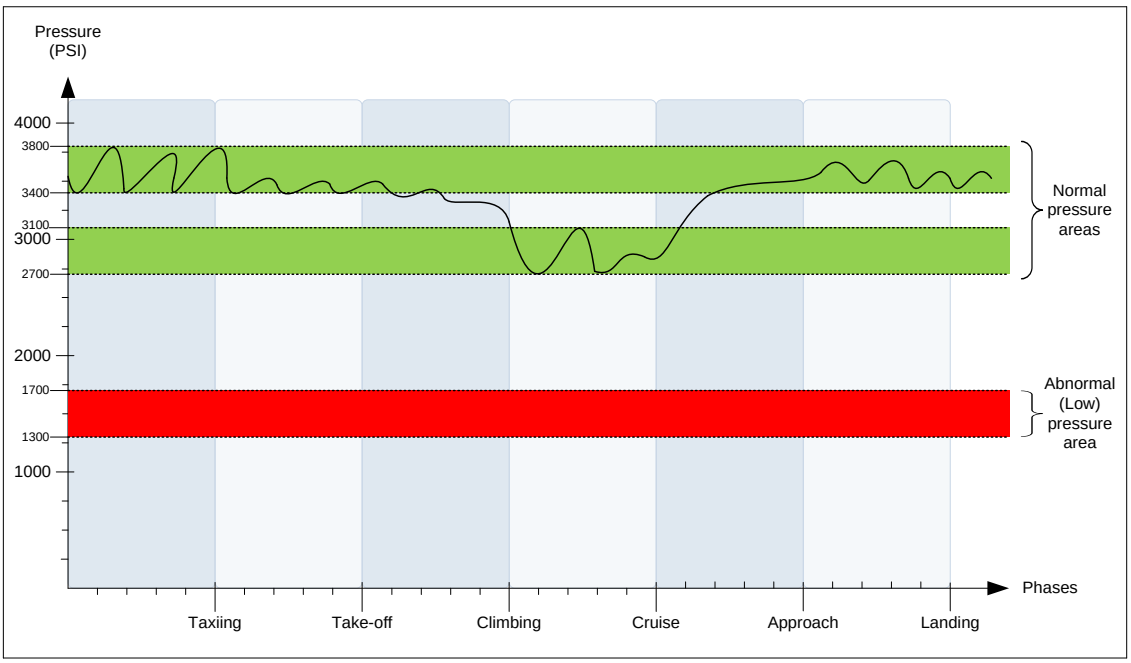


Figure 6-6: Hydraulic Pressure Behaviour across the Flying Phases.

The three pressure lines, the basic components and the design of the ABS support a passive fault-tolerant controlling (PFTC) means. To illustrate the behavioural transitions that support the PFTC, Figure 6-7 shows the hierarchical state-machines of the ABS. Transitions from one mode to another are triggered by either low pressure on the operative line or a fault of a basic component.

More specifically, the failure state of any basic component (BC [FS]) or low pressure of the operative line could result in state transitions of the LWB and RWB. The new states of the LWB and RWB trigger transition in the state-machine of the ABS and thus the entire ABS switches among the three modes[2]: normal mode (NM), alternative mode (AM) and accumulative mode (ACM). Figure 6-7 shows also how the state transitions of the ABS result in state transitions of the LWB and RWB.

[1] Hydraulic pressure of the three lines is typically provided by the left and right hydraulic systems of the aircraft.

[2] This is according to the definition of the mode, in section 3.3.2.

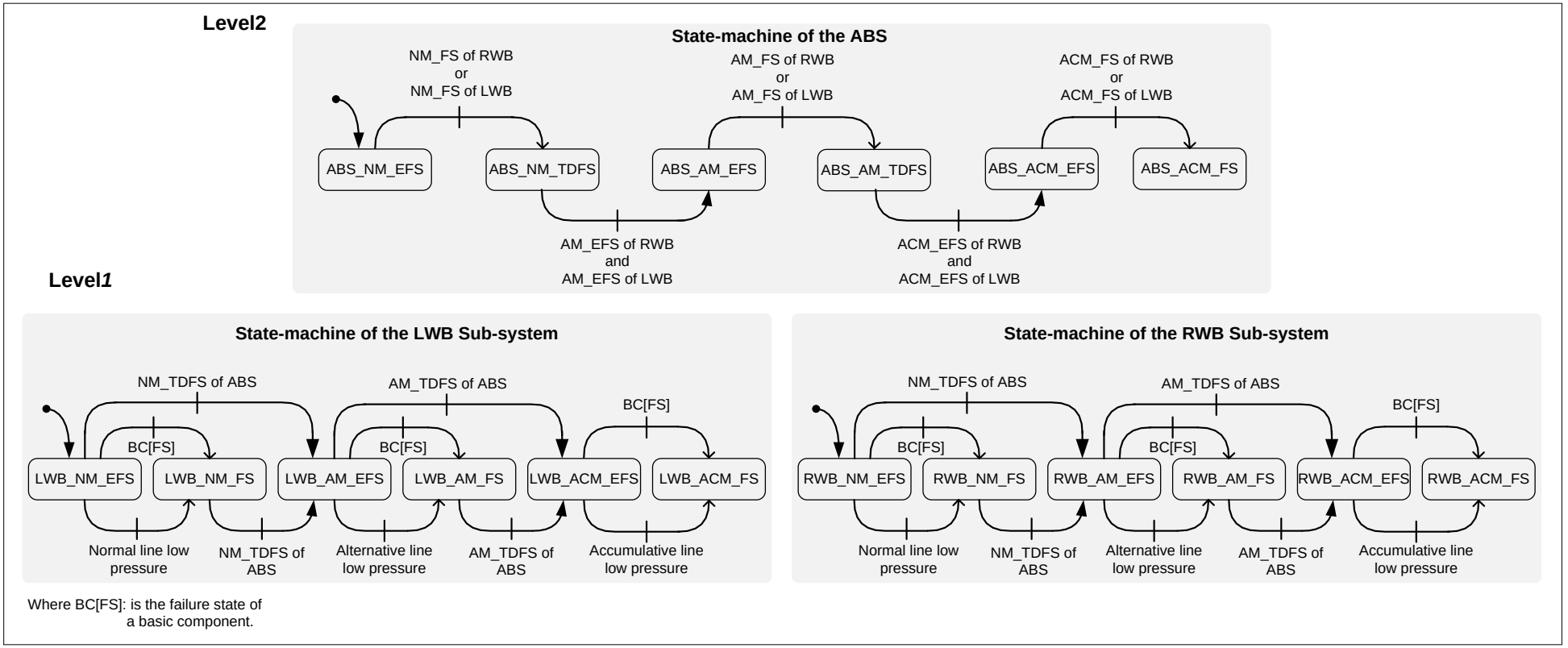


Figure 6-7: A Hierarchy of State-machines of the ABS and its Sub-systems; LWB and RWB.

6.3 Monitoring Agents Deployment

Figure 6-8 shows the hierarchical deployment and the collaboration protocols of three monitoring agents deployed to monitor the ABS. The agent that is deployed to monitor the ABS is termed ABS_MAG and the two agents that monitor the LWB and RWB are termed LWB_MAG and RWB_MAG, respectively.

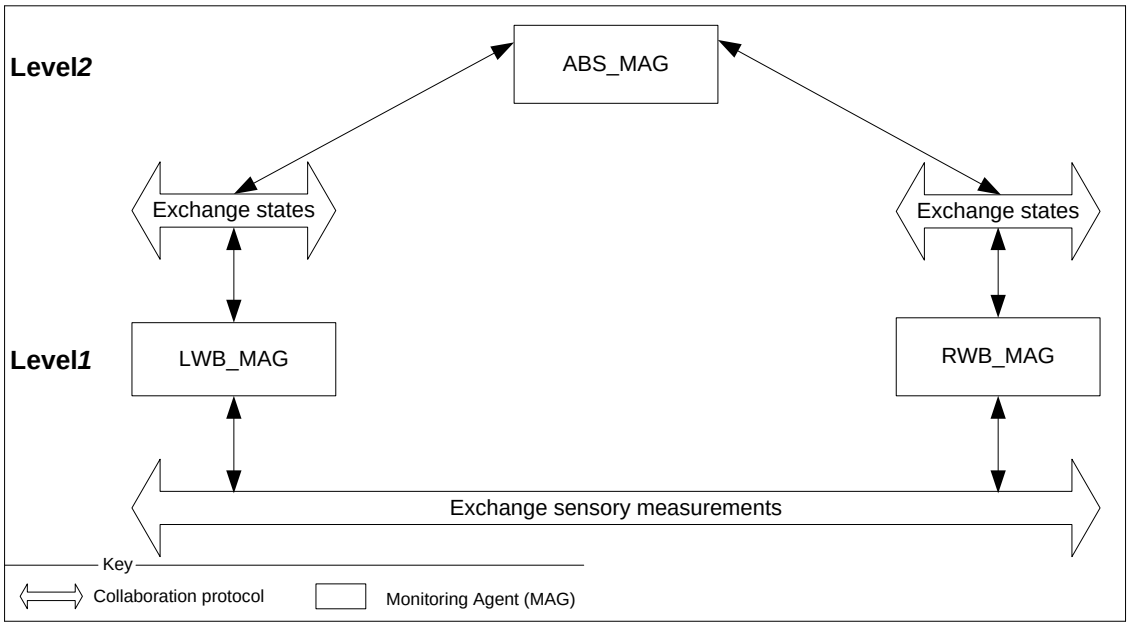


Figure 6-8: Collaboration Protocols of the Monitoring Agents.

6.4 AADL Model and the Distributed Monitoring Model

According to the AADL modelling nature, the assessment model of the ABS consists of firstly, a generic error model (or reusable model) in which abnormal behaviour of the ABS, the LWB and RWB sub-systems and the basic components is declared; secondly an architectural, behavioural (normal and abnormal) and error propagation specifications model. Both models are used to derive and formalise the distributed monitoring model. For the sake of brevity, the AADL model, presented here, includes only the aspects that concern the distributed monitoring model and the illustration of two failure scenarios simulated in section 6.5.1 and section 6.5.2.

6.4.1 Generic AADL Error Model

Figure 6-9 shows the generic error model of the AFS. In this model, the type declaration part "error model Basic" declares the following:

- A number of error-free states (EFSs) as initial states; EFS of the normal mode (NM_EFS), EFS of the alternative mode (AM_EFS), EFS of the accumulative mode (ACM_EFS) and EFS of the basic components (EFS).
- A number of error states: (a) temporary degraded/failure states of normal mode (NM_TDFS) and alternative mode (AM_TDFS); (b) failure states of normal mode (NM_FS), alternative mode (AM_FS), and accumulative mode (ACM_FS); (c) failure state of the basic components (FS).
- A number of error events: fault, fault1, fault2, fault3, fault4, fault5, fault6, fault7, fault8, fault9, fault_toleranceA and fault_toleranceACC.
- Error that may propagate among the basic components (low_pressure).

```
package Generic_Errors
public
annex Error_Model
{**
error model Basic
    features
        NM_EFS, AM_EFS, ACM_EFS, EFS : initial error state;
        NM_TDFS, AM_TDFS, NM_FS, AM_FS, ACM_FS, FS: error state;
        fault, fault1, fault2, fault3, fault4, fault5, fault6, fault7, fault8, fault9,fault_toleranceA, fault_toleranceACC: error event;
        low_pressure: in out error propagation;
 end Basic;

error model implementation Basic.BasicComponent
    transitions
        EFS – [fault, in low_pressure] – > FS;
        FS – [out low_pressure] – > FS;
end Basic.BasicComponent;

error model implementation Basic.BasicSubSystem
    transitions
        NM_EFS – [fault1] – > NM_FS;
        NM_EFS – [fault2] – > NM_FS;
        NM_FS – [fault_toleranceA] – > AM_EFS;
        NM_EFS – [fault_toleranceA] – > AM_EFS;
        AM_EFS – [fault3] – > AM_FS;
        AM_EFS – [fault4] – > AM_FS;
        AM_EFS – [fault_toleranceAC] – > ACM_EFS;
        AM_FS – [fault_toleranceAC] – > ACM_EFS;
        ACM_EFS – [fault5] – > ACM_FS;
        ACM_EFS – [fault6] – > ACM_FS;
end Basic.BasicSubSystem;

error model implementation Basic.BasicSystem
    transitions
        NM_EFS – [fault7] – > NM_TDFS;
        NM_TDFS – [fault_toleranceA] – > AM_EFS;
        AM_EFS – [fault8] – > AM_TDFS;
        AM_TDFS – [fault_toleranceAC] – > ACM_EFS;
        ACM_EFS – [fault9] – > ACM_FS;
end Basic.BasicSystem;
**}
end Generic_Errors;
```

Figure 6-9: Generic AADL Error Model of the ABS and its Components.

These declarations are applied to the basic components, sub-systems and the entire system; as shown by the three error model implementation parts: "error model implementation Basic.BasicComponent", "error model implementation Basic.BasicSubSystem" and "error model implementation Basic.BasicSystem".

The "error model implementation Basic.BasicComponent" part shows that every basic component has one EFS and one FS. While a basic component is in the FS, it propagates low pressure to the dependent one. It shows also that transitions from the EFS to the FS are triggered either by fault of the basic component or fault propagated from other components "in low_pressure".

The "error model implementation Basic.BasicSubSystem" shows that the abnormal behaviour of every sub-system can be described as follows:

- Three pairs of states, each consisting of EFS and FS. By matching those pairs with the "modes" parts of Figure 6-11 and Figure 6-14, it can be seen how each pair belongs to a mode; NM_EFS and NM_FS belong to the normal mode, AM_EFS and AM_FS belong to the alternative mode and ACM_EFS and ACM_FS belong to the accumulative mode.
- By matching three pairs of state with the "annex Error_model" parts of Figure 6-11 and Figure 6-14, it can also be seen that (a) transition between the states of each pair is triggered either by the FS of the basic components or by low pressure at the operative line; (b) transition from one pair to another (of a different mode) is triggered in order to tolerate a fault in the basic components of the other sub-system. For the purpose of simplicity, this scenario is illustrated also in the hierarchal state-machines of Figure 6-7.

The implementation declaration part of the system, "error model implementation Basic.BasicSystem", shows that the ABS has:

- Three pairs of states: two pairs consisting of EFS and TDFS and the third consisting of EFS and FS. By matching those pairs with the "modes" part of Figure 6-10, it can be seen how each pair belongs to a mode; NM_EFS and NM_TDFS belong to the normal mode, AM_EFS and AM_TDFS belong to the alternative mode and ACM_EFS and ACM_FS belong to the accumulative mode.
- By matching the three pairs of states with the "annex Error_model" part of Figure 6-10, it can be also seen that (a) the behaviour of the ABS is derived from the behaviour of the LWB and RWB, as shown after the "derived" clause; (b) transition between the states of each pair is triggered by a failure state of either the LWB or RWB; (c) transitions from TDFSs to EFSs are triggered when the LWB and RWB are able to tolerate faults. For the purpose of simplicity, this scenario is illustrated also in the hierarchal state-machines of Figure 6-7.

6.4.2 AADL Model and Monitoring Model of ABS

Figure 6-10 shows an abbreviated version of an AADL model of the ABS. The "subcomponents" and "modes" parts show the hierarchical specifications and the state

transitions of the ABS, respectively. The "annex Error_Model" part shows the hierarchical state mapping[1] of the ABS and how the state transitions of the LWB and RWB trigger transitions of the ABS.

```
system ABS
end ABS;

system implementation ABS.imp
   subcomponents
      LWB: system LWB.imp;
      RWB: system RWB.imp;

   modes
      Normal: initial mode;
      Alternative, Accumulative: mode;
      Normal: Normal – [NM_EFS] – > Normal;
      Normal1: Normal – [NM_TDFS] – > Normal;
      ToAlternative: Normal – [AM_EFS] – > Alternative;
      Alternative1: Alternative – [AM_TDFS] – > Alternative;
      ToAccumulative: Alternative – [ACM_EFS] – Accumulative;
      Accumulative1: Accumulative – [ACM_FS] – Accumulative;

   annex Error_Model {**
      model => Generic.model : : Basic.System;
      Model_hierarchy => derived;
      Derived_State_Maping =>
      Guard_Transition =>
      (LWB[NM_EFS] and RWB[NM_EFS] applies to Normal);
      Guard_Transition =>
      (LWB[NM_FS] or RWB[NM_FS] applies to Normal1);
      Guard_Transition =>
      (LWB[AM_EFS] and RWB[AM_EFS] applies to ToAlternative);
      Guard_Transition =>
      (LWB[AM_FS] or RWB[AM_FS] applies to Alternative1);
      Guard_Transition =>
      (LWB[ACM_EFS] and RWB[ACM_EFS] applies to ToAccumulative);
      Guard_Transition =>
      (LWB[ACM_FS] or RWB[ACM_FS] applies to Accumulative1);
      **}
end ABS.imp
```

Figure 6-10: Architectural Model and Error Model of ABS Presented by AADL.

To derive the state-transition table of the ABS it is necessary to consider both Figure 6-9 and Figure 6-10. Accordingly, trigger events that are shown in Figure 6-10, which appear after "Guard_Transition" clauses, are formalised into monitoring expressions as follows:

- The event that represents an exit from NM_EFS is:

  LWB[NM_FS] or RWB[NM_FS] → LWB_NM_FS == true OR RWB_NM_FS == true;

- The event that represents an exit from NM_TDFS is:

  LWB[AM_EFS] and RWB[AM_EFS] → LWB_AM_EFS == true
  AND

[1] More information about hierarchical mapping can be found in Feiler and Rugina (2007).

RWB_AM_EFS == true;

- The event that represents an exit from AM_EFS is:

  LWB[AM_FS] or RWB[AM_FS] → LWB_AM_FS == true OR RWB_AM_FS == true;

- The event that represents an exit from AC_TDFS is:

  LWB[ACM_EFS] and RWB[ACM_EFS] → LWB_ACM_EFS == true
  AND
  RWB_ACM_EFS == true;

- The event that represents an exit from ACM_EFS is:

  LWB[ACM_FS] or RWB[ACM_FS] → LWB_ACM_FS == true
  OR
  RWB_ACM_FS == true;

Accordingly, Table 6-2 illustrates the state-transition table of the ABS.

Table 6-2: State-transition Table of the ABS.

CURRENT STATE	CONDITIONS	EVENT	NEW STATE
ABS_NM_EFS	**Assessment**: normal line is operative and brake could be applied automatically or manually. **Guidance**: switching between manual and auto-brake is possible.	LWB_NM_FS == true **OR** RWB_NM_FS == true	ABS_NM_TDFS
ABS_NM_TDFS	**Assessment**: brake system is in a temporary failure. **Guidance**: fault controlling is in progress.	LWB_AM_EFS == true **AND** RWB_AM_EFS== true	ABS_AM_EFS
ABS_AM_EFS	**Assessment**: LWB and RWB are pressured by the alternative line. **Guidance**: only manual brake is applicable.	LWB_AM_FS == true **OR** RWB_AM_FS == true	ABS_AM_TDFS
ABS_AM_TDFS	**Assessment**: brake system is in a temporary failure. **Guidance**: fault controlling is in progress.	LWB_ACM_EFS== true **AND** RWB_ACM_EFS== true	ABS_ACM_EFS
ABS_ACM_EFS	**Assessment**: LWB and RWB are pressured by the accumulative line. **Guidance**: apply manual brake but anti-skid is unavailable.	LWB_ACM_FS == true **OR** RWB_ACM_FS== true	ABS_ACM_FS
ABS_ACM_FS	**Assessment**: brake system has failed permanently. **Guidance**: emergency conditions.	none	none

6.4.3 AADL Model and Monitoring Model of RWB Sub-system

Figure 6-11 shows an abbreviated version of the AADL model of the RWB. The "subcomponents" and "modes" parts show the hierarchical specifications and state transitions of the RWB, respectively. The "annex Error_Model" part shows the hierarchical state mapping and how the state transitions of the RWB are triggered by the transitions of the basic components and the states of the ABS.

```
system RWB
  features
  NormalLine: in event port;
  AlternativeLine: in event port;
  AccumulativeLine: in event port;
end RWB;

system implementation RWB.imp
  connections
    event port self.ToAlter – > AlternativeLine in modes Normal;
    event port self.ToAcc – > AccumulativeLine in modes Alternative;

  subcomponents
    N_PS: device N_PS.imp in modes Normal;
    A_PS: device A_PS.imp in modes Alternative;
    A_CV: device A_CV.imp in modes Alternative;
    AC_PS: device AC_PS.imp in modes Accumulative;
    RN_ASV: device RN_ASV.imp in modes Normal;
    RN_MV: device RN_MV.imp in modes Normal;
    RN_CV: device RN_CV.imp in modes Normal;
    RA_ASV: device RN_ASV.imp in modes Alternative;
    RA_MV: device RN_MV.imp in modes Alternative;
    RA_CV: device RN_CV.imp in modes Alternative;
    RAC_MV: device RN_MV.imp in modes Accumulative;
    RAC_CV: device RN_CV.imp in modes Accumulative;

  modes
    Normal: initial mode;
    Alternative, Accumulative: mode;
    Normal – [self.NM_EFS] – > Normal;
    Normal – [self.NM_FS] – > Normal;
    Normal – [self.ToAlter] – > Alternative;
    Alternative – [self.AM_EFS] – > Alternative;
    Alternative – [self.AM_FS] – > Alternative;
    Alternative – [self.ToAcc] – > Accumulative;
    Accumulative – [self.ACM_EFS] – > Accumulative;
    Accumulative – [self.ACM_FS] – > Accumulative;

  annex Error_Model {**
    model => Generic.model : : Basic.SubSystem;
    Model_Hierarchy => derived;
    Derived_State_Maping =>
    NM_EFS when N_PS[EFS] and RN_CV[EFS] and RN_MV[EFS] and RN_ABV[EFS] and RN_ASV[EFS],
    NM_FS when RN_CV[FS] or RN_ABV[FS] or RN_ASV[FS],
    NM_FS when N_PS[FS],
    AM_EFS when A_PS[EFS] and RA_CV[EFS] and RA_MV[EFS] and RA_ASV[EFS],
    AM_FS when RA_CV[FS] or RA_MV[FS] or RA_ASV[FS],
    AM_FS when A_PS[FS],
    ACM_EFS when AC_PS[EFS] and RAC_CV[EFS] and RAC_MV[EFS],
    ACM_FS when RAC_CV[FS] or RAC_MV[FS],
    ACM_FS when AC_PS[FS];
    **}
end RWB.imp
```

Figure 6-11: Architectural Model and Error Model of RWB Sub-system Presented by AADL.

To derive the state-transition table of the RWB, Figure 6-11 and Figure 6-9 should be considered together. Trigger events, which appear after "when" clauses in Figure 6-11, can be formalised into monitoring expressions as follows:

- Events that represent exits from NM_EFS are:

RN_CV[FS] or RN_ABV[FS] or RN_ASV[FS][1] → RN_ASV_OP > RN_ASV_C + 50
OR
RN_ASV_OP < RN_ASV_C - 50;

ToAlter → ABS_NM_TDFS == true;

N_PS[FS] → N_PS_P – 1300 < 50;

"ToAlter" represents an event that triggers a transition to the alternative mode; as shown in the "connections" part of Figure 6-11. For simplicity, this event is illustrated as the trigger event, NM_TDFS of ABS, in the state-machine of the RWB of Figure 6-7 and its occurrence results in a state transition from state RWB_NM_EFS to state RWB_AM_EFS.

- The event that represents an exit from NM_FS is:

ToAlter → ABS_NM_TDFS == true;

"ToAlter" represents an event that triggers a transition to the alternative mode, as shown in the "connection" part of Figure 6-11. It is applicable to the two states NM_EFS and NM_FS of the normal mode. For simplicity, this event is illustrated as the trigger event, NM_TDFS of ABS, in the state-machine of the RWB of Figure 6-7 and its occurrence results in a state transition from state RWB_NM_FS to state RWB_AM_EFS.

- Events that represent exits from AM_EFS are:

RA_CV[FS] or RA_MV[FS] or RA_ASV[FS] → RA_ASV_OP > RA_ASV_C + 50
OR
RA_ASV_OP < RA_ASV_C - 50;

ToACC → ABS_AM_TDFS == true;

A_PS[FS] → A_PS_P – 1300 < 50;

"ToACC" represents an event that triggers a transition to the accumulative mode; as shown in the "connections" part of Figure 6-11. For simplicity, this event is illustrated as the trigger event, AM_TDFS of ABS, in the state-machine of the RWB of Figure 6-7 and its occurrence results in a state transition from state RWB_AM_EFS to state RWB_ACM_EFS.

- The event that represents an exit from AM_FS is:

ToACC → ABS_AM_TDFS == true;

[1] A range of ±50 PSI is the assumed bias of the sensors of the brake system.

"ToACC" represents an event that triggers a transition to the accumulative mode, as shown in the "connection" part of Figure 6-11. It is applicable to the two states AM_EFS and AM_FS of the normal mode. For simplicity, this event is illustrated as the trigger event, AM_TDFS of ABS, in the state-machine of the RWB of Figure 6-7 and its occurrence results in a state transition from state RWB_AM_FS to state RWB_ACM_EFS.

- Events that represent exits from ACM_EFS are:

RAC_CV[FS] or RAC_MV[FS] → RAC_MV_OP > RAC_MV_C + 50
OR
RAC_MV_OP < RAC_MV_C - 50;

AC_PS[FS] → AC_PS_P – 1300 < 50;

Accordingly, Table 6-3 illustrates the state-transition table of the RWB.

Table 6-3: State-transition Table of the RWB Sub-system.

CURRENT STATE	CONDITION	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
RWB_NM_EFS	**Assessment:** RWB operates normally**. Guidance:** none	RN_ASV_OP > RN_ASV_C + 50 OR RN_ASV_OP < RN_ASV_C - 50;	normal brake failed	- RA_CV_C = 1; - RN_CV_C = 0;	needed	RWB_NM_FS
		ABS_NM_TDFS == true	none	- RA_CV_C = 1; - RN_CV_C = 0;	not_needed	RWB_AM_EFS
		N_PS_P – 1300 < 50	low pressure on normal line	- RA_CV_C = 1; - RN_CV_C = 0;	Pressure at the normal line is low.	RWB_NM_FS
RWB_NM_FS	**Assessment:** normal line of RWB has failed**. Guidance:** fault controlling is in progress.	ABS_NM_TDFS == true	none	not_needed	not_needed	RWB_AM_EFS
RWB_AM_EFS	**Assessment:** RWB is pressured by the alternative line. **Guidance:** apply manual brake.	RA_ASV_OP > RA_ASV_C + 50 OR RA_ASV_OP < RA_ASV_C - 50;	alternative brake failed	- RAC_CV_C=1; - RA_CV_C = 0;	needed	RWB_AM_FS
		ABS_AM_TDFS == true	none	- RAC_CV_C=1; - RA_CV_C = 0;	not_needed	RWB_ACM_EFS
		A_PS_P – 1300 < 50	low pressure at alternative line	- RAC_CV_C=1; - RA_CV_C = 0;	Pressure at the alternative line is low.	RWB_AM_FS
RWB_AM_FS	**Assessment:** alternative line of RWB has failed. **Guidance:** fault controlling is in progress.	ABS_AM_TDFS == true	none	not_needed	not_needed	RWB_ACM_EFS
RWB_ACM_EFS	**Assessment:** RWB is pressured by the accumulative line. **Guidance:** apply manual brake.	RAC_MV_OP > RAC_MV_C + 50 OR RAC_MV_OP < RAC_MV_C - 50;	accumulative brake failed	impossible	needed	RWB_ACM_FS
		AC_PS_P – 1300 < 50	low pressure on accumulative line	impossible	Pressure on the accumulative line is low.	RWB_ACM_FS
RWB_ACM_FS	**Assessment:** RWB fails there is no brake on the right-side landing gear **Guidance:** no brake is available	none	none	not_needed	not_needed	none

Figure 6-12 shows the architectural model along with the error model of the basic components of the normal line of the RWB sub-system.

```
device N_PS
  features
    N_PS_in1: in event port;
    N_PS_out1: out event port;
    N_PS_out2: out event port N_PS_P;
end N_PS;

device implementation N_PS.imp
annex Error_Model {**
      Model => Generic.model : : Basic.BasicComponent;
               Guard_In =>
                        low_pressure when N_PS_in1[low_pressure] applies to N_PS_in1;
               Guard_Out =>
                         low_pressure when N_PS_in1[low_pressure] or self [FS] applies to N_PS_out1;
               Guard_Out =>
                         low_pressure when N_PS_in1[low_pressure] or self [FS] applies to N_PS_out2;
**};
end N_PS.imp;

device RN_CV
  features
    RN_CV_in1: in event port N_PS_out1;
    RN_CV_in2: in event port RN_CV_C;
    RN_CV_out1: out event port;
    RN_CV_out2: out event port RN_CV_PM;
end RN_CV;

device implementation RN_CV.imp
annex Error_Model {**
      Model => Generic.model : : Basic.BasicComponent;
               Guard_In =>
                        low_pressure when RN_CV_in1[low_pressure] applies to RN_CV_in1;
               Guard_Out =>
                         low_pressure when N_PS_in1[low_pressure] or self [FS] applies to RN_CV_out1;
               Guard_Out =>
                         low_pressure when N_PS_in1[low_pressure] or self [FS] applies to RN_CV_out2;
**};
end RN_CV.imp;

device RN_ABV
  features
    RN_ABV_in1: in event port RN_CV_out1;
    RN_ABV_in2: in event port RN_ABV_C;
    RN_ABV_out1: out event port;
    RN_ABV_out2: out event port RN_ABV_INprM;
    RN_ABV_out3: out event port RN_ABV_OprM;
end RN_ABV;

device implementation RN_ABV.imp
annex Error_Model {**
      Model => Generic.model : : Basic.BasicComponent;
               Guard_In =>
                        low_pressure when RN_ABV_in1[low_pressure] applies to RN_ABV_in1;
               Guard_Out =>
                         low_pressure when RN_ABV_in1[low_pressure] or self [FS] applies to RN_ABV_out1;
               Guard_Out =>
                         low_pressure when RN_ABV_in1[low_pressure] or self [FS] applies to RN_ABV_out2;
               Guard_Out =>
                         low_pressure when RN_ABV_in1[low_pressure] or self [FS] applies to RN_ABV_out3;
**};
end RN_ABV.imp;

device RN_ASV
  features
    RN_ASV_in1: in event port RN_ABV_out1;
    RN_ASV_in2: in event port RN_ABV_C;
    RN_ASV_out1: out event port;
    RN_ASV_out2: out event port RN_ASV_INprM;
    RN_ASV_out3: out event port RN_ASV_OprM;
end RN_ASV;

device implementation RN_ASV.imp
annex Error_Model {**
      Model => Generic.model : : Basic.BasicComponent;
               Guard_In =>
                        low_pressure when RN_ASV_in1[low_pressure] applies to RN_ASV_in1;
               Guard_Out =>
                         low_pressure when RN_ASV_in1[low_pressure] or self [FS] applies to RN_ASV_out1;
               Guard_Out =>
                         low_pressure when RN_ASV_in1[low_pressure] or self [FS] applies to RN_ASV_out2;
               Guard_Out =>
                         low_pressure when RN_ASV_in1[low_pressure] or self [FS] applies to RN_ASV_out3;
**};
end RN_ASV.imp;
```

Figure 6-12: Architectural and Error Models of the Basic Components of RWB, as AADL Model.

In Figure 6-12, it can be seen how the basic components are connected to each other and propagate errors. Consider, for example, the N_PS which has an input port N_PS_in1 and two output ports N_PS_out1 and N_PS_out2. It might be affected by a propagated error of low pressure on the normal line, as shown by the "Guard_in =>" clause. It might also propagate an error of low pressure through the two output ports, whenever it is affected by the propagation of low pressure error or by its own fault, as shown by the "Guard_Out =>" clause. Note that error that is propagated by port N_PS_out2 means that the sensor delivers a measurement that reflects the current conditions as that port is connected to BSCU; the port and the connection can be seen in Figure 6-3.

Figure 6-13 shows the formal diagnostic model of the failure event RWB_NM_EFS_E1, declared in Table 6-3. The model is derived from the error propagation state-machines of the AADL model shown by Figure 6-12.

In Figure 6-13, monitoring expressions of the symptoms are formalised into monitoring expressions as follows:

- The monitoring expression of the symptom "anti-skid valve RN_ASV is faulty" is:

RN_CV_PO == 1 AND RN_ABV_IP – RN_ABV_OP < 100;

Where RN_CV_PO == 1: is to verify whether valve RN_CV is open.

RN_ABV_IP – RN_ABV_OP < 100: is to verify whether valve RN_ABV provides a normal function. RN_ABV_IP – RN_ABV_OP is the difference between the input and output pressure of valve RN_ABV. For stopping the rotation of the wheels while the gears are retracted, there is a small amount of pressure passes continually through auto-brake valve (ABV). The pressure is also used to verify the normal function of the ABV.

100: is the accumulation of the possible bias of the two pressure sensors, RN_ABV_IP and RN_ABV_OP, of the RN_ABV; 50 PSI a possible bias of each sensor.

- The monitoring expression of the symptom "auto-brake valve RN_ABV is faulty" is:

RN_ABV_IP – RN_ABV_OP > 100;

- The monitoring expression of the symptom "control valve RN_CV is faulty" is:

RN_CV_PO == 0

FailureEvent: RN_ASV_OP > RN_ASV_C + 50 **OR** RN_ASV_OP < RN_ASV_C - 50.
Propagator: RN_ASV **OR** RN_ABV **OR** RN_CV.

EStateName: RN_ASV.
Symptom: RN_CV_PO == 1 AND RN_ABV_IP – RN_ABV_OP < 100.
Fault: anti-skid valve RN_ASV is faulty.
Controlling: none.

EStateName: RN_ABV.
Symptom: RN_ABV_IP – RN_ABV_OP > 100.
Fault: auto-brake valve RN_ABV is faulty.
Controlling: none.

EStateName: RN_CV.
Symptom: RN_CV_PO **==** 0.
Fault: control valve RN_CV is faulty.
Controlling: none.

Figure 6-13: Diagnostic Model of the Failure Event "normal brake failed".

6.4.4 AADL Model and Monitoring Model of LWB Sub-system

Figure 6-14 shows an abbreviated version of the AADL model of the LWB. The "subcomponents" and "modes" parts show the hierarchical specifications and the state transitions of the LWB. The "annex Error_Model" part shows the hierarchical state mapping and how the state transitions of the LWB are triggered by the transitions of the basic components and the states of the ABS.

To derive the state-transition table of the RWB, Figure 6-14 and Figure 6-9 should be considered together. Trigger events, which appear after "when" clauses in Figure 6-14, are formalised as monitoring expressions as follows:

- Events that represent exits from NM_EFS are:

  LN_CV[FS] or LN_ABV[FS] or LN_ASV[FS] → LN_ASV_OP > LN_ASV_C + 50
  OR
  LN_ASV_OP < LN_ASV_C - 50;

  ToAlter → ABS_NM_TDFS == true;

  "ToAlter" represents an event that triggers a transition to the alternative mode, as shown in the "connections" part of Figure 6-14. For simplicity, this event is illustrated as trigger event NM_TDFS of ABS in the state-machine of the LWB of Figure 6-7 and its occurrence results in a state transition from state LWB_NM_EFS to state LWB_AM_EFS.

- The event that represents an exit from NM_FS is:

  ToAlter → ABS_NM_TDFS == true;

  "ToAlter" again represents an event that triggers a transition to the alternative mode, as shown in the "connections" part of Figure 6-14. It is applicable to the two states NM_EFS

and NM_FS of the normal mode. For simplicity, this event is illustrated as trigger event NM_TDFS of ABS in the state-machine of the LWB of Figure 6-7 and its occurrence results in a state transition from state LWB_NM_FS to state LWB_AM_EFS.

```
system LWS
   features
   NormalLine: in event port;
   AlternativeLine: in event port;
   AccumulativeLine: in event port;
end LWS;

system implementation LWS.imp
   connections
      event port self.ToAlter – > AlternativeLine in modes Normal;
      event port self.ToAcc – > AccumulativeLine in modes Alternative;

   subcomponents
      LN_ASV: device LN_ASV.imp in modes Normal;
      LN_MV: device LN_MV.imp in modes Normal;
      LN_CV: device LN_CV.imp in modes Normal;
      LA_ASV: device LN_ASV.imp in modes Alternative;
      LA_MV: device LN_MV.imp in modes Alternative;
      LA_CV: device LN_CV.imp in modes Alternative;
      LAC_MV: device LN_MV.imp in modes Accumulative;
      LAC_CV: device LN_CV.imp in modes Accumulative;

   modes
      Normal: initial mode;
      Alternative, Accumulative: mode;
      Normal – [self.NM_EFS] – > Normal;
      Normal – [self.NM_FS] – > Normal;
      Normal – [self.ToAlter] – > Alternative;
      Alternative – [self.AM_EFS] – > Alternative;
      Alternative – [self.AM_FS] – > Alternative;
      Alternative – [self.ToAcc] – > Accumulative;
      Accumulative – [self.ACM_EFS] – > Accumulative;
      Accumulative – [self.ACM_FS] – > Accumulative;

   annex Error_Model {**
      model => Generic.model : : Basic.SubSystem;
      Model_Hierarchy => derived;
      Derived_State_Maping =>
      NM_EFS when LN_CV[EFS] and LN_MV[EFS] and LN_ABV[EFS] and LN_ASV[EFS],
      NM_FS when LN_CV[FS] or LN_ABV[FS] or LN_ASV[FS],
      NM_FS when N_PS[FS],
      AM_EFS when LA_CV[EFS] and LA_MV[EFS] and LA_ASV[EFS],
      AM_FS when LA_CV[FS] or LA_MV[FS] or LA_ASV[FS],
      AM_FS when A_PS[FS],
      ACM_EFS when LAC_CV[EFS] and LAC_MV[EFS],
      ACM_FS when LAC_CV[FS] or LAC_MV[FS],
      ACM_FS when AC_PS[FS];
      **}
end LWS.imp
```

Figure 6-14: Architectural Model and Error Model of LWB Sub-system Presented by AADL.

- Events that represent exits from AM_EFS are:

LA_CV[FS] or LA_MV[FS] or LA_ASV[FS] → LA_ASV_OP > LA_ASV_C + 50
OR
LA_ASV_OP < LA_ASV_C - 50;

ToACC → ABS_AM_TDFS == true;

"ToACC" represents an event that triggers a transition to the accumulative mode, as shown in the "connections" part of Figure 6-14. For simplicity, this event is illustrated as trigger event AM_TDFS of ABS in the state-machine of the RWB of Figure 6-7 and its occurrence results in a state transition from state LWB_AM_EFS to state LWB_ACM_EFS.

- The event that represents an exit from AM_FS is:

  ToACC → ABS_AM_TDFS == true;

  "ToACC" again represents an event that triggers a transition to the accumulative mode, as shown in the "connections" part of Figure 6-14. It is applicable to the two states AM_EFS and AM_FS of the normal mode. For simplicity, this event is illustrated as trigger event ABS_AM_TDFS in the state-machine of the LWB of Figure 6-7 and its occurrence results in a state transition from state LWB_AM_FS to state LWB_ACM_EFS.

- The event that represents an exit from ACM_EFS is:

  LAC_CV[FS] or LAC_MV[FS] → LAC_MV_OP > LAC_MV_C + 50
  OR
  RAC_MV_OP < LAC_MV_C - 50;

Accordingly, Table 6-4 illustrates the formal state-transition table of the LWB.

Table 6-4: State-transition Table of the LWB Sub-system.

CURRENT STATE	CONDITIONS	EVENT	ALARM	CONTROLLING	DIAGNOSIS	NEW STATE
LWB_NM_EFS	**Assessment:** LWB operates normally. **Guidance:** none.	LN_ASV_OP > LN_ASV_C + 50 OR LN_ASV_OP < LN_ASV_C - 50;	normal brake failed	- LA_CV_C = 1; - LN_CV_C = 0;	needed	LWB_NM_FS
		ABS_NM_TDFS == true	none	- LA_CV_C = 1; - LN_CV_C = 0;	not_needed	LWB_AM_EFS
LWB_NM_FS	**Assessment:** normal line of LWB has failed. **Guidance:** fault controlling is in progress.	ABS_NM_TDFS == true	none	not_needed	not_needed	LWB_AM_EFS
LWB_AM_EFS	**Assessment:** LWB is pressured by the alternative line **Guidance:** apply manual brake.	LA_ASV_OP > LA_ASV_C+ 50 OR LA_ASV_OP < LA_ASV_C- 50;	alternative brake failed	- LAC_CV_C=1; - LA_CV_C = 0;	needed	LWB_AM_FS
		ABS_AM_TDFS == true	none	- LAC_CV_C=1; - LA_CV_C = 0;	not_needed	LWB_ACM_EFS
LWB_AM_FS	**Assessment:** the alternative line of the RWB has failed. **Guidance:** fault controlling is in progress.	ABS_AM_TDFS == true	none	not_needed	not_needed	LWB_ACM_EFS
LWB_ACM_EFS	**Assessment:** LWB is pressured by the accumulative line. **Guidance:** apply manual brake.	LAC_MV_OP > LAC_MV_C + 50 OR LAC_MV_OP < LAC_MV_C - 50;	Accumul-ative brake failed	impossible	needed	LWB_ACM_FS
LWB_ACM_FS	**Assessment:** LWB fails there is no brake on the left-side landing gear **Guidance:** no brake is available.	none	none	not_needed	not_needed	none

6.5 Experiment

In order to test the monitor and evaluate its ability to deliver the intended tasks, a simulator has been developed to:

- simulate the normal conditions of the two sub-systems, LWB and RWB, and the entire brake system over the normal, alternative and accumulative modes;
- inject faults of the basic components or pressure lines and simulate their effects on the behaviour of the sub-systems and the entire brake system;
- simulate conditions that follow application of corrective measures.

During normal conditions the pressure behaviour on the normal, alternative and accumulative lines is simulated according to the behaviour captured by Figure 6-6. The simulator generates ranges of pressure measurements that fall within the normal fluctuation of the hydraulic pressure of the three lines. While the brake system is in the normal mode, the simulator generates two pressure ranges, one belonging to the taxiing, take-off, climbing, approach and landing phases, which fluctuates over 3400 to 3800 PSI and the other belonging to the cruise phase, which fluctuates over 2700 to 3100 PSI. Similar ranges are also generated for the alternative and accumulative modes.

The simulator has also an ability to inject different faults and virtualise abnormal conditions, e.g. generating low pressure measurement (less than 1300 PSI) during the normal mode. The simulator can, moreover, execute corrective measures instructed by the monitor. For instance, when the pressure of the normal line drops abnormally, the monitor can control the conditions by instructing the simulator to switch to the alternative mode. At this point, the simulator generates measurements reflecting the following: (a) valves LN_CV and RN_CV are closed and valves LA_MV and RA_CV are opened; (b) pressure range of the alternative line, which is measured by pressure sensor A_PS.

In order to demonstrate the ability of the monitor to diagnose the injected faults, the simulator generates also measurements that provide the relevant symptoms with corresponding sensory measurements after injecting a fault. For example, the injection of a fault of the auto-brake valve RN_ABV would be followed by generating the following:

- Normal pressure range measured by N_PS_P.
- Valve RN_CV is open (RN_CV_PO = 1).
- Normal behaviour of the anti-skid valve RN_ASV. Such behaviour can be virtualised by generating anti-skid commanded pressure that is less than or equal to the input pressure of the valve (RN_ASV_IP ≥ RN_ASV_C).

To facilitate the evaluation of the response time and the prompt delivery of the monitoring tasks, each monitoring agent is provided with an ability to count (a) the time that is taken to achieve a monitoring cycle; (b) the interval that falls between the injection of a fault and announcing the corresponding alarm; (c) the interval that falls between the injection of a fault and diagnosing the underlying causes; (d) the interval that falls between the injection and controlling a fault.

Table 6-5 lists faults that are injected by the developed simulator to demonstrate the ability of the monitor to deliver the intended monitoring tasks and also to evaluate the effectiveness of those tasks. The table shows a number of possible faults of every component of the brake system. The injection of these faults results in verifying the occurrence of different failure events and simulating different failure scenarios, such as low pressure at an operative line and incorrect or no brake function is provided.

Table 6-5: Faults that are injected by the Developed Simulator.

Component	Injected Faults	Description
Pressure Line	Low pressure at the operative line.	Pressure at the three lines, normal, alternative and accumulative, is dropped under 1300 PSI due to a failure of the left and right hydraulic systems of the aircraft.
Sensor	Permanent functional failure of the sensor itself.	A sensor delivers permanent spurious measurements that lie out of the expected range or deliver non-linear drift or biased measurements.
Control Valve	Commission close command	A valve is closed inadvertently.
	Commission open command	A valve is opened inadvertently.
	Stuck between open and closed	A valve sticks half open.
	Stuck closed	A valve sticks closed and does not respond to open instructions.
	Stuck open	A valve sticks open and does not respond to close instructions.
Auto-brake Valve, Meter Valve and Anti-skid Valve	Non-regulate	A valve does not regulate pressure according to the demanded brake.
	Stuck closed	A valve sticks closed and does not respond to BSCU to pass the demanded pressure.
	Stuck open	A valve sticks open, passes extra pressure and does not respond to brake system control unit (BSCU) to pass the demanded pressure.

Simulations of two different failure scenarios have been selected to be discussed herein. The first scenario is simulated by injecting a fault of anti-skid valve RN_ASV, which results in a

"Failure of the normal line of RWB Sub-system". The second failure scenario is simulated by injecting fault of low pressure on the alternative line, which results in a "Failure of the alternative line". Along with simulating these scenarios the delivery of the monitoring tasks is also evaluated based on the effectiveness of (a) detecting, diagnosing and controlling faults; (b) the design of the operators' interface and also the announced alarm information.

6.5.1 First Injected Fault: Failure of the Normal Line of RWB Sub-system

Among the faults that could result in the failure of the normal line, such that the brake system transits to the alternative mode, is the fault of the anti-skid valve RN_ASV. The monitoring expression that verifies the occurrence of this event is as follows:

$$\begin{gathered} RN\_ASV\_OP > RN\_ASV\_C + 50 \\ OR \\ RN\_ASV\_OP < RN\_ASV\_C - 50; \end{gathered}$$

Once the occurrence of the above expression is verified, agent RWB_MAG perceives its state-transition table (Table 6-3) and achieves the following procedure:

- From the relevant ALARM attribute, agent RWB_MAG quotes "normal brake failed" and alarms the pilot.
- From the relevant CONTROLLING attribute, agent RWB_MAG opens valve RA_CV and closes valve RN_CV, to switch to the alternative line.
- From the relevant DIAGNOSIS attribute, agent RWB_MAG verifies the need for a diagnostic process. At this point and before applying the corrective measures, agent RWB_MAG updates the symptoms of the relevant diagnostic model with the relevant measurements; this has been illustrated by plan $L_1\_P_{17}$ and the associated pseudo code (Section 4.7.2).
- From the relevant NEW STATE attribute, agent RWB_MAG transits to a new state which is the failure state RWB_NM_FS. From this state, the pilot is provided with assessment, "normal line of RWB has failed" and guidance, "fault controlling is in progress".
- Agent RWB_MAG communicates state RWB_NM_FS to the parent agent (ABS_MAG).

Since a diagnostic process is needed, then before launching a new monitoring cycle, agent RWB_MAG retrieves the position of the relevant diagnostic model, which is shown by Figure 6-13. For the purpose of diagnosis, agent RWB_MAG exploits the diagnostic algorithm that was developed in Figure 3-27 in Chapter Three.

When agent ABS_MAG receives a message that conveys the RWB_NM_FS, it perceives its state-transition table (Table 6-2) and achieves the following procedure:

- As the current state is the ABS_NM_EFS, the received state results in verifying the occurrence of LWB_NMFS == true OR RWB_NMFS == true.
- From the relevant NEW STATE attribute, agent ABS_MAG transits to a new state which is the temporary degraded/failure state ABS_NM_TDFS. From this state the pilot is provided with assessment, “brake system is in a temporary failure”, and guidance, “fault controlling is in progress”.
- Agent ABS_MAG communicates the new state to the children (RWB_MAG and LWB_MAG).

When agents RWB_MAG and LWB_MAG receive the messages, each achieves a certain procedure, as follows:

Agent RWB_MAG perceives the state-transition table (Table 6-3) and achieves the following procedure:

- While the current state is the RWS_NM_FS, the received state results in verifying the occurrence of ABS_NM_TDFS == true.
- As the relevant ALARM, CONTROLLING and DIAGNOSIS attributes require no action then from the relevant NEW STATE attribute, agent RWB_MAG transits to a new error-free state RWB_AM_EFS. From this state the pilot is provided with assessment, “RWB is pressured by the alternative line”, and guidance, “apply manual brake”.
- Agent RWB_MAG communicates that state to parent agent (ABS_MAG).

Agent LWB_MAG perceives its state-transition table (Table 6-4) and achieves the following procedure:

- While the current state is the LWS_NM_EFS, the received state results in verifying the occurrence of ABS_NM_TDFS == true.
- As the relevant ALARM attribute shows “none”, no alarm is released.
- From the relevant CONTROLLING attribute, agent LWB_MAG opens valve LA_CV and closes valve LN_CV, to switch to the alternative line.
- As the relevant DIAGNOSIS attribute holds “not_needed”, no action is taken.
- From the relevant NEW STATE attribute, agent LWB_MAG transits to a new state which is the error-free state LWB_AM_EFS. From this state the pilot is provided with assessment, “LWB is pressured by the alternative line” and guidance, “apply manual brake”.
- Agent LWB_MAG communicates that state to the parent agent (ABS_MAG) and launches a monitoring cycle.

When agent ABS_MAG receives messages sent by agents LWB_MAG and RWB_MAG, it accordingly perceives its state-transition table (Table 6-2) and achieves the following procedure:

- While the current state is the ABS_NM_TDFS, the received state results in verifying the occurrence of LWB_AM_EFS == true AND LWB_AM_EFS == true.
- From the relevant NEW STATE attribute, agent ABS_MAG transits to a new state which is the error-free state ABS_AM_EFS. From this state the pilot is provided with assessment, "LWB and RWB are pressured by the alternative line" and guidance, "only manual brake is applicable".

After achieving the above procedures, the alternative mode of the entire brake system would be launched and the operational structure of that mode is as shown by Figure 6-15.

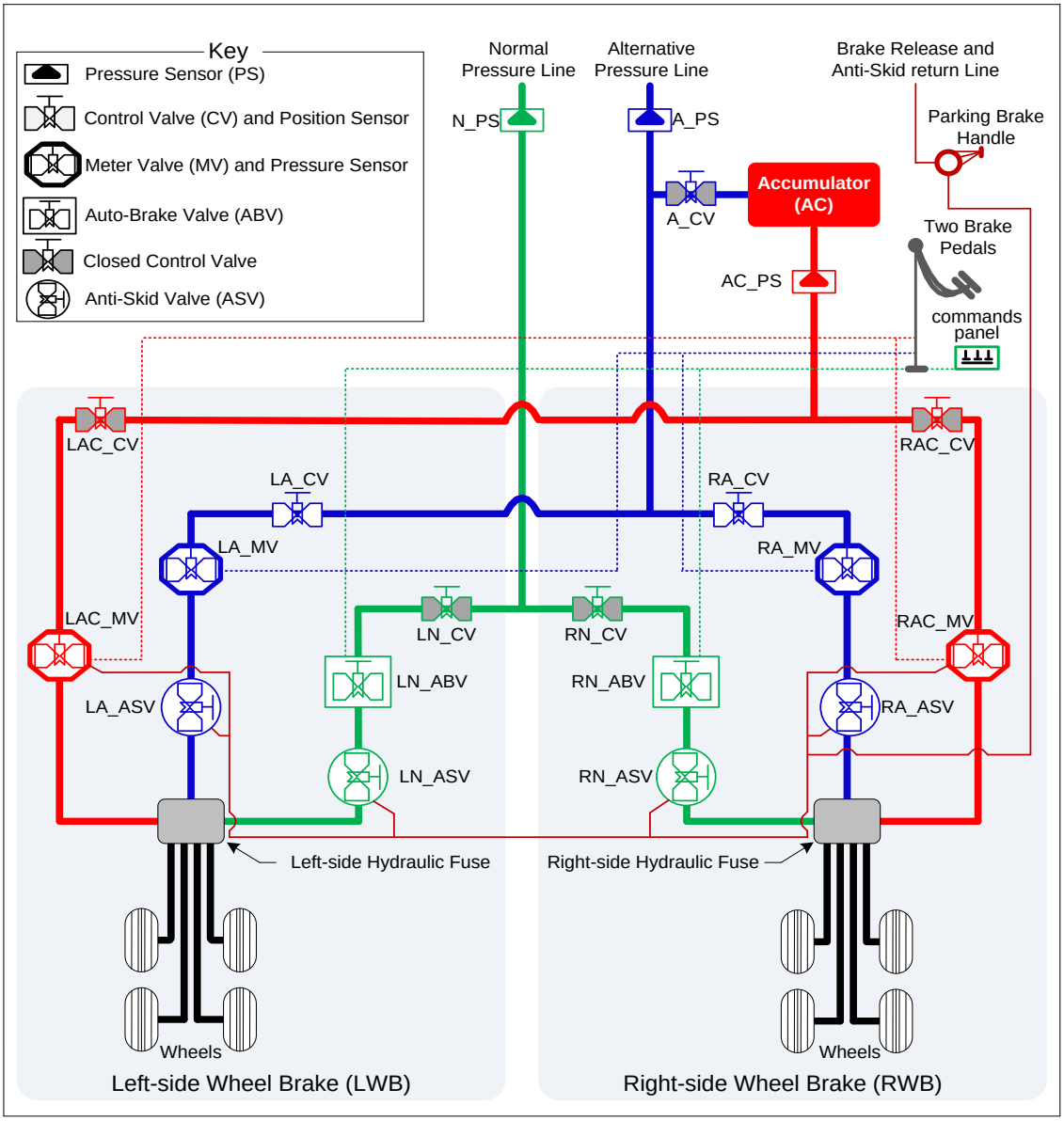


Figure 6-15: Operational Structure of the ABS during the Alternative Mode.

6.5.1.1 Fault Detection and Diagnosis of "Failure of Normal Line of RWB Sub-system"

As brake function is critical and required to be prompt, the monitoring expression that can verify the failure of the normal line has been formalised as a simple expression, with no filtration interval. However, the expression has considered the possible bias of the involved sensor to avoid the possibility of releasing a false alarm.

Over the injection of three different faults of the anti-skid valve RN_ASV (see Table 6-5), less than 0.5 seconds is taken to detect the fault. This is followed by one second to diagnose the fault. Practically, the strategy in which fault controlling is launched immediately after detecting the fault is the reason behind taking one second to diagnose the cause. However, this strategy is necessary to the prompt recovery of the brake function. The loss of the brake function for a longer interval could possibly result in disastrous consequences especially in the landing and take-off phases. It can be said, therefore, that the monitor of the brake system is able to deliver prompt fault detection and diagnose tasks.

6.5.1.2 Alarm Annunciation on the "Failure of Normal Line of RWB Sub-system"

As the alarm is announced immediately after verifying the occurrence of the failure, the pilot could respond by applying the manual brake. It is important to note that the pilot is informed on using the manual brake by the electronic guidance and there is no need to understand an alerting message from the Engine Indication and Crew Alerting System (EICAS) and achieve a manual corrective procedure. This would certainly result in reducing the workload of the pilot during emergency conditions and make exacerbation less likely. Figure 6-16 illustrates the pilot interface when the alarm, the assessment and the guidance on the occurrence of "Failure of Normal Line of RWB Sub-system" are presented.

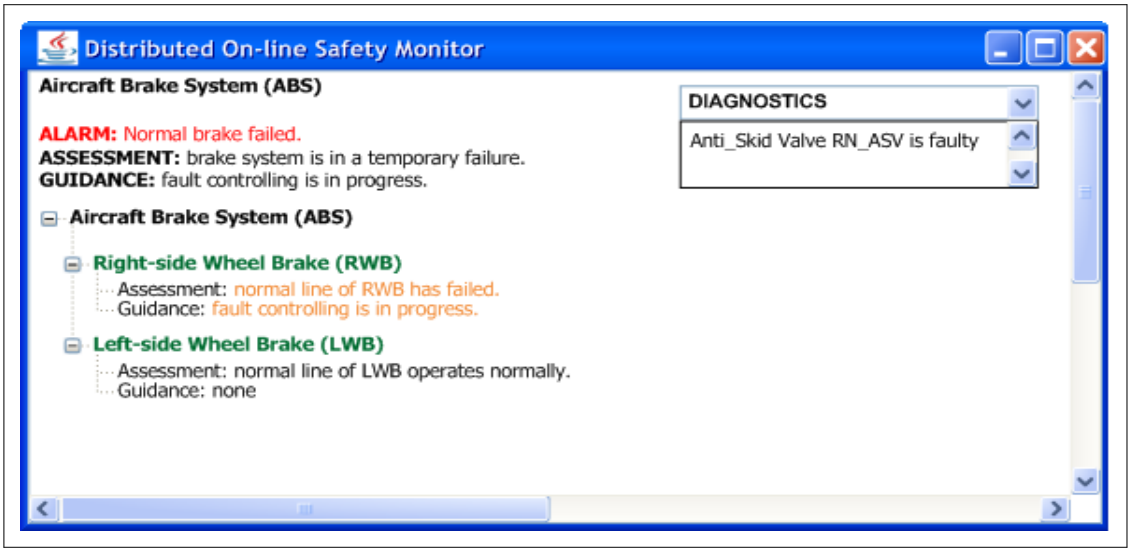


Figure 6-16: Operator's Interface after the Occurrence of Failure of the Normal Line of RWB.

It can be seen that the interface provides the pilot with informative alarm on the given conditions and at the same time offers an optional access to the diagnostic list, which holds the name of the diagnosed cause. The presentation of the interface seems adequate to understand the actual conditions of the brake system from abstracted and helpful information. More importantly, the interface states clearly that fault controlling is in progress and the pilot does not have to interfere or take any further action.

6.5.1.3 Fault Controlling of the "Failure of Normal Line of RWB Sub-system"

As mentioned earlier, the brake system tolerates any failure that occurs in the normal line by switching to the alternative line. Upon verifying the abnormal pressure at the output of the anti-skid valve (RN_ASV_OP), agent RWB_MAG closes control valve RN_CV and opens control valve RA_CV. Similarly, agent LWB_MAG closes control valve LN_CV and opens control valve LA_CV. As such, the normal line is isolated and the alternative line is activated and thus manual brake would be the only option for the pilot to apply brake function. This can be confirmed as assessment and guidance to the pilot as shown by Figure 6-17.

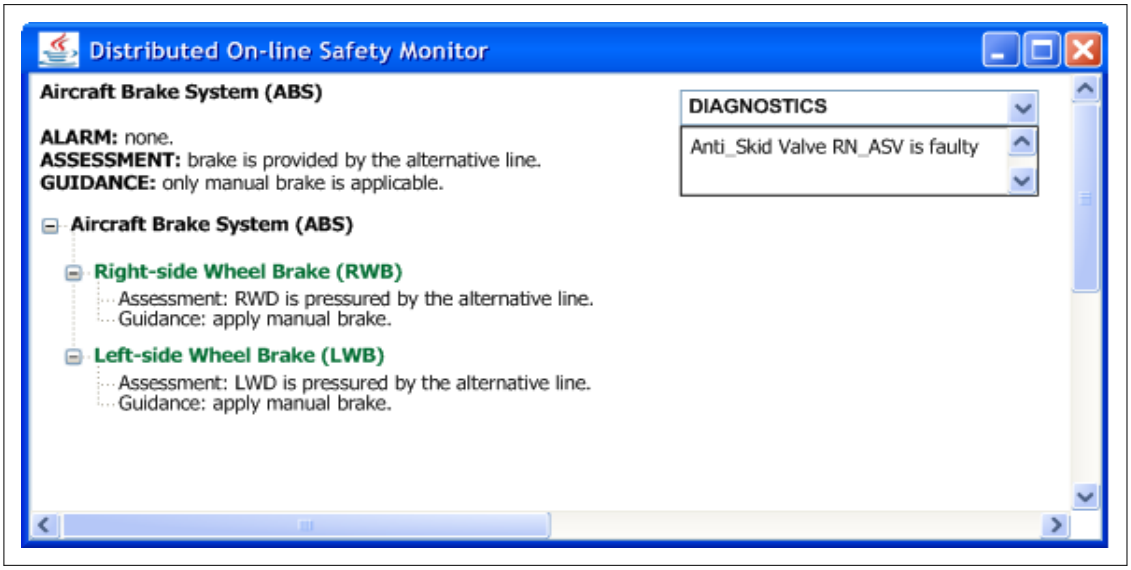


Figure 6-17: Operator's Interface Confirming Fault Controlling (Switching to the Alternative Mode).

Along with the progress of controlling the fault, the interface is updated to maintain the awareness of the pilot; see the difference between the interface of Figure 6-16 and the interface of Figure 6-17. In Figure 6-17 the pilot is informed on using the manual brake as the automatic brake function is not available due to the failure of the normal line. Therefore, it can be said that the fault controlling and alarm annunciations tasks integrate to control the fault and guide the operators. Such integration can assure the avoidance of any exacerbation during the emergency conditions.

6.5.2 Second Injected Fault: Failure of the alternative line

During the alternative mode, low pressure at the alternative line is detected by agent RWB_MAG. This is because pressure sensor A_PS is included in the monitoring model of the RWB sub-system, as shown in Figure 6-11, Figure 6-12 and Table 6-3. The monitoring expression that can verify the occurrence of this event is as follows:

$$A\_PS\_P - 1300 < 50\ PSI$$

Once the occurrence of the above expression is verified, agent RWB_MAG perceives its state-transition table (Table 6-3) and achieves the following procedure:

- While the current state is RWB_AM_EFS, from the relevant ALARM attribute, agent RWB_MAG quotes "low pressure at alternative line" and alarms the pilot.
- From the relevant CONTROLLING attribute, agent RWB_MAG opens valve RAC_CV and closes valve RA_CV; to switch to the accumulative line.
- From the relevant DIAGNOSIS attribute, agent RWB_MAG quotes "pressure at the alternative line is low" as this event and its underlying cause are in a one-to-one relationship.
- From the relevant NEW STATE attribute, agent RWB_MAG transits to a new state which is a failure state RWB_AM_FS. From this state the pilot is provided with assessment, "alternative line of RWD has failed" and guidance, "fault controlling is in progress".
- Agent RWB_MAG communicates that state to parent agent (ABS_MAG) and launches a monitoring cycle.

Upon receiving state RWB_AM_FS, agent ABS_MAG perceives its state-transition table (Table 6-2) and achieves the following procedure:

- While the current state is the ABS_AM_EFS, the received state results in verifying LWB_AM_FS == true OR RWB_AM_FS == true.
- From the relevant NEW STATE attribute, agent ABS_MAG transits to a new state, ABS_AM_TDFS. From this state the pilot is provided with assessment, "brake system is in temporary failure" and guidance, "fault controlling is in progress".
- Agent ABS_MAG communicates the ABS_AM_TDFS to child agents: RWB_MAG and LWB_MAG.

When agents RWB_MAG and LWB_MAG receive the messages, each achieves a certain procedure, as follows:

Agent RWB_MAG perceives its state-transition table (Table 6-3) and achieves the following procedure:

- While the current state is the RWS_AM_FS, the received state results in verifying ABS_AM_TDFS == true.
- As the ALARM, CONTROLLING and DIAGNOSIS attributes require no action, then from the relevant NEW STATE attribute, agent RWB_MAG transits to a new state, which is RWB_ACM_EFS. From this the pilot is provided with assessment, “RWB is pressured by the accumulative line” and guidance, “apply manual brake”.
- Agent RWB_MAG communicates state RWB_ACM_EFS to the parent agent (ABS_MAG) and launches a monitoring cycle.

Agent LWB_MAG perceives its state-transition table (Table 6-4) and achieves the following procedure:

- While the current state is the LWB_AM_EFS, the received state results in verifying ABS_AM_TDFS == true.
- As the relevant ALARM attribute shows “none”, an alarm is not released.
- From the relevant CONTROLLING attribute, agent LWB_MAG opens valve LAC_CV and closes valve LA_CV; to switch to the accumulative line.
- As the relevant DIAGNOSIS attribute is “not_needed”, agent LWB_MAG would not launch a diagnostic process.
- From the relevant NEW STATE attribute, agent LWB_MAG achieves a transition to a new state which is LWB_ACM_EFS. From this state the pilot is provided with assessment, “LWB is pressured by the accumulative line” and guidance, “apply manual brake”.
- Agent LWB_MAG communicates that state to the parent agent (ABS_MAG) and launches a monitoring cycle.

Upon receiving messages sent by agent LWB_MAG and RWB_MAG, the parent agent (ABS_MAG) perceives Table 6-2 and achieves the following procedure:

- While the current state is ABS_AM_TDFS, the received state results in verifying LWB_ACM_EFS == true AND RWB_ACM_EFS == true.
- From the relevant NEW STATE attribute, agent ABS_MAG achieves a transition to the corresponding state, which is ABS_ACM_EFS. From this state the pilot is provided with assessment, “LWB and RWB are pressured by the accumulative line” and guidance, “apply manual brake but anti-skid is unavailable”.

After achieving the above procedures, the accumulative mode of the entire brake system would be launched. Thus, the new operational structure of the ABS is as shown by Figure 6-18.

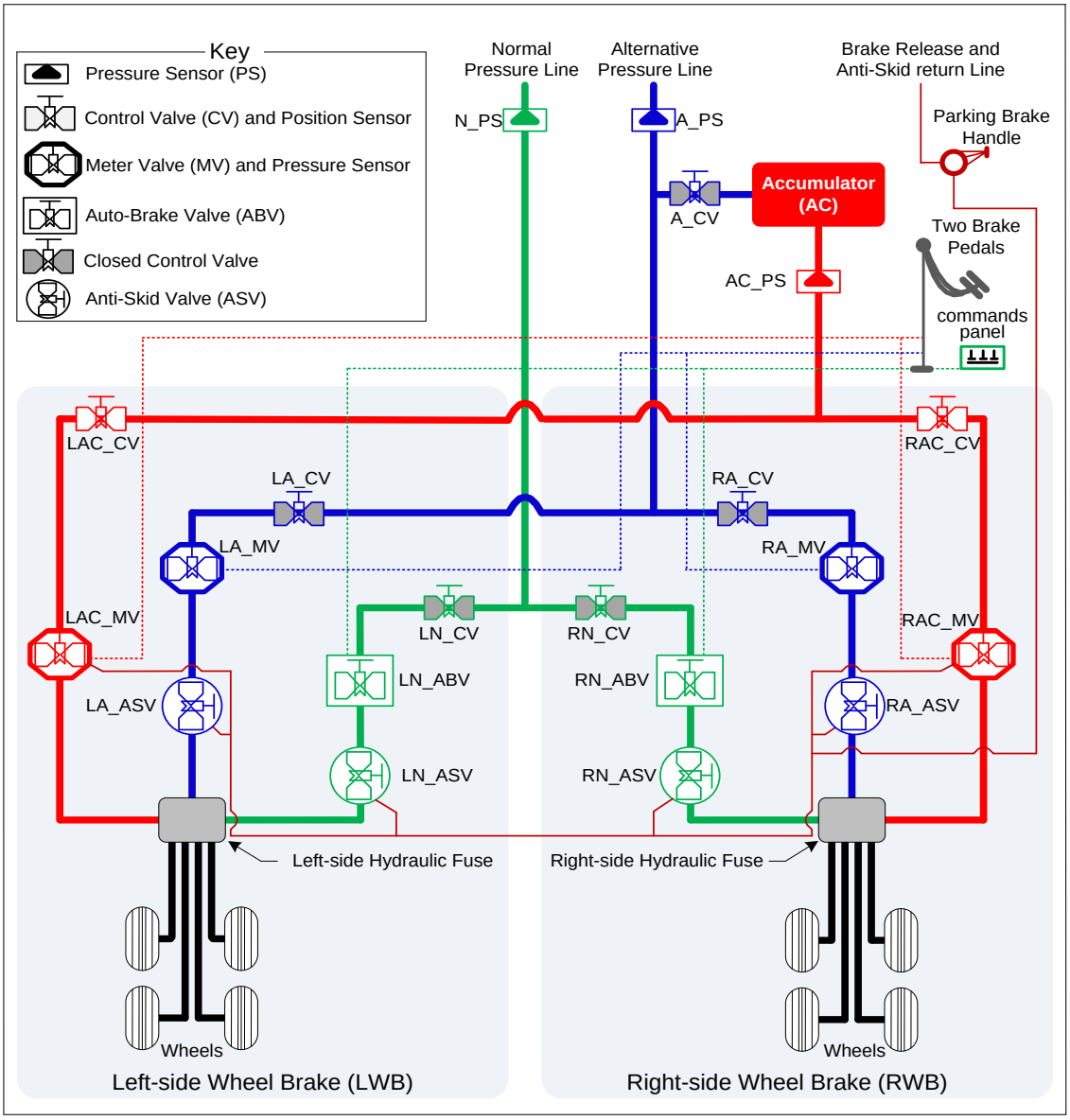


Figure 6-18: Operational Structure of the ABS during the Accumulative Mode.

6.5.2.1 Fault Detection and Diagnosis of "Failure of the alternative line"

As before, due to the high criticality of the brake function, the monitoring expression of the "failure of the alternative line" has no filtration interval. Thus, the occurrence of this event can be verified promptly after injecting the fault. However, to avoid releasing a false alarm, the expression considers the possible bias of pressure sensor A_PS_P.

Since the pressure is provided by a different system then a diagnostic process is not needed in this case. Accordingly, the relevant cell of the state-transition table (Table 6-3) encloses "pressure at the alternative line is low". After injecting the fault, an alarm on the occurrence of the event and diagnosis of the underlying cause are announced immediately; after less than 0.5 seconds. Therefore, it can be said that this response shows another evidence of the ability of the monitor to deliver prompt and effective fault detection and diagnosis tasks.

6.5.2.2 Alarm Annunciation on the Occurrence of "Failure of the alternative line"

Upon detecting low pressure at the alternative line, the alarm "Low pressure at the alternative line" is released immediately. The pilot is also provided with assessment and guidance on the temporary failure and ongoing controlling. Figure 6-19 illustrates the pilot interface after the occurrence of "Failure of the alternative line".

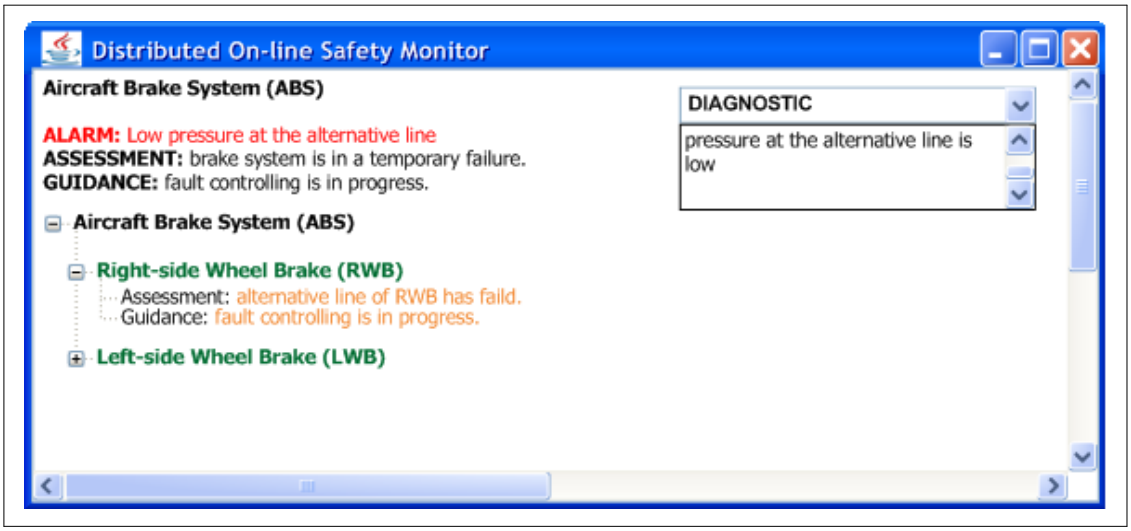


Figure 6-19: Operator Interface after the Occurrence of Failure of the Alternative Line.

The operator interface of Figure 6-19 shows, once more, the way in which the pilot is provided with alarm, assessment and guidance on the given conditions and offers at the same time an optional access to the diagnostic list. The interface seems helpful in presenting abstracted and adequate information to help the pilot to understand the actual conditions of the brake system. The interface states clearly that fault controlling is in progress and the pilot does not have to interfere or take any further action.

6.5.2.3 Fault Controlling after the Occurrence of "Failure of the alternative line"

As mentioned earlier, the aircraft brake system tolerates any failure that occurs in the alternative line by switching to the accumulative line. Upon verifying low pressure at the alternative line, the monitoring agent RWB_MAG closes valve RA_CV and opens valve RAC_CV. Similarly, the monitoring agent LWB_MAG closes valve LA_CV and opens valve LAC_CV. Thus, the alternative line is isolated and the accumulative line is activated. As the anti-skid is not applicable on this line, the pilot should be cautious and apply the manual brake while avoiding any possible skidding. This is confirmed to the pilot as shown by Figure 6-20.

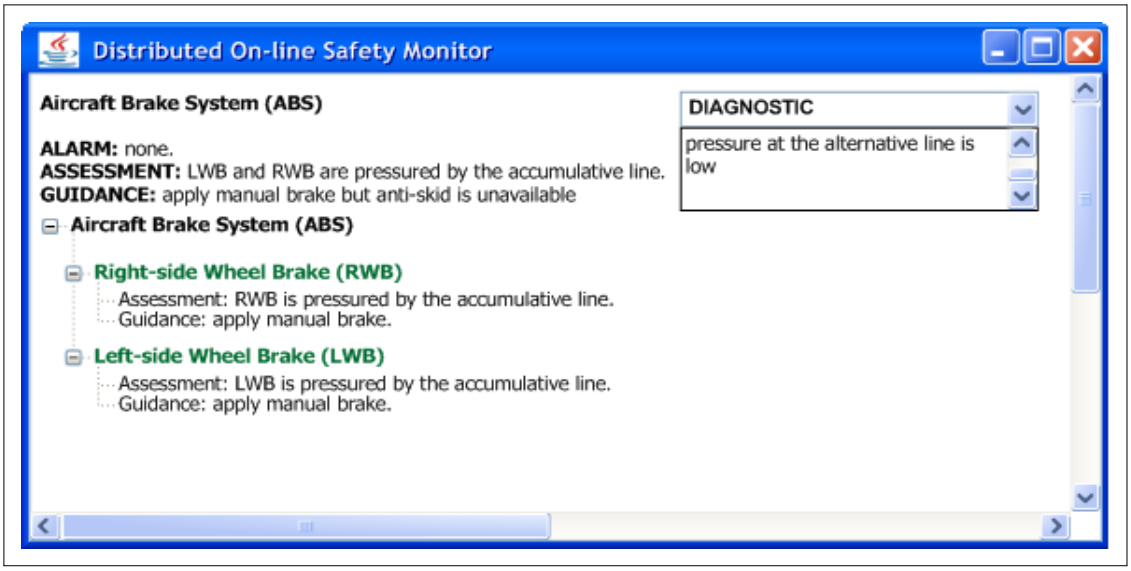


Figure 6-20: Operator Interface after Controlling the Failure of the Alternative Line.

Along with the progress of controlling the fault of the low pressure, the interface is updated promptly and the awareness of the pilot is maintained; see the difference between the interface of Figure 6-19 and the interface of Figure 6-20. Controlling this fault shows, once more, that the fault controlling and alarm annunciations tasks integrate to control the fault and guide the operators and thus any exacerbation of the emergency conditions can be avoided.

6.6 Summary

This chapter has demonstrated the way in which the distributed monitoring model is derived from an AADL safety assessment model. A case study on an aircraft brake system was also presented where, for experimental purposes, a simulator of the brake system was developed. It was shown that the monitoring concepts developed in this thesis can be applied in a different system across a different application domain, with the monitor still delivering the intended monitoring tasks. Moreover, using a set of simulated failure scenarios of different nature, this chapter has additionally shown the ability of the monitor to deliver the following tasks:

- Prompt fault detection and diagnosis.
- Effective alarm annunciation that is presented as well-organised alarm and multi-level assessment and guidance.
- Control the injected faults automatically, removing some of the demands for control that could otherwise have been unnecessarily posed on operators.

Chapter Seven

Evaluation

7.1 Introduction

This chapter evaluates the work undertaken and the results of this thesis. The evaluation strategy includes the following:

- Evaluation of work and results against the objectives set out in the Introduction chapter.
- Evaluation of the effectiveness of the tasks delivered by the developed monitor, based on the results obtained from the case studies in Chapter Five and Chapter Six.
- Discussion of the relative merits and limitations of the developed monitor vis-à-vis the earlier monitors discussed in Chapter Two.

7.2 Evaluation Against Objectives

This section revisits the objectives set out in the Introduction chapter and compares them with what has been achieved so far.

7.2.1 Development of a Distributed Monitoring Model

A distributed monitoring model has been defined as a constituent of the proposed monitor. It holds the monitoring knowledge that informs the on-line safety reasoning. Chapter Two has provided insight into the model-based and data-based approaches. The outcome was a decision to exploit model-based safety assessment models as a distributed monitoring model.

In Chapter Three, a set of behavioural and diagnostic models were defined to capture the monitoring knowledge and to support the delivery of the intended monitoring tasks. Typically, HiP-HOPS and AADL yield similar off-line assessment models that contain well formed knowledge about the operation and failure of a system. In order to bring this knowledge forward and make it useful in context of on-line safety monitoring an approach towards formalising this knowledge as an executable model was developed (section 3.6).

7.2.2 Development of a Multi-agent System that Operates on the Distributed Monitoring Model

The multi-agent system is the engine that executes the distributed monitoring model in real-time. It consists of a number of intelligent agents whose reasoning is informed by the distributed model to monitor the operational conditions of the monitored system. It has been developed in the following steps:

1- Investigating the benefits of incorporating intelligent agents:

In Chapter Two, the merits and limitations of both monolithic (i.e. centralised) and multi-agent monitors were discussed. For flexibility, performance, and scalability reasons, a decision was accordingly made to develop a distributed monitor based on a multi-agent system. The selection of a suitable intelligent agent paradigm and platform and the design of effective collaboration protocols therefore became significant issues of this research.

In Chapter Four, BDI agents were investigated, selected and introduced to develop the monitoring agents. Moreover, the hierarchical deployment and the required collaboration protocols were developed. Due to variety in roles and deployment implications, the design of the proposed multi-agent system includes several agents addressing different monitoring concerns at different levels of the hierarchy of the monitored system. The agents were designed and programmed accordingly, as discussed in section 4.3, section 4.4, section 4.5 and section 4.6.

2- Identifying the monitoring role of the agents:

The monitoring role of agents varies from one hierarchical level to another. As discussed in section 4.3, the role of the agents of level*1* is as follows:

- Track the behaviour of the monitored sub-systems via primary observation of parameters.
- Detect and diagnose faults.
- Provide the operators with alarm, assessment, guidance and prognosis.
- Collaborate with each other and with their parents to enable the delivery of consistent global monitoring tasks.

In section 4.4, the role of the agents of levels extending from level*2* to level*n-1* is identified as follows:

- Track the behaviour of the monitored sub-systems mainly via conditions reported by lower level agents.

- Provide the operators with multi-level assessment, guidance and prognosis;
- Collaborate across the hierarchy (as parent and children) to deliver global monitoring tasks.

In section 4.5, the role of the agent of level*n* was identified; it is quite similar to the role of an intermediate levels agent. The only difference is that the level*n* agent has no parent to collaborate with.

3- The development of monitoring algorithms:

In Chapter Three and Chapter Four, diagnostic and monitoring algorithms were developed. The role of diagnostic algorithms extends from traversing the diagnostic model and relating failure events to their underlying causes to prescribing corresponding corrective measures that should be taken according to the diagnosed causes, as shown in Figure 3-22 and Figure 3-27. Monitoring algorithms that can track the behaviour of the monitored system and its sub-systems have also been developed, as shown in section 4.7.

7.2.3 Application in Two Case Studies and Evaluation the Effectiveness of the Monitor

The approach has been applied to two case studies. Chapter Five includes a case study of an aircraft fuel system along with its HiP-HOPS safety assessment from which the distributed monitoring model was derived. For experimental purposes, a simulator was developed to animate the behaviour of the fuel system and experimentally inject faults and interact with the monitoring agents. Faults were systematically injected from a list of possible faults for all components and the ability of the monitor to deliver effective safety tasks was demonstrated and evaluated (see the next section for discussion of the results).

In a second application, Chapter Six includes a case study of an aircraft brake system along with its AADL safety assessment from which the distributed monitoring model was derived. It was shown that the approach is repeatable across applications in different domains. Again a variety of faults were systematically injected in a simulator and the ability of the monitor to deliver effective monitoring tasks was evaluated (see the next section for discussion of the results).

Exhaustive fault injection was confined to single faults only, because the combinations are too many to test exhaustively, so only a few were tested to examine the response of the monitor to multiple faults. Assuming that there are *n* possible component faults in a system, assessment of combinations of *m* of those failures requires that the analysis is repeated *n!/((n-m)!*m!* times. For large systems this would clearly be infeasible. For 500 faults there are approximately 12.500 combinations of two and for 1,000 faults there are approximately 500,000 combinations of two.

7.3 Evaluation of the Safety Monitoring Tasks Delivered by the Monitor

The case studies have shown the practical ability of the monitor to deliver the range of the intended monitoring tasks; prompt fault detection and diagnosis, effective alarm annunciation and fault controlling.

7.3.1 Evaluation of the Fault Detection and Diagnosis Task

One of the key requirements of safety monitoring is the early and accurate detection of the manifestation of anomalies in the system. Early and accurate detection was pursued by the following mechanisms incorporated in the design of the monitor:

- Identifying the hierarchical level for effective monitoring. Hierarchical levels of the monitored system were examined and level*1* was selected as an effective level to monitor the parameters, being the closest to the process with some knowledge about the behavioural context, as discussed in section 3.4.
- Enabling temporal operators in the formalisation of monitoring expressions which can be used for monitoring long-term trends and the timely verification of the occurrence of events and symptoms, as discussed in section 3.6.1.1. The evaluation of the monitoring expressions is also supported by efficient data storage mechanisms; effectively single and multi-measurement buffers that are updated systematically to hold up-to-date sensor measurements (as discussed in section 3.6.1.5).
- Providing the monitor with techniques for filtering and validating sensor measurements and for detection, diagnosis and control of sensor faults, as discussed in section 3.6.1.2 and section 3.6.1.3. The validation technique can compensate for a single sensor failure when there is hardware or functional redundancy but it does not totally solve the problem of validity of measurements particularly in the case of multiple dependent sensor failures (see also section 5.5.3.1).
- Reducing the reasoning load of the monitoring agents. This was achieved by making the agents able to track state transitions and cyclically monitor events that represent exits from the current state "only" (active events), as discussed in section 3.6.2, section 4.3, section 4.4, section 4.5 and section 4.6.
- Sharing sensor measurements globally among the agents of level*1*, as discussed in section 4.2.2.

In addition to exploiting the detection mechanisms above, accurate diagnosis was pursued via the following additional measures:

- Identifying the hierarchical level at which faults can be detected and their underlying causes can be diagnosed readily, as discussed in section 3.4.
- Developing diagnostic models from popular safety assessment techniques (HiP-HOPS and AADL) that support effective tracking of the error propagation paths, as discussed in section 3.6.3.
- Developing diagnostic algorithms that effectively couple between blind-depth-first and heuristic traverse strategies to track the error propagation paths and diagnose the underlying causes of faults, as discussed in section 3.6.3.

7.3.2 Evaluation of the Effective Alarm Annunciation Task

The effectiveness of alarm annunciation was pursued via the following measures:

- Defining effective thresholds whose violation represents an actual occurrence of an event.
- Suppressing unimportant and false alarms. Alarms are announced only on the occurrence of failure events of level*1*. This is supported by the incorporation of alarm clauses in level*1*'s state-transition tables (section 3.6.2) and also by designing and implementing agents of that level with the ability to announce alarms, as discussed in section 4.3 and section 4.7.2. Filtration and validation of sensor measurements have contributed, moreover, to eliminate any potential of false alarm annunciations, as discussed in section 3.6.1.2 and section 3.6.1.3 and demonstrated in section 5.5.1, section 5.5.3 and section 6.5.2.
- Prioritising alarm presentation. This is achieved by presenting the important alarms with high priority distinguished by different presentation colours and facilitating optional access to the assessment, guidance and diagnostic list; as shown, for example, in Figure 5-21 and Figure 5-24.
- Incorporating information that could help the operators to direct the system effectively in the presence of faults: multi-level assessment, guidance and prognosis, as shown in section 5.5.3.2.

7.3.3 Evaluation of the Fault Controlling Task

Effective control of faults was pursued via the following measures:

- Achieving (a) active fault-tolerant control (AFTC), as demonstrated in Chapter Five; (b) passive fault-tolerant control (PFTC), as demonstrated in Chapter Six.
- Controlling sensor failure, as discussed in section 3.6.1.3 and demonstrated in section 5.5.3.
- Supporting the operators with multi-level assessment, guidance and prognoses to take correct safety decisions in emergency conditions, as discussed in section 3.6.2 and demonstrated in section 5.5.3.2 and section 6.5.2.3.

7.4 Comparison of the Distributed On-line Safety Monitor with Early Developed Monitors

The key aim of the thesis was to explore the concept of a novel architecture for a distributed safety monitor operating on a safety assessment model that hopefully could address some problems of earlier monitors and deliver effectively a broad range of safety monitoring tasks. Thus, it is reasonable to compare it with the earlier monitors that were discussed in Chapter Two and weigh their monitoring merits and drawbacks against each other. The comparison can only be theoretical because access to these earlier monitors as well as the case studies that they were tested upon was not possible.

In this context, it seems important to note that in this section "*early monitors*" denotes early developed monitors, whereas "*distributed monitor*" denotes the monitor developed in this thesis. It is also important to note that to our knowledge, this thesis has developed the first distributed safety monitor that operates on results of safety assessment or safety studies of the system. However, there is a vast body of work on fault detection and diagnosis and therefore comparisons with other monitors which are conceptually close or attempt to deliver similar functions is possible.

7.4.1 Comparison with Monolithic Monitors

In the work of Papadopoulos (2003) and Peng *et al.* (2007), model-based monolithic (i.e. centralised) monitors are developed to monitor critical applications and deliver fault detection and diagnosis, alarm annunciation and fault controlling. These monitors resemble the distributed monitor in both the model-based approach and the deliverable monitoring tasks, but differ in their monolithic nature.

The monitoring model developed by Papadopoulos (2003) is quite similar in many ways to the monitoring model of the distributed monitor; it too can be derived from the HiP-HOPS assessment model. The author has indentified the following limitations:

- Vulnerability to sensor failures.
- The centralised nature of the architecture has limited the applicability of the monitor and made it unable to scale up to monitor large-scale and distributed systems, e.g. nuclear power plants or chemical processes.

The distributed monitor addresses to some extent these limitations via the following:

- The exploitation of techniques to validate sensor measurements, up to a certain extent. With careful use of functional and hardware redundancy, single sensor failures can be captured and masked.

- As it is based on a distributed concept in which monitoring agents are deployed according to the hierarchical architecture of the monitored system, the distributed monitor has an extendable architecture that at least in theory makes it able to scale up and monitor large scale systems.

The monolithic monitor developed by Peng *et al.* (2007) is also:

- Vulnerable to sensor failures.
- Unable to scale up to monitor large-scale and distributed systems
- Unable to provide the operators with prognosis

As mentioned above, the distributed monitor is provided with the required techniques and provisions that address these limitations.

The data-based monolithic monitors of Doan and Srinivasan (2008) and Rusinov *et al.* (2009) are developed to detect and diagnose faults of chemical processes. These monitors have a narrower scope than that of the distributed monitor and differ in their monolithic nature and data-based monitoring knowledge. In the case of Rusinov *et al.* (2009) the system has been tested on a large number of components, but the monitor has no provision to cope consistently with dynamic behaviour (unlike the distributed monitor, which can track and follow states) and does not deal with sensor failures.

7.4.2 Comparison with Multi-agent Monitors

The model-based multi-agent monitors of Eo *et al.* (2000, 2001) and Ren *et al.* (2006) are aimed to be applied to large-scale and distributed processes. They match the distributed monitor in the delivery of this task and the exploitation of the model-based approach and multi-agent system. These monitors differ from the distributed monitor in scope as they only focus on fault detection and diagnosis and they do not deliver the alarm organisation and fault controlling tasks.

Eo *et al.* (2000, 2001) suggests the following limitations in their work:

- The monitor does not incorporate local diagnostic models. It depends, rather, on global diagnostic decision taken among the monitoring agents. This does not work well when more than one agent has faulty monitored conditions and in such a case the delivery of erroneous diagnostics is quite possible.
- The monitor is vulnerable to sensor failures.
- The monitor lacks a protocol for effective collaboration among its monitoring agents. In the currently implemented protocol there is no direct communication among the agents and

messages may take a long time to be transmitted from one agent to another until they reach the intended agents. This delay could result in ineffective monitoring.

The distributed monitor addresses to some extent these limitations with the following provisions:

- Providing every monitoring agent of level*1* with a diagnostic algorithm and a number of diagnostic models, so they can achieve local diagnosis and deliver accurate diagnostics.
- Applying techniques to filter and validate sensor measurements and detect, diagnose and control single sensor failure.
- Developing collaboration protocols by which messages can be exchanged among the agents directly and with no delay.

Two limitations have been observed in the monitor of Ren *et al.* (2006). Both concern the diagnostic process and can be listed as follows:

- As the diagnosis is achieved globally and depends mainly on exchange of messages among the high level agent and lower level agents, this may place a heavy communication load on the higher level agent and consequently result in its late response.
- As the diagnostic decision is processed globally and based on identifying the anomalies among the consistent conditions, the appearance of a number of anomalous conditions could potentially mislead the diagnostic process.

As a precaution against such limitations, the distributed monitor has been provided with the following strategies:

- The communication load is reduced, as the monitoring agents of level*1* are provided with diagnostic models and algorithms so they achieve a local and independent diagnostic process.
- The diagnostic process is achieved based on local observations of every sub-system and it is not affected by anomalous conditions of other sub-systems.

The data-based multi-agent monitors of Niu *et al.* (2007) and Ng and Srinivasan (2010) are developed to detect and diagnose faults of dynamic chemical processes. They match the distributed monitor in the delivery of this task and exploitation of the multi-agent system and they differ in their data-based monitoring knowledge. The monitor of Niu *et al.* (2007) can detect and diagnose both single and multiple faults. Practically, this is an outcome of exploiting sensor fusion methods and also global fusion collaboration among the agents. Similarly, the distributed monitor is able to detect, diagnose and moreover control single and multiple faults (but not multiple dependent sensor failures). This has been materialised by providing agents of

level*1* with effectively formalised monitoring expressions and models to achieve local detection and diagnosis. Moreover, across the hierarchical levels, agents collaborate to achieve global reasoning over the entire monitored process.

7.5 Summary

As this chapter has demonstrated, the research has produced a novel system and fulfilled the objectives set out in Chapter One. Moreover, the distributed monitor has been theoretically at least shown to compare well with earliest monitors in terms of the range of the delivered tasks and in its ability to scale up, thanks to its distributed nature. The theoretical comparison with earlier work was focused on limitations mainly stated by the authors of that work, which this thesis attempted to address with a variety of conceptual and design measures taken at design stage. Not every problem was solved of course and there are limitations to this thesis which are discussed in the next chapter.

Chapter Eight

Conclusion

8.1 The Approach of Distributed On-line Safety Monitoring

The Introduction chapter of this thesis has stated the following hypothesis:

A distributed on-line safety monitor that operates on a model containing knowledge derived from safety assessments could:

a- deliver a wide range of effective monitoring tasks which extend from fault detection and diagnosis to effective alarm annunciation and fault controlling.
b- address the limitations associated with earlier safety monitors and deliver improvements in the effectiveness, timeliness, correctness, flexibility and scalability of on-line safety monitoring tasks.
c- have a generic architecture that would make it applicable in a variety of safety-critical systems and contexts, such as those used in numerous transport industries or industrial processes.

A distributed model-based safety monitor that operates on results of safety studies was indeed researched, developed and evaluated in this thesis, and was theoretically and practically (via case studies) tested against the above criteria.

a- As anticipated in the hypothesis above, the monitor delivers a wide range of monitoring tasks which span from fault detection and diagnoses, effective alarm annunciation and fault controlling. The use of safety assessment models brings a wealth of knowledge, which currently remains unexploited, into the on-line reasoning about the safety of the monitored system.
b- A key limitation of earlier relevant work on safety monitors (see Papadopoulos 2000, 2003) is the monolithic nature of the architecture of these monitors which affects the timeliness of the deliverable monitoring tasks, the flexibility of application and their scalability. This thesis developed a more flexible distributed monitor that addresses some of these problems by deploying a number of intelligent agents in a distributed hierarchical architecture monitor. It was shown that agents can perform fast local monitoring and collaborate with each other horizontally and vertically across the hierarchical architecture to solve more complex monitoring problems that require global reasoning. The architecture gives a degree of composability and therefore greater flexibility. Also, distribution of the monitoring reasoning and workload in a hierarchical architecture of multiple agents means greater scalability.

c- The architecture of the developed monitor is generic as it was envisaged. There is nothing specific to the application in this architecture and the agents are transposable; hence the monitor can be applied in other contexts. As long as the monitoring model complies with the specifications that have been provided for HiP-HOPS and AADL, the monitor will work. This was demonstrated in two case studies in two different applications domains.

The hypothesis has been investigated and validated through systematic steps that form the three key objectives of this thesis as set out in the introduction:

1- Development of a distributed monitoring model.
2- Development of a multi-agent system that operates on the distributed monitoring model.
3- Application in two case studies and evaluation of the developed monitor.

The first objective was achieved via research into state-of-the-art model-based safety assessment techniques and exploitation of the safety assessment models derived by application of techniques such as HiP-HOPS and AADL. The knowledge contained in those models was brought in the service of on-line safety monitoring following formalisation of the assessment models into an executable specification. A number of interrelated data structures have been declared to hold the monitoring models that conform to this specification. Models are parsed, interpreted and uploaded to the data structures according to their formal definitions (see E-BNF grammars). Fault detection, diagnostic and behavioural tracking algorithms have also been developed; they operate on formal models exploiting real-time observations of the monitored process.

The second objective was achieved by defining the engine of the monitor as a set of BDI monitoring agents, establishing collaboration protocols among them and deploying them hierarchically over the monitored system and its sub-systems. Every agent has been provided with its corresponding portion of the monitoring model. Monitoring models have informed the local and global reasoning on the on-line conditions of the monitored components. The collaboration protocols enable agents to reason as a group and though it cannot be claimed that the system exhibits full logical omniscience, global reasoning is made possible to enable solutions of problems that require integration of local monitoring models and monitored conditions.

On the achievement of the third objective, the monitor was evaluated via application to two case studies; an aircraft fuel system and an aircraft brake system. The case studies were performed on simulated models of those systems while Gaussian noise was added to signals to make the simulation more realistic and introduce the possibility of measurements temporarily going outside normal ranges, e.g. as a result of transient sensor failures. Systematic, exhaustive fault

injection was able to confirm that the monitor delivers correct monitoring tasks as long as the safety assessment model was correct. Exhaustive injection of multiple failures was not possible given the inevitable combinatorial explosion of possible combinations. However, some failures were injected. The results showed that independent failures were dealt with success while dependent failures[1] confused the monitor when their simultaneous occurrence had not been anticipated in the monitoring model.

Case studies have shown also that the monitoring concepts developed in this thesis can be applied in different safety assessment models (HiP-HOPS and AADL) and different systems across different application domains, with the monitor still delivering the intended monitoring tasks.

Practical comparison of results against other monitors was not possible. However, a theoretical discussion was included in the evaluation chapter. It was mainly focused on features of the monitor such as its distributed nature which addresses some of the limitations reported in earlier literature on safety monitors, potentially making application of this work more flexible and scalable.

8.2 Limitations and Suggestions for Further Research

In the context of the two experiments achieved in Chapters Five and Six, some limitations have been observed. Addressing these limitations could contribute towards improving effectiveness in the delivery of the monitoring tasks and thus a number of further research possibilities are suggested.

Experiments confirmed that the quality of monitoring and the inferences drawn by the monitor are strongly dependent on the *consistency*, *completeness* and *correctness* of the monitoring model. It was observed, for example, that an incorrect model may cause a monitor to lose track of the state of the system, or that combinations of dependent multiple failures that were unanticipated could cause the monitor to take inappropriate action that would be appropriate only if the failures happened separately and independently. A large part of the thesis was concerned with ways of achieving a good monitoring model. However, safety assessment techniques like HiP-HOPS and AADL can only address part of the problem. They can guarantee consistency and improve the quality of the model, but they cannot ensure that all the assumptions about the behaviour of the system are complete or correct. System simulation (or testing) can help analysts to identify areas where the monitoring model might be incomplete or incorrect. The validation of the monitoring model, though, is clearly an issue that creates much

[1] More information about the nature of the dependent failures can be found in section 2.4.1.

wider scope for further research, perhaps also via application of verification methods like "model checking".

Another limitation of the monitor is its very limited support for dealing with uncertain data. A technique of three-value logic enables evaluation of expressions in the context of incomplete information. The technique masks unknown truth values and, ultimately, computes the truth value of compound expressions from the known or unknown truth values of their constituent components. In practice, this technique allows the monitor to produce early alarms in the presence of (detected) sensor failures and incomplete process data without violating the logic specified in monitoring expressions. Although this technique is useful, uncertainty is a much broader problem. There can be uncertainty about the quality of measurements or about the implication relationship between observations made and conclusions reached in the course of the detection and diagnosis. This uncertainty is often inevitable, but the current monitor has no ways of representing this uncertainty or reasoning under this type of uncertainty. The incorporation of Bayesian probabilities and Bayesian network modelling is suggested for future investigation.

There are a number of improvements that could be made in the implementation of the monitor. Using a text file may not be the best format to support effective parsing and interpretation of the monitoring model. To address this issue, the suitability of Extensible Mark-up Language (XML) could be investigated in future research. The data structures are mainly implemented as vectors with references held as positions of referenced data in the vector where they belong. Use of self-referential structures would provide a more intuitive representation.

Finally, in concluding this thesis, we wish to say that the monitoring concepts and algorithms proposed here create opportunities for exploiting in real-time enormous knowledge about the behaviour of the system in conditions of failure. The proposed methods also incorporate principles of distributed reasoning that, in theory, could enable their application in large and complex systems. Small case studies have been performed to evaluate this approach. However, a conclusive evaluation of the real value and scalability of this approach could only be achieved in a much wider and more realistic context of application.

References

Abdelghani M. and Friswell M. I., 2001. A parity space approach to sensor validation. *In the Proceeding of the International Society for Optical Engineering (SPIE'2001)*. USA, Bellingham: Society of Photo-Optical Instrumentation Engineers, ISSN: 0277-786X CODEN, 4359 (1), 405-411.

Ahn S. J., Lee C. J., Jung Y., Han C., Yoon E. S. and Lee G., 2008. Fault diagnosis of the multi-stage flash desalination process based on signed digraph and dynamic partial least square. *Desalination*, 228(1-3), 68-83.

Aizpurua O., Caballero R., Galan R., and Jimenez A., 2009. A New Methodology for Massive Alarm Management System in Electrical Power Administration. *In the Proceeding of 7th Latin American and Caribbean Conference for Engineering and Technology*. June 2 - 5, 2009, San Cristobal, Venezuela.

Aldewereld H., Buzing P. and Jonker G., 2008. Multi-agent plan diagnosis and negotiated repair. *In the Proceeding of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS'08)*. May, 12-16, 2008, Estoril, Portugal, 1659-1660.

Anderson J., 2007. *Alarm handler user's guide* [online]. The University of Chicago, as Operator of Argonne National Laboratory, Deutches Elektronen-Synchrotron in der Helmholtz-Gemelnschaft (DESY) and Berliner Speicherring-Gesellschaft fuer Synchrotron-Strahlung mbH (BESSY). Available: http://www.slac.stanford.edu/comp/unix/package/epics/extensions/alh/alhUserGuide.pdf [Accessed February 23rd 2011].

ARP 4761, 1996. *Aerospace recommended practice: Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment*. Society of Automotive Engineering. Warrendale, PA: SAE.

Augustine A., Karunakaran K. and Murray J., 2008. *Alarm management strategy* [online]. USA: Lafarge Group, Lakeside Process Controls Ltd and Emerson Process Management. Available from: http://www.emersonprocessxperts.com/articles/ICR/Alarm-Management-Strategy.pdf [Accessed 16th February 2011].

Baraldi P., Cammi A., Mangili F. and Zio E., 2010a. An ensemble approach to sensor fault detection and signal reconstruction for nuclear system control. *Annals of Nuclear Energy*, 37(6), 778-790.

Baraldi P., Cammi A., Mangili F. and Zio E., 2010b. Local fusion of an ensemble of models for the reconstruction of faulty signals. *IEEE Transactions on Nuclear Science*, 57 (2), 793 - 806.

Baraldi P., Zio E., Gola G., Roverso D. and Hoffmann M., 2009. *A procedure for the reconstruction of faulty signals by means of an ensemble of regression models based on principal components analysis* [online]. USA: NPIC-HMIT 2009 Topical Meeting, April 5–9. Available from http://www.ife.no/publications/2009/mto/publication.2009-05-14.3706488274?set_language=en&cl=en [Accessed 19th February 2011].

BEA, 2011. *Safety investigation into the accident on 1 June 2009 to the Airbus A330-203, flight AF447* [online]. France: Bureau of Investigations and Analysis for the safety of civil aviation (BEA). Available: http://www.bea.aero/fr/enquetes/vol.af.447/note29juillet2011.en.pdf [Accessed 2nd August 2011].

Beard R.V., 1971. Failure Accommodation in Linear Systems through Self-Reconfiguration. *MIT, Cambridge*: MA, USA.

Bellifemine F., Caire G. and Greenwood D., 2007. *Developing multi-agent systems with JAD*. UK, Chichester: John Wiley & Sons Ltd.

Bhagwat A., Srinivasan R. and Krishnaswamy P. R., 2003. Multi-linear model-based fault detection during process transitions. *Chemical Engineering Science*, 58(9), 1649-1670.

Bieber P., Bougnol C., Castel C., Heckmann J., Kehren C., Metge S. and Seguin C., 2004. Safety assessment with Altarica - lessons learnt based on two aircraft system studies. *In the proceeding of the 18th IFIPWorld Computer Congress, Topical Day on New Methods for Avionics Certification*, Toulouse, France, 26 - 26 August 2004, Vol. 156, 505-510.

Bieber P., Bougnol C., Castel C., Heckmann J., Kehren C., Metge S., and Seguin C., 2004. Safety assessment with altarica - lessons learnt based on two aircraft system studies. *In the proceeding of the 18th IFIP World Computer Congress, Topical Day on New Methods for Avionics Certification*, Toulouse, France, 26 - 26 August 2004.

Bieber P., Castel C. and Seguin C., 2002. Combination of fault tree analysis and model checking for safety assessment of complex system. *In the Proceeding of the 4th European Dependable Computing Conferenc*. Springer, Berlin.Vol. 2485 of LNCS, 19-31.

Biggles Software, 2002. *Boeing 757-200/300 Systems Description* [online]. Biggles Software Available from: http://www.biggles-software.com/software/757_tech/contents.htm [Accessed on 4th of May 2011].

Billings C. E., 1991. *Human-centred aircraft automation: A concept and guidelines* [online]. Field CA United States: NASA Technical Memorandum TM-103885, NASA Ames Research Centre, Moffett. Available: http://www.archive.org/details/nasa_techdoc_19910022821 [Accessed July 15th 2010].

Boeing, 2000. *Boeing 747-441 operations manual: international lease finance corp* [online]. USA: The Boeing Company. D6-30151-425. Available: http://www.dream-air.ucoz.ru/new/pilotam/Boeing_747-400_Operations_Manual.pdf [Accessed April 15th 2011].

Bordini R. H., Braubach L., Dastani M., Seghrouchni A. E. F., Gomez-Sanz J. J., Leite J., O'Hare G., Pokahr A. and Ricci A., 2006. A survey of programming languages and platforms for multi-agent systems. *Informatica* 30(1), 33−44.

Bordini R. H., Hubner J. F. and Wooldridge M., 2007. *Programming multi-agent systems in AgentSpeak using Jason*. UK, Chichester: John Wiley & Sons Ltd.

Borner M., Straky H., Weispfenning T. and Isermann R., 2002. Model based fault detection of vehicle suspension and hydraulic brake systems. *Mechatronics*, 12(8), 999-1010.

Bozzano M. and Villafiorita A., 2003a. Improving system reliability via model checking: the FSAP/NuSMV-SA safety analysis platform. *In the Proceeding of the SAFECOMP 2003*. Edinburgh, Scotland, United Kingdom, 23-26 September, 2003. Lecture Notes in Computer Science (2788/2003), Berlin, Heidelberg: Springer-Verlag, 49-62.

Bozzano M. and Villafiorita A., 2003b. Integrating fault tree analysis with event ordering information. *In the Proceeding of the ESREL 2003*, Maastricht, Netherlands, 15-18 June, 2003, 247-254.

Bozzano M., Cavallo A., Cifaldi M., Valacca L. and Villafiorita A., 2003. Improving Safety assessment of Complex Systems: An Industrial Case Study. *In the Proceeding of the Formal Methods*. Lecture Notes in Computer Science (2805/2003), Berlin, Heidelberg: Springer-Verlag, 208-222.

Bratman M., 1987. Intentions, plans, and practical reason. *Harvard University Press*: Cambridge, MA.

Breen M., 2005. Experience of using a lightweight formal specification method for a commercial embedded system product line. *Requirements Engineering Journal*, 10(2):161–172.

Brown W. S., O'Hara J.M. and Higgins J.C., 2000. *Advanced alarm systems: revision of guidance and its technical basis* [online]. US Nuclear Regulatory Commission, Washington, DC (NUREG-6684). Available: http://www.bnl.gov/humanfactors/files/pdf/NUREG_CR-6684.pdf [Accessed February 22nd 2011].

Chambers L., 2005. A hazard analysis of human factors in safety-critical systems engineering. *In the proceeding of the 10th Australian Workshop on Safety Related Programmable Systems (SCS'05)*, Darlinghurst, Australia, 27-41.

Chang C. T. and Hwang J. I., 1998. Simplification techniques for EKF computations in fault diagnosis-suboptimal gains. *Chemical Engineering Science*, 53 (22), 3853-3862.

Chang Y., 2004. Application of sequential probability ratio test to computerized criterion-referenced testing. *Sequential Analysis*, 23, 45-61.

Chen L. W. and Modarres M., 1992. Hierarchical decision process for fault administration. *Computers & Chemical Engineering*, 16(5), 425-448.

Chen T. L. and You R. Z., 2008. A novel fault-tolerant sensor system for sensor drift compensation. *Sensors and Actuators A: Physical*, 147(2), 623-632.

Cheng S. and Pecht M., 2007. Multivariate state estimation technique for remaining useful life prediction of electronic products. *In the Proceeding of the AAAI Fall Symposium on Artificial Intelligence for Prognostics*. Arlington, VA, November 2007, 26-32.

Chung D. T., Modarres M. and Hunt R. N. M., 1989. GOTRES: an expert system for fault detection and analysis. *Reliability Engineering & System Safety*, 24(2), 113-137.

Clark k. and McCabe F., 2003. Go! for multi-threaded deliberative agents. *In the Proceeding of the 1st International Workshop on Declarative Agent Languages and Technologies (DALT03)*, Melbourne, Australia, 14-18 July 2003, 54-75.

Clark R. N., 1978. Instrument fault detection. *IEEE Transactions on Aerospace and Electronic Systems*, 14(3), 456-465.

D'Inverno M., Kinny D., Luck M. and Wooldridge M., 1997. A formal specification of dMARS. *In the Proceeding of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, 24−26 July 1997, Springer, Berlin, Vol. 1365 of LNAI, 155-176.

Dastani M. and Gomez-Sanz J., 2006. Programming multi-agent systems. *Knowledge Engineering Review*, 20(2), 151-164.

Davey E. C. and Feher M. P. and Guo K. Q., 1995. An improved annunciation strategy for CANDU plants. *Presented in the American Nuclear Society Conference and Embedded Topical Meeting "Computer-based Human Support Systems:Technology, Methods and Future"*. June 25-29, 1995, Philadelphia, Pennsylvania.

Davis R. 1984. Diagnostic reasoning based on structure and behaviour. *Artificial Intelligence*, 24(3), 347-410.

Davis R. and Hamscher W., 1992. Model based reasoning: troubleshooting. In: Hamscher W., Console L., de Kleer J., eds. *Readings in Model-based Diagnosis*, San Francisco, CA, USA, Morgan Kaufman, 3-28.

Debruyne V., Simonot-Lion F. and Trinquet Y., 2005. EAST-ADL - an architecture description language validation and verification aspects. *In the Proceeding of the IFIP International Federation for Information Processing*, Springer, Boston, USA, Vol. 176/2005, 181-195.

Delta Virtual Airlines, 2003. *Boeing 767-200/300ER/400ER operating manual* [online]. 3rd ed., Delta Virtual Airlines. Available: http://www.deltava.org/library/B767%20Manual.pdf [Accessed 5th March 2011].

Dheedan A., Papadopoulos Y., 2010. Multi-agent safety monitoring system. *In the Proceeding of the 10th IFAC Workshop on Intelligent Manufacturing Systems (IMS'10)*, 1-2 July 2010, Lisbon, Portugal, 93-98.

Dheedan A., Papadopoulos Y., 2011. Model-based distributed on-line safety monitoring. *In the Proceeding of the 3rd International Conference on Emerging Network Intelligence (EMERGING 2011)*, 20-25 November 2011, Portugal, Lisbon, 7 pages.

Dheedan A., Papadopoulos Y. and Davis D., 2012-in print. Distributed On-line Safety Monitor Based on Assessment Model and Multi-agent System. *In the Proceeding of the 8th International Conference on Computer Science & Information Systems*, Greece, Athens, 21-24 May 2012, 9 pages, to be published as book chapter.

D'Inverno M., Luck M., Georgeff M., Kinny D. and Wooldridge M., 2004. The dMARS architecture: a specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9 (1-2), 5-53.

Doan X. and Srinivasan R., 2008. Online monitoring of multi-phase batch processes using phase-based multivariate statistical process control. *Computers & Chemical Engineering*, 32(1-2), 230-243

Dong G., Chongguang W., Beike Z. and Xin M., 2010. Signed directed graph and qualitative trend analysis based fault diagnosis in chemical industry. *Chinese Journal of Chemical Engineering*, 18(2), 265-276.

Du X., Dixon, R. and Goodall R. M., and Zolotas A. C., 2010. Modelling and control of a high redundancy actuator. *Mechatronics*, 20(1), 102-112.

Dugan J. B., Sullivan K. J. and Coppit D., 1999. Developing a high-quality software tool for fault tree analysis. *In the Proceeding of the 10th International Symposium on Software Reliability Engineering.* Boca Raton, Florida, 1-4 November 1999, 222-231.

El-Mezyani T., Dustegor D., Srivastava S. K. and Cartes D. A., 2010, Parity space approach for enhanced fault detection and intelligent sensor network design in power systems. *In the Proceeding of the IEEE'2010 Conference on Power and Energy Society General Meeting*. Minneapolis, MN, 25-29 July 2010, 1-8.

Eo S. Y., Chang T. S., Lee B., Shin D. and Yoon E. S., 2000. Cooperative problem solving in diagnostic agents for chemical processes. *Computers & Chemical Engineering*, 24 (2-7), 729-734.

Eo S. Y., Chang T. S., Lee B., Shin D. and Yoon E. S., 2001. Function-behavior modeling and multi-agent approach for fault diagnosis of chemical processes. *Computer Aided Chemical Engineering*, 9 (2001), 645-650.

Feiler H. P. and Rugina A., 2007. *Dependability Modelling with the Architecture Analysis and Design Language (AADL)* [online], USA: Software Engineering Institute and Carnegie

Mellon University, CMU/SEI-2007-TN-043. Available from: ftp://ftp.sei.cmu.edu/pub/documents/07.reports/07tn043.pdf [Accessed November 7th 2010].

Feiler H. P., Gluch P. D. and Hudak J. J., 2006. *The Architectural Analysis and Design Language (AADL): An Introduction*, USA: Software Engineering Institute and Carnegie Mellon University, CMU/SEI-2006-TN-001. Available: http://i12www.ira.uka.de/~engelc/lehre/seminarSS07/material/aadlIntro.pdf [Accessed November 7th 2010].

Felkel L., Grumbach R. and Saedtler E., 1978. Treatment, analysis and presentation of information about component faults and plant disturbances. *In the Symposium on Nuclear Power Plant Control and Instrumentation, IAEA-SM-266/40,* 340-347.

Ferguson R. C., Tate R. and Thompson H. C., 2006. Implementing space shuttle data processing system concepts in programmable logic devices, *In the MAPLD International Conference,* September 26-28, 2006 Washington, D.C.

Finin T., Weber J., Wiederhold G., Genesereth M., Fritzson R., McKay D., McGuire J., Pelavin R., Shapiro S. and Beck C., 1993. *Specification of the KQML agent communication language* [online]. DARPA Knowledge Sharing Initiative External Interfaces Working Group. Available from: www.cs.umbc.edu/kqml/kqmlspec.ps [Accessed January 10th 2010].

FIPA, 2002. *FIPA contract net interaction protocol specification* [online]. Technical Report 00029, Foundation for Intelligent Physical Agents. Available from: http://www.fipa.org/specs/fipa00029/SC00029H.pdf [Accessed November 7th 2010].

Fisher M., 2004. Temporal development methods for agent-based systems. *Autonomous Agents and Multi-Agent Systems* 10(1), 41−66.

Funk K., 2009. A Methodology and Tools for the Prospective Identification of NextGen Human Factors Issues. *In the Proceeding of the 15th International Symposium on Aviation Psychology*, Dayton, OH, April 27 – 30, 2009, 106-111.

Ghariani A., Toguyeni A. K. A. and Craye E., 2002. A functional graph approach for alarm filtering and fault recovery for automated production systems. *In the proceeding of the 6th International Workshop on Discrete Event Systems (WODES'02)*. 02-04 October 2002, Zaragoza, Spain, 289-294.

Giarratano J. and Riley G., 2005. E*xpert systems, principles and programming*. 4th ed. Bosto: PWS publishing company.

Giret A., Argente E., Valero S., Gomez P. and Julian V., 2005. Applying multi-agent system modelling to the scheduling problem in a ceramic tile factory. *In the proceeding of the International Mass Customization Meeting: Concepts-Tools-Realization, 2005 (IMCM'05),* June 2-3, 2005 Klagenfurt, Austria, 151-162.

Grunske L. and Strabe H., 2003. Transformational patterns for the improvement of safety properties in architectural specification. *In the Proceeding of the 2nd Nordic Conference on Pattern, Languages of Programs (VikingPLoP 03)*, Bergen, Norge.

Halpern J. Y. and Pucella R., 2011. Dealing with logical omniscience: Expressiveness and pragmatics. *Artificial Intelligence*, 175 (2011), 220–235.

Han H., Gu B., Wang T. and Li Z. R., 2010. Important sensors for chiller fault detection and diagnosis (FDD) from the perspective of feature selection and machine learning. *International Journal of Refrigeration*, 34(2), 586-599.

Hashemian H. M., 2011. On-line monitoring applications in nuclear power plants. *Progress in Nuclear Energy*, 53(2), 167-181.

Hashemian H. M., 2004. History of on-line calibration monitoring developments in nuclear power plants. *Presented at the Technical Meeting on Increasing Instrument Calibration Interval Through On-Line Calibration Technology, International Atomic Energy Agency (IAEA)*, 27-29 September, 2004, Halden, Norway.

Heimerdinger W. L. and Weinstock C. B., 1992. *A conceptual framework for system fault tolerance*. Technical Report CMU/SEI-92-TR-033 ESC-TR-92-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213.

Hickling E., 1994. Ergonomics and engineering aspects of designing an alarm system for a modem nuclear power plant. In Stanton N., ed. *Human factors in alarm system design.* London: Taylor and Francis.

Hill D. A., 1993. Deep knowledge approach to aircraft safety management. *In the Proceeding of the 13th international conference in Artificial Intelligence and Expert Systems*. ISSN 2-906-899-879, 355-365.

Hines J. W. and Davis E., 2005. Lessons learned from the U.S. nuclear power Plant on-line monitoring programs. *Article Progress in Nuclear Energy*, 46(3-4), 176-189.

Hogg D. N., Folleso K., Volden F.S. and Torralba B., 1995. Development of a situation awareness measure to evaluate advanced alarm systems in nuclear power plant control rooms. *Ergonomics*, 38**(**11), 2394-2413.

Holder B. E., 2003. *Improving the Boeing Quick Reference Handbook* [online]. Aviation System Safety, Boeing Commercial Airplanes. Available: http://humansystems.arc.nasa.gov/eas/download/EAS_Symposium_Presentations/Boeing_QRH.pdf [Accessed 2nd March 2011].

HSE, 2000. *Better alarm handling, HSE Information Sheet* [online]. UK: Health and Safety Executive, chemical sheet No 6. Available from: http://www.hse.gov.uk/pubns/chis6.pdf, [Accessed 13th February 2011]

Hu W., Starr A. G. and Leung A. Y. T., 2003. Operational fault diagnosis of manufacturing systems. *Journal of Materials Processing Technology*, 133(1-2), 108-117.

Huang F., Lee Y., Hwang S., Yenn T., Yu Y., Hsu C. and Huang H., 2007. Experimental evaluation of human–system interaction on alarm design. *Nuclear Engineering and Design*, 237(3), 308-315.

Huerta M. A., Fernandez B. and Koutanoglu E., 2007. Manufacturing multi-agent system for scheduling optimization of production tasks using dynamic genetic algorithms. *In the Proceeding of the IEEE International Symposium on Assembly and Manufacturing, 2007 (ISAM '07)*, 22-25 July 2007 Ann Arbor, Texas at Austin, Austin, 245-250.

Hwang S., Lin J., Liang G., Yau Y., Yenn T. and Hsu C., 2008. Application control chart concepts of designing a pre-alarm system in the nuclear power plant control room. *Nuclear Engineering and Design*, 238(12), 3522-3527.

IAEA, 2008. *On-line monitoring for improving performance of nuclear power plants part 2: process and component condition monitoring and diagnostics* [online]. Vienna: International Atomic Energy Agency (IAEA), STI/PUB/1323. Available: http://www-pub.iaea.org/MTCD/publications/PDF/Pub1323_web.pdf [Accessed 7th March 2011].

Isermann R., 1993. Fault diagnosis of machines via parameter estimation and knowledge processing-Tutorial paper. *Automatica*, 29(4), 815–835.

Jang G., Seong D., Keum J. and Park H., 2008. Kim Y. The design characteristics of an advanced alarm system for SMART. *Annals of Nuclear Energy*, 35(6), 1006-1015.

Jiang J., 2005. Fault-tolerant control systems-An introductory overview. *Automatica SINCA*, 31(1), 161–174.

Johnson D. M., 1996. A review of fault management techniques used in safety-critical avionic systems. *Progress in Aerospace Sciences*, 32(5), 415-431.

Joshi A. and Heimdahl M. P. E., 2005. Model-based safety analysis of Simulink models using SCADE design verifier. *In the Proceeding of the 24th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, Springer, Lecture Notes in Computer Science, 2005, Volume 3688/2005, 122-135, DOI: 10.1007/11563228_10.

Joshi A., Heimdahl M. P. E., Miller S. P. and Whalen M. W., 2006. *Model-Based Safety Analysis*[online]. NASA Centre. USA: Minneapolis, Minnesota, 55455. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20060006673_2006007118.pdf [Accessed November 06, 2010].

Joshi A. and Heimdahl M. P. E., 2007. Behavioral fault modeling for model-based safety analysis. *In the Proceeding of the 10th IEEE High Assurance Systems Engineering Symposium (HASE '07), IEEE Computer Society*, 3-5 December 2008, Washington, DC, USA. 199-208.

Joshi A., Binns P. and Vestal S., 2007a. Automatic generation of fault trees from AADL models. *Presented at Workshop on Architecting Dependable Systems of The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. Edinburgh, UK.

Joshi A., Binns P. and Vestal S., 2007b. Automatic generation of static fault trees from AADL models. *In the proceeding of the Architecting Dependable Systems (DSN)*, June 2007, UK, Edinburgh, Scotland.

Kaistha N. and Upadhyaya B. R., 2001. Incipient fault detection and isolation of field devices in nuclear power systems using principal component analysis. *Nuclear Technology*, 136, 221-230.

Kelly E. M. and Bartlett L. M., 2006. Application of the digraph method in system fault diagnostics. *In the Proceeding of the 1st International Conference on Availability, Reliability and Security (ARES'06), IEEE Computer Society*, 20-22 April 2006, Vienna, Austria, 693-700.

Kim I. S. and Modarres M., 1987. Application of goal tree-success tree model as the knowledge-base of operator advisory systems. *Nuclear Engineering and Design*, 104 (1), 67-81.

Kim I. S., Modarres M. and Hunt R. N. M., 1988. *A Model-based approach to on-line process disturbance management: the models* [online]. USA: System Research Centre: University of Maryland, SRC TR 88-111. Available from: http://drum.lib.umd.edu/bitstream/1903/4837/1/TR_88-111.pdf [Accessed August 4th 2011].

Kim I. S., Modarres M. and Hunt R. N. M., 1990. A model-based approach to on-line disturbance management: the models, *Reliability Engineering and System Safety*, 28(3), 265-305.

Kim I. S., 1992. Computer-based diagnostic monitoring to enhance the human-machine interface of complex processes. *In the Proceeding of the Power Plant Dynamics, Control and Testing Symposium*. 27-29 May 1992, in Knoxville, TN, United States.

Kim I. S., 1994. Computerised systems for on-line management of failures: a state-of-the-art discussion of alarm systems and diagnostic systems applied in the nuclear industry. *Reliability Engineering & System Safety*, 44(3), 279-295.

Kohda T., Matsumoto S. and Nakagawa M., 2008. Risk analysis of phased-mission systems with multiple failure modes. *In the proceeding of the Reliability and Maintainability Symposium (RAMS)*, 28-31 Jan 2008, Las Vegas, 15 - 20.

Langton R., Clark C., Hewitt M. and Richards L., 2009. *Aircraft Fuel Systems*. London, UK: Wiley.

Lapp S. A. and Powers G. A., 1977. Computer-aided synthesis of fault trees. *IEEE Transactions on Reliability*, 26 (1), 2-13.

Larsson J. E., 1994. Diagnostic reasoning strategies for means-end models. *Automatica*, 30(5), 775-787

Larsson J. E., 1996. Diagnosis based on explicit means-end models. *Artificial Intelligence*, 80(1), 29-93.

Larsson J. E., 2002. Diagnostic reasoning based on means-end models: experiences and future prospects. *Knowledge-Based Systems*, 15(1-2), 103-110.

Lasnier G., Robert T., Pautet L. and Kordon F., 2010. Architectural and behavioural modelling with AADL for fault tolerant embedded systems. *In the proceeding of the 13th IEEE international symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, 5-6 May 2010, Carmona, Seville, 87-91.

Lee J., Kim J., Park J., Hwang I. and Lyu S., 2010. Computer-based alarm processing and presentation methods in nuclear power plants. *In the Proceeding of the World Academy of Science, Engineering and Technology*. Library of Congress, Electronic Journals Library, 65(2010), 594-598.

Lopez I. and Sarigul-Klijn N., 2010. A review of uncertainty in flight vehicle structural damage monitoring, diagnosis and control: Challenges and opportunities. *Progress in Aerospace Sciences*, 46 (7), 247-273.

Lu N., Wang X., 2007. SDG-based HAZOP and fault diagnosis analysis to the inversion of synthetic ammonia. *Tsinghua Science & Technology*, 12(1), 30-37.

Lupton L. R., Lapointe P. A. and Guo K. Q., 1992. Survey of international developments in alarm processing and presentation techniques. *In the Proceeding of the International Symposium on Nuclear Power Plant Instrumentation & Control*. Tokyo, Japan.

Ma J. and Jiang J., 2011. Applications of fault detection and diagnosis methods in nuclear power plants: A review. *Progress in Nuclear Energy*, 53(3), 255-266.

MacGregor J. F. and Kourti T., 1995. Statistical process control of multivariate processes. *Control Engineering Practice*, 3(3), 403-414.

Mangina E., 2005. Intelligent agent-based monitoring platform for applications in engineering. *International Journal of Computer Science & Applications*, 2(1), 38-48.

Marshall G. and Chapman D., 2002. *Resilience, reliability and redundancy* [online]. United Kingdom: Copper Development Association- IEE Endorsed Provider. Available from: http://www.copperinfo.co.uk/power-quality/downloads/pqug/41-resilience-reliability-and-redundancy.pdf [Accessed November 06, 2010].

Mascardi V., Martelli M. and Sterling L., 2004. Logic-based specification languages for intelligent software agents. *Theory and Practice of Logic Programming*, 4(4), 429−494.

Maurya M. R., Paritosh P. K., Rengaswamy R. and Venkatasubramanian V., 2010. A framework for on-line trend extraction and fault diagnosis. *Engineering Applications of Artificial Intelligence*, 23(6), 950-960.

Maurya M. R., Rengaswamy R. and Venkatasubramanian V., 2005. Fault diagnosis by qualitative trend analysis of the principal components. *Chemical Engineering Research and Design*, 83 (9), 1122-1132.

Maurya M. R., Rengaswamy R. and Venkatasubramanian V., 2006. A signed directed graph-based systematic framework for steady-state malfunction diagnosis inside control loops. *Chemical Engineering Science*, 61(2006), 1790–1810.

Maurya M. R., Rengaswamy R. and Venkatasubramanian V., 2007. A signed directed graph and qualitative trend analysis-based framework for incipient fault diagnosis. *Chemical Engineering Research and Design*, 85(10), 1407-1422.

McArthur S. D. J., Davidson E. M., Hossack J. A. and McDonald J. R., 2004. Automating power system fault diagnosis through multi-agent system technology. *In the Proceeding of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, 5-8 Jan 2004, Big Island, Hawaii, 1-4.

Mendes M., Santos B. and Costa J., 2009. A Matlab/Simulink multi-agent toolkit for distributed networked fault tolerant control systems. *In the Proceeding of the 7th IFAC symposium on Fault Detection, Supervision and Safety of Technical Processes*. 30 June - 3 July 2009 , in Barcelona, Spain.

Miller J. P., 2006. Statistical signatures used with principal component analysis for fault detection and isolation in a continuous reactor. *Journal of Chemometrics*, 20(1-2), 34-42.

Milne R., 1987. Strategies for diagnosis. *IEEE Transactions on System, Man and Cyber*. 17 (3), 333-339.

Modarres M. and Cheon S. W., 1999. Function-centered modeling of engineering systems using the goal tree–success tree technique and functional primitives. *Reliability Engineering & System Safety*, 64(2), 181-200.

Narasimhan S., Vachhani P. and Rengaswamy R., 2008. New nonlinear residual feedback observer for fault diagnosis in nonlinear systems. *Automatica*, 44(9), 2222-2229.

Nelson W.R., 1982. REACTOR: an expert system for diagnosis and treatment of nuclear reactor accidents. *In the Proceeding of AAAI 82*, August 1982, 296-301.

Nelson V. P., 1990. Fault-tolerant computing: fundamental concepts**.** *Computer,* 23(7), 19-25.

Ng Y. S. and Srinivasan R., 2010. Multi-agent based collaborative fault detection and identification in chemical processes. *Engineering Applications of Artificial Intelligence*, 23(6), 934-949.

Niu G., Han T., Yang B. and Tan A. C. C., 2007. Multi-agent decision fusion for motor fault diagnosis. *Mechanical Systems and Signal Processing*, 21(3), 1285-1299.

Object Direct, 2010. *SysML Modelling Language Explained Guillaume FINANCE* [online]. Object Direct Analyst & Consultant. Available from: http://www.omgsysml.org/SysML_Modelling_Language_explained-finance.pdf [Accessed 5th of October 2010].

O'Hara J. M., Brown W. S., Halbert B., Skraaning G., Wachtel J., Persensky J., 1997. The use of simulation in the development of human factors guidelines for alarm systems. *In the Proceeding of the 1997 IEEE 6th Conference on Human Factors and Power Plants,*

Global Perspectives of Human Factors in Power Generation. 08 - 13 Jun 1997, Orlando, FL, USA, 1807-1813.

O'Hara J. M., Brown W. S., Lewis P. M. and Persensky J. J., 2002. *Human-System Interface Design Review Guidelines* [online]. Energy Sciences & Technology Department and Brookhaven National Laboratory, Upton, NY 11973-5000. Available: http://www.nrc.gov/reading-rm/doc-collections/nuregs/staff/sr0700/nureg700.pdf [Accessed February 22nd 2011].

Oulton B., 1992. Structured programming based on IEC SC 65 A, using alternative programming methodologies and languages with programmable controllers. *In the Proceeding of the IEEE Conference on Electrical Engineering Problems in the Rubber and Plastic Industries, IEEE no: 92CH3111-2, IEEE service centre*, 31-14 April, Akron, OH , USA, 18-20.

Padgham L., and Winikoff M., 2004. *Developing intelligent agent systems: A practical guide*. Chichester: John Wily & Sons LTD.

Papadopoulos Y., 2000. *Safety-directed system monitoring using safety cases*. DPhil Thesis. Technical Report No YCST-2000-08. University of York.

Papadopoulos Y., 2002. Model-based on-line monitoring using a state sensitive fault propagation model. *In the Proceeding of the 21st International Conference on Computer Safety, Reliability and Security (SAFECOMP '02)*. Lecture Notes in Computer Science (2434/2002), London: Springer-Verlag, 149-158.

Papadopoulos Y, 2003. Model-based system monitoring and diagnosis of failures using state-charts and fault trees. *Reliability Engineering and System Safety*, 8(3), 325-341.

Papadopoulos Y. and McDermid, J. A., 1999. Hierarchically performed hazard origin and propagation studies. *In the proceeding of the 18th International Conference in Computer Safety, Reliability and Security*. Toulouse, France, 139-152.

Papadopoulos Y. and McDermid J. A., 2001. Automated safety monitoring: a review and classification of methods. *International Journal of Condition Monitoring and Diagnostic Engineering Management,* 4(4), 14-32.

Papadopoulos Y., McDermid J., Sasse R. and Heiner G., 2001. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety,* 71 (3), 229-247.

Papazoglou I. A and Aneziris O. N., 2003. Master Logic Diagram: method for hazard and initiating event identification in process plants Original. *Journal of Hazardous Materials*, 97 (1-3), 11-30.

Patton R. J., 1997. Fault-tolerant control: The 1997 situation. *In the Proceeding of the 3rd IFAC symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS'97)*. August 1997, Hull, United Kingdom, 1033-1055.

Peng H., Shang W., Shi H. and Peng W., 2007. On-Line Monitoring and Diagnosis of Failures Using Control Charts and Fault Tree Analysis (FTA) Based on Digital Production Model. *In the Proceeding of the 2nd international conference on Knowledge science, engineering and management (KSEM'07)*. Lecture Notes in Computer Science (4798/2007). Berlin, Heidelberg: Springer, 544-549.

Petti T. F., Klein J. and Dhurjati P. S., 2004. Diagnostic model processor: Using deep knowledge for process fault diagnosis. *AIChE Journal*, 36(4), 565–575.

Pirus D., 1996. Alarm processing - ways to the future. *Presented at the International Atomic Energy Agency Specialists' Meeting on Experience and Improvements in Advanced*

Alarm Annunciation Systems in Nuclear PowerPlants. Vienna: International Atomic Energy Agency.

Pumfrey D, 1999. *The principled design of computer system safety analyses*, DPhil Thesis, University of York.

Rabiner L. and Juang B., 1986. An introduction to hidden Markov models. *IEEE-ASSP Magazine*, 3(1), 4-16.

Ramesh T. S., Shum S. K. and Davis J. F., 1988. A Structured Framework for Efficient Problem-Solving in Diagnostic Expert Systems. *Computers and Chemical Engineering*, 12(9-10), 891-902.

Ramesh T. S., Davis J. F. and Schwenzer G. M., 1989. Catcracker: an expert system for process and malfunction diagnosis in fluid catalytic cracking units. *Annual meeting of the American Institute of Chemical Engineering (AIChE)*, November 1989, San Francisco, CA.

Rao A. S., 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. *In the Proceeding of the 7th Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, 22-25 January, Eindhoven (ed. Van de Velde W and Perram J), vol. 1038 of LNAI, London: Springer, 42-55.

Reiter R., 1987. A Theory of diagnosis from first principles, *Artificial Intelligence, 32 (1), 57-96.*

Ren X., Thompson H. A. and Fleming P. J., 2006. An agent-based system for distributed fault diagnosis. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 10(2006), 319-335.

Rich S. H., Venkatasubramanian V., Nasrallah M. and Matteo C., 1989. Development of a diagnostic expert system for a whipped toppings process. *Journal of Loss Prevention in the Process Industries* 2 (3), 145-154.

Romagnoli J. and Palazoglu A., 2005. *Introduction to process control - chemical industries*. London: Taylor & Francis.

Roos N., Teije A., Bos A. and Witteveen C., 2002. An analysis of multi-agent diagnosis. *In the Proceeding of the 1st international joint conference on Autonomous agents and multiagent systems (AAMAS '02)*. 15-19 July 2002, Bologna, Italy, 986-987.

Roos N., Teije A. and Witteveen C., 2003a. A protocol for multi-agent diagnosis with spatially distributed knowledge. *In the Proceeding of the second international joint conference on Autonomous agents and multiagent systems (AAMAS '03)*. 14-18 July 2003, Melbourne, Australia, 655- 661.

Roos N., Teije A. and Witteveen C., 2003b. Multi-agent diagnosis with semantically distributed knowledge. *In the Proceeding of the 15th Belgium-Dutch Conference on Artificial Intelligence (BNAIC-2003).* 23-24 October 2003, Netherlands, Nijmegen, 259-266.

Roth E. and O'Hara J., 2002. *Integrating digital and conventional human-system interfaces: lessons learned from a control room modernization program* [online]. Division of Systems Analysis and Regulatory Effectiveness Office of Nuclear Regulatory Research U.S. Nuclear Regulatory Commission, Washington, DC 20555-0001 NRC Job Code W6546. Available: http://www.nrc.gov/reading-rm/doc-collections/nuregs/contract/cr6749/6749-021104.pdf [Accessed February 23rd 2011].

Rusinov L. A., Rudakova I. V., Remizova O. A. and Kurkina V. V., 2009. Fault diagnosis in chemical processes with application of hierarchical neural networks. *Chemometrics and Intelligent Laboratory Systems*, 97 (1-15), 98-103.

Russell J. and Norvig P., 1995. *Artificial Intelligence a Modern Approach*. USA, New Jersey: Prentice-Hall, Inc. Education.

Russell J. and Norvig P., 2010. *Artificial Intelligence a Modern Approach*. 3rd ed. UK: Pearson Education.

Sayda A. F., 2011. Multi-agent systems for industrial applications: design, development, and challenges. *In: Alkhateeb F., Al Maghayreh E., Abu Doush L., ed. Multi-Agent Systems - Modeling, Control, Programming, Simulations and Applications*. Rijeka, Croatia: InTech, 469-494.

Seo C. and Kim B. K. 1996. Robust and reliable H∞ control for linear systems with parameter uncertainty and actuator failure. *Automatica*, 32(3), 465-467.

Shoham Y., 1993. Agent-oriented Programming. *Artificial Intelligence*, 60 (1), P 51-92.

Shooman M. L., 2002. *Reliability of computer systems and networks: fault tolerance, analysis, and design*. USA, New York: John Wiley & Sons, Inc.

Somani K. A. and Trivedi K. S., 1994. Phased-mission system analysis using Boolean algebraic methods. *In the Proceeding of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems*. Nashville, Tennessee, United States, 98–107.

Srichander R. and Walker B. K., 1993. Stochastic stability analysis for continuous-time fault tolerant control systems. *International Journal of Control*, 57(2), 433–452.

Suh Y. S., Park J. Y., Kang H. T. and Kim H. S., 2008. An overview of instrumentation and control systems of a Korea standard nuclear power plant: A signal interface standpoint. *Nuclear Engineering and Design*, 238(12), 3508-3521.

Sullivan K. J., Dugan J. B. and Coppit D., 1999. The Galileo fault tree analysis tool. *In the Proceeding of the 29th Annual International Symposium on Fault-Tolerant Computing*, Madison, Wisconsin, 15-18 June 1999, 232-235.

Trimble E. J., 1990. *Report on the accident to Boeing 737-400 G-OBME near kegworth, leicestershire on 8 January 1989* [online]. Department of Transport Air Accidents Investigation Branch, Royal Aerospace Establishment. London, HMSO. Available from http://www.aaib.gov.uk/cms_resources.cfm?file=/4-1990%20G-OBME.pdf [Accessed 5th of October 2010]

U.S.NRC, 2008. *Fact sheet: backgrounder on the Three Mile Island Accident* [online]. United State nuclear regulatory commission (U.S.NRC). Available: http://www.nrc.gov/reading-rm/doc-collections/fact-sheets/3mile-isle.pdf [Accessed 20th February 2011].

Vardi M. Y., 1986. On epistemic logic and logical omniscience. *In the Proceeding of the 1986 conference on Theoretical aspects of reasoning about knowledge (TARK '86)*. Publishers; Morgan Kaufmann, Los Altos, CA, 293-306.

Veillette R. J., Medanic J. B. and Perkins W. R., 1992. Design of reliable control systems. *IEEE Transactions on Automatic Control*, 37(3), 290-304.

Venkatasubramanian V., Rengaswamy R., Yin K. and Kavuri S. N., 2003a. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods. *Computers and Chemical Engineering*, 27(3), 293-311.

Venkatasubramanian V., Rengaswamy R., Yin K. and Kavuri S. N., 2003b. A review of process fault detection and diagnosis: Part II: Qualitative Models and Search Strategies. *Computers and Chemical Engineering*, 27(3), 313-326.

Venkatasubramanian V., Rengaswamy R., Yin K. and Kavuri S. N., 2003c. A review of process fault detection and diagnosis: Part III: Process History based methods. *Computers and Chemical Engineering*, 27(3), 327-346.

Vesely W. E., Goldberg F. F., Robert N. H. and Haasl D. F., 1981. *Fault Tree Handbook*. U.S. Nuclear Regulatory Commission. USA: Washington.

Vesely W. E., Stamatelatos M., Dugan J. B., Fragola J., Minarick J. and Railsback J., 2002. *Fault tree handbook with aerospace applications*. NASA Office of Safety and Mission Assurance. USA: Washington.

Vieira R., Moreira A., Wooldridge M. and Bordini R. H., 2007. On the formal semantic of speech-act based communication in an agent-oriented programming language. *Artificial Intelligence Research*, 29(2007), 221-267.

Wallace C. J., Jajn G. J. and Mcarthur S. D. J., 2011. Multi-agent system for nuclear condition monitoring. *In the Proceeding of the 2nd International Workshop on Agent Technologies for Energy System (ATES'11), a workshop of the 10th International Conference of Agent and Multi-agent System (AAMAS'11)*, 2nd of May 2011, in Taipei, Taiwan.

Weilkiens T., 2006. Systems engineering with SysML/UML: modeling, analysis, design. London: Morgan Kaufmann.Tim Wei. .lkiensWise B.M., Gallagher N.B., 1996. The process chemometrics approach to process monitoring and fault detection. *Journal of Process Control*, 6(6), 329-348.

Welch G. and Bishop G., 2001. An Introduction to the Kalman Filter. *Design*, 7(1), 1-16.

Willsky A. S., 1976. A survey of design methods for failure detection in dynamic systems. *Automatica*, 12, 601-611.

Wold S., Ruhe A., Wold H. and Dunn W., 1984. The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses. *SIAM Journal of Science Statistical Computer*, 5, 735-743.

Yu C. and Su B., 2006. Eliminating false alarms caused by fault propagation in signal validation by sub-grouping. *Progress in Nuclear Energy*, 48(4), 371-379.

Zhang Y. and Jiang J., 2008. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual Reviews in Control*, 32(2), 229-252.

Zhao Q. And Jiang J., 1998. Reliable state feedback control system design against actuator failures. *Automatica*, 34(10), 1267-1272.

Zolghadri A., Henry D. and Monsion M., 1999. Design of nonlinear observers for fault diagnosis: a case study. *Control Engineering Practice*, 4(11), 1535-1544.

Zolghadri A., 2002. Early warning and prediction of flight parameter abnormalities for improved system safety assessment. *Reliability Engineering & System Safety*, 76(1), 19-27.

Appendix A: List of Abbreviations

AADL: Architectural Analysis and Design Language

ABS: Aircraft Brake System.

ABS_MAG: Aircraft Brake System Monitoring Agent.

AFS: Aircraft Fuel System.

AFS_MAG: Aircraft Fuel System Monitoring Agent.

AFTC: Active Fault-Tolerant Controlling.

ANDN: AND Node.

ANN: Artificial Neural Network.

BDI agent: Belief, Desire and Intention agent.

BEN: Basic Event Node.

BSCU: Braking System Control Unit.

CD: Central Deposit sub-system of the aircraft fuel system.

CD_MAG: Central Deposit sub-system Monitoring Agent.

DFD: Data Flow Diagrams.

E-BNF: Extended-Backus Naur Form.

EF: Engine Feed sub-system of the aircraft fuel system.

EF_MAG: Engine Feed sub-system Monitoring Agent.

EFS: Error-free State.

EICAS: Engine Indication and Crew Alerting System.

ES: Error State.

FCC: Flight Control Computer.

FDD: Fault Detection and Diagnosis.

FEN: Failure Event Node.

FFA: Functional Failure Analysis.

FFBD: Functional Flow Block Diagram.

FMEA: Failure Mode and Effect Analysis.

FS: Failure state.

FSCU: Fuel System Control Unit.

GTST: Goal Tree Success Tree.

HAZOP: HAZard and OPerability study.

HiP-HOPS: Hierarchically Performed Hazard Origin and Propagation Studies.

LW: Left Wing sub-system of the aircraft fuel system.

LW_MAG: Left Wing sub-system Monitoring Agent.

LWB: Left-side Wheel Brake sub-system of the aircraft brake system.

LWB_MAG: Left-side Wheel Brake sub-system Monitoring Agent.

MFM: Multi-level Flow Model.

ORN: OR Node.

PDS: Permanent Degraded State.

PFTC: Passive Fault-Tolerant Controlling.

RW: Right Wing sub-system of the aircraft fuel system.

RW_MAG: Right Wing sub-system Monitoring Agent.

RWB: Right-side Wheel Brake sub-system of the aircraft brake system.

RWB_MAG: Right-side Wheel Brake sub-system Monitoring Agent.

S_MAG: System Monitoring Agent.

SFDT: Sensory Failure Diagnosis Tree.

Ss_MAG: Sub-system Monitoring Agent.

STA: Statistical Trend Analysis.

TDFS: Temporary Degraded/Failure State.

TEN: Top Event Node.

UML: Unified Model Language.