This item was submitted to Loughborough's Research Repository by the author.
Items in Figshare are protected by copyright, with all rights reserved, unless otherwise indicated.

Systems reliability modelling for phased missions with maintenance-free operating periods

PLEASE CITE THE PUBLISHED VERSION

PUBLISHER

© Samuel Chew

LICENCE

CC BY-NC-ND 4.0

REPOSITORY RECORD

Chew, Samuel P.. 2010. “Systems Reliability Modelling for Phased Missions with Maintenance-free Operating Periods”. figshare. https://hdl.handle.net/2134/6331.

This item was submitted to Loughborough's Institutional Repository (https://dspace.lboro.ac.uk/) by the author and is made available under the following Creative Commons Licence conditions.

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/

Loughborough University

Thesis Access Form

Copy No……..…….**Location**……..……...…

Author……...……..…….

Title……..

Status of access OPEN / RESTRICTED / CONFIDENTIAL

Moratorium Period:……years, ending……../……200…….

Conditions of access approved by (CAPITALS):……

Supervisor (Signature)……...……...

Department of……...……

Author's Declaration: *I agree the following conditions:*

Open access work shall be made available (in the University and externally) and reproduced as necessary at the discretion of the University Librarian or Head of Department. It may also be digitised by the British Library and made freely available on the Internet to registered users of the EThOS service subject to the EThOS supply agreements.

*The statement itself shall apply to **ALL** copies including electronic copies:*

This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

Restricted/confidential work: All access and any photocopying shall be strictly subject to written permission from the University Head of Department and any external sponsor, if any.

Author's signature……Date……...……...……...

users declaration: for signature during any Moratorium period (Not Open work): ***I undertake to uphold the above conditions:***			
Date	Name (CAPITALS)	Signature	Address

Systems Reliability Modelling for Phased Missions with Maintenance Free Operating Periods

By

Samuel Chew

A Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University

May 2010

© by Samuel Chew 2010

Loughborough University

CERTIFICATE OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this thesis, that the original work is my own except as specified in acknowledgments or in footnotes, and that neither the thesis nor the original work contained therein has been submitted to this or any other institution for a degree.

……. (Signed)

……. (Date)

In memory of my dad

Abstract

In 1996, a concept was proposed by the UK Ministry of Defence with the intention of making the field of reliability more useful to the end user, particularly within the field of military aerospace. This idea was the Maintenance Free Operating Period (MFOP), a duration of time in which the overall system can complete all of its required missions without the need to undergo emergency repairs or maintenance, with a defined probability of success. The system can encounter component or sub-system failures, but these must be carried with no effect to the overall mission, until such time as repair takes place. It is thought that advanced technologies such as redundant systems, prognostics and diagnostics will play a major role in the successful use of MFOP in practical applications.

Many types of system operate missions that are made up of several sequential phases. For a mission to be successful, the system must satisfactorily complete each of the objectives in each of the phases. If the system fails or cannot complete its goals in any one phase, the mission has failed. Each phase will require the system to use different items, and so the failure logic changes from phase to phase. Mission unreliability is defined as the probability that the system fails to function successfully during at least one phase of the mission. An important problem is the efficient calculation of the value of mission unreliability.

This thesis investigates the creation of a modelling method to consider as many features of systems undergoing both MFOPs and phased missions as possible. This uses Petri nets, a type of digraph allowing storage and transit of tokens which represent system states. A simple model is presented, following which, a more complex model is developed and explained, encompassing those ideas which are believed to be important in delivering a long MFOP with a high degree of confidence. A demonstration of the process by which the modelling method could be used to improve the reliability performance of a large system is then shown. The complex model is employed in the form of a Monte-Carlo simulation program, which is applied to a life-size system such as may be encountered in the real world. Improvements are suggested and results from their implementation analysed.

Acknowledgements

Enormous thanks must go to my supervisors, Professor John Andrews and Dr Sarah Dunnett, for their huge efforts in guiding and supporting me as I worked through my Ph.D. Without them and the incredible patience they have shown, the creation of this thesis would not have been possible.

Thanks must also be given to my sponsors at BAE Systems – Angus Murray and Clive Downes, plus Greg Slaughter – for their generous funding, and for taking so much time and effort to ensure that I had everything I needed to complete the work.

There are also many people in the Aeronautical and Automotive Engineering at Loughborough University who made the experience of carrying out this research one which I will always treasure.

To my mum, Cathy, my brother, Jim, my sisters, Sine and Lindsay, and all my friends and family who have taken an interest in the progress of this work, you should know that without your support and encouragement, it is unlikely that this would have been completed.

Thank you God for giving me the skills and opportunities I have had to make it this far, and for being with me throughout all of this.

Finally, to my wife, Christabel: I can never ever repay the huge amount of belief, support, management, guidance, understanding, patience and love that you have shown to me in the time it has taken to finish this work. If it were not for you, I am sure I would have given up on this work many years ago. The existence of this thesis is as much testament to your hard work as it is to mine.

Contents

Chapter 1 Introduction **1**

1.1 Introduction to Reliability and Risk Assessment 1
1.2 System Failure Quantification 3
1.3 Fault Tree Analysis 5
1.4 Markov Methods 7
1.5 Monte-Carlo Simulation 7
1.6 Phased Mission Systems 8
1.7 Maintenance-Free Operating Periods 9
1.8 Research Objectives 10

Chapter 2 Reliability Analysis Tools **12**

2.1 Introduction 12
2.2 Fault Tree Analysis 12
2.2.1 Fault Tree Construction 12
2.2.2 Qualitative Analysis 14
2.2.3 Quantitative Analysis 16
2.2.3.1 Top Event Probability 16
2.2.3.1.1 Upper and Lower Bounds for System Unavailability 17
2.2.3.1.2 Minimal Cut Set Upper Bound 17
2.2.3.2 Importance Measures 17
2.3 Binary Decision Diagrams 19
2.3.1 BDD Properties 20
2.3.2 Formation of a BDD using If-Then-Else Structure 21
2.3.3 BDD Minimisation 22
2.4 Markov Analysis 23
2.5 Summary 25

Chapter 3 Maintenance Free Operating Periods **27**

3.1 Introduction 27
3.2 Explanation of MFOP 28
3.3 MFOP Basics 29
3.4 Impact of MFOP 30

3.4.1 Technologies Within MFOP 30
3.4.2 Advantages 31
3.4.3 Disadvantages 32
3.4.4 Arguments Surrounding MFOP 33
3.5 Using and Achieving MFOP 34
3.5.1 Enablers of MFOP 35
3.5.1.1 Inherent Reliability (IR) 35
3.5.1.2 Prognostics 37
3.5.1.3 Diagnostics 38
3.5.1.4 Redundancy/Reconfigurability 38
3.5.2 Considering MFOP in Design 40
3.5.3 Maintenance Recovery Period 41
3.6 Analysing MFOP 43
3.6.1 Mathematics of MFOP 43
3.6.2 Ultra Reliable Aircraft Model (URAM) 48
3.7 Summary 50

Chapter 4 Phased Missions and Petri Nets 52

4.1 Introduction 52
4.2 Phased Missions 52
4.2.1 Non-Repairable Systems 52
4.2.1.1 Transformation of a Multi-Phased Mission to an Equivalent Single-Phased Mission 53
4.2.1.1.1 Cut Set Cancellation 55
4.2.1.2 Obtaining Bounds for Mission Unreliability 56
4.2.1.3 Expected Number of Failures 59
4.2.1.4 Laws of Boolean Phase Algebra 59
4.2.1.5 Binary Decision Diagrams 66
4.2.1.6 Non-Coherent Fault Trees 71
4.2.1.7 Importance Measures 74
4.2.1.8 Markov Methods 77
4.2.2 Repairable Systems 79
4.2.2.1 Combinatorial Approaches 79
4.2.2.2 Markov Methods 80

4.2.2.2.1 Homogeneous Markov Model 80
4.2.2.2.2 Non-Markovian Models 89
4.2.3 Summary 92
4.3 Petri Nets 93
4.3.1 Introduction 93
4.3.2 Formal Definition 94
4.3.3 Basics of Petri nets 94
4.3.3.1 Original Petri net Concept 94
4.3.3.2 Timed Petri nets 95
4.3.3.3 Arc Multiplicities 97
4.3.3.4 Inhibitor Arcs 98
4.3.3.5 Reachability Graphs 99
4.3.3.6 Coloured and Aging Petri nets 101
4.3.3.7 Other Variations 102
4.3.4 Application of Petri nets to Reliability 102
4.3.4.1 Markov Analysis 103
4.3.4.2 Fault Tree Modelling 105
4.3.4.3 Multi-Phased Missions 108
4.3.5 Summary 111

Chapter 5 Development of Initial Model 113

5.1 Introduction 113
5.2 Development of Model 114
5.2.1 System Net 114
5.2.1.1 Phase Petri Nets 116
5.2.1.2 Component Petri Net 118
5.2.1.2.1 Dependency Modelling 119
5.2.2 Phase Net (Master Petri Net) 120
5.2.3 Simulation Model 123
5.3 Model Validation 124
5.3.1 Non-repairable Small System 124
5.3.1.1 Theoretical Analysis 125
5.3.1.2 Petri Net Model Results 130
5.3.1.3 Comparison of Results 132

5.3.2 Repairable Small System 132
5.3.2.1 Markov Analysis 133
5.3.2.2 Comparison of Results 135
5.3.3 Larger Repairable System 136
5.3.3.1 Results 138
5.4 Summary 140

Chapter 6 Development of Complex Model 141

6.1 Introduction 141
6.2 Phased Missions 141
6.2.1 Phase Lengths 141
6.2.2 Probability Transitions 142
6.2.3 Fleet of Platforms 143
6.2.4 Platform Operation 144
6.2.5 MFOP Operation 146
6.2.6 Mission Operation and Abandonment 147
6.2.7 Total Loss of Platform 148
6.2.8 MFOP Abandonment 149
6.2.9 Mission Abandonment to Phase 150
6.2.10 MRP Modelling 151
6.2.11 General Failure 152
6.2.12 Phase Selection 154
6.2.12.1 Event-driven Phase Selection 154
6.2.12.2 Probability-driven Phase Selection 155
6.2.13 Phase Insertion 156
6.3 Reliability & Maintainability and MFOP Enablers 157
6.3.1 Inherent Reliability 157
6.3.2 Discrete Events 158
6.3.3 Component Replacement 159
6.3.4 Line Replaceable Items 160
6.3.5 Scheduled Life Replacements 164
6.3.6 Diagnostics 166
6.3.6.1 Sensors 166
6.3.6.2 Inspection 168

6.3.7 Redundancies 177
6.3.8 Prognostics 185
6.3.8.1 Modelling Component Wear 186
6.3.8.2 Diagnosing Wear Level 188
6.3.8.3 Predicting Component Failure Times 190
6.3.8.4 Assessing MFOP Criticality 194
6.3.8.5 Modelling Actions Taken Based on Predictions 195
6.4 Summary 199

Chapter 7 Application of Model to Large System 201

7.1 Introduction 201
7.2 Application to Previous Systems 201
7.2.1 Non-repairable Small System 202
7.2.2 Repairable Small System 202
7.2.3 Repairable Larger System 203
7.3 Application to a Large System 205
7.3.1 The System 206
7.3.1.1 Potential for Total Loss 206
7.3.1.2 Potential for Mission Abandonment 207
7.3.1.3 Missions and Phases 208
7.3.1.4 Platforms and MFOPs 209
7.3.1.5 Component data 210
7.3.1.6 Enablers 210
7.3.2 Performance of Simulations and Results 210
7.3.2.1 First Generation 210
7.3.2.2 Second Generation 217
7.3.2.3 Third Generation 225
7.4 Summary 229

Chapter 8 Conclusions and Further Work 230

8.1 Summary 230
8.2 Conclusions 235
8.3 Further Work 237
8.3.1 Verification of Repairable Larger System Results 237

8.3.2 Cost modelling 237
8.3.3 Reconfigurability 238
8.3.4 Phase and Component Dependency 238
8.3.5 Degradation 238
8.3.6 Prognostics 239
8.3.7 Platforms and MFOPs 239
8.3.8 MFOPs and Missions 239
8.3.9 Maintenance & Logistics Support 240
8.3.10 Importance Measures 240
8.3.11 Optimisation 240
8.3.12 System of Systems 241

References 242

Appendix A Description of Simulation Program 250

A.1 Introduction 250
A.2 Basic Information 251
A.3 Inputting Data to the Program 254
A.3.1 Phase Fault Tree, Component and Sensor Data 254
A.3.1.1 Phase Fault Trees 256
A.3.1.2 Component Failure Data 259
A.3.1.3 Sensor Data 261
A.3.1.4 Discrete Events 262
A.3.1.5 Component Dependencies 263
A.3.2 MFOP Enabler Data 264
A.3.2.1 LRI Data 265
A.3.2.2 Scheduled Life Replacements 268
A.3.2.3 Inspections 268
A.3.2.4 Redundancies 270
A.3.2.5 Prognostics 271
A.3.3 Platform, MFOP and Mission Data 273
A.3.3.1 Fleet and MFOP information 274
A.3.3.2 Missions Information 275
A.3.3.3 Creating the Master Petri Net 279

A.3.3.4 Discrete Events 280
A.4 Performing Simulations 281
A.4.1 Simulation options 281
A.4.2 Simulation Process 283
A.4.2.1 Initial Processes 283
A.4.2.2 Sampling Transition Switching Times 287
A.4.2.3 Simulation Performance 289
A.4.2.4 Assessing Component Criticality 292
A.5 Outputs 297
A.5.1 Petri Net information 298
A.5.2 Simulation Results 300
A.5.2.1 Platform Failure Information 301
A.5.2.2 MFOP Failure Information 302
A.5.2.3 Mission Failure Information 302
A.5.2.4 Phase Failure Information 304
A.5.2.5 System Failure Information 305
A.5.2.6 Component Failure Information 307
A.5.2.7 Discrete Event Information 308
A.5.2.8 LRI information 308
A.5.2.9 Prognostics Information 309
A.5.2.10 Redundancies Information 309
A.5.2.11 Inspections Information 310
A.6 Summary 310

Appendix B Data Input Files 312

B.1 First Generation 313
B.1.1 Fault Tree, Component Data and Discrete Events file 313
B.1.2 MFOP Enablers 332
B.1.3 Platform, MFOP, Mission and Phase Data 332
B.2 Second Generation 334
B.2.1 Fault Tree, Component Data and Discrete Events file 334
B.2.2 MFOP Enablers 336
B.3 Third Generation 337
B.3.1 Fault Tree, Component Data and Discrete Events file 337
B.3.2 MFOP Enablers 339

Nomenclature

$A(t)$	Availability function
$[\mathbf{A}]$	State transition matrix
A_j	Occurrence of component A in phase j
$A_{(j)}$	Existence of component A in phase j
A_{ij}	Failure of component A from start of phase i to end of phase j
a_j	Failure of component A in phase j
$\overline{A_j}$	Success of component A in all phases up to and including phase j
C	Consequence of an event
C_i	Existence of cut set i
$C_{i_{(j)}}$	Existence of cut set i in phase j
D	Vector of switching delays of transitions
E	Set of edges (arcs) in Petri net
$E[A_j]$	Expectation of component A in phase j
E_j	Boolean expression for the failure combinations of phase j
$F(t)$	Unreliability function
$f_i(t)$	Failure probability density function (PDF) for item i
$F_i(t)$	Failure cumulative distribution function (CDF) for item i
$G_c(\boldsymbol{q}(t))$	Criticality function for event c (Birnbaum's measure of importance)
$G_{i,j}$	Criticality function for component i in phase j
$G^T_{i,j,k}$	Phase transition criticality function for component i in phase k prior to phase j
$\mathbf{G}_{\mathrm{PN}}$	Set of Petri net data
$h(t)$	Conditional failure rate (hazard rate)
I_c^{CR}	Criticality measure of component importance
I_c^{FV}	Fussell-Vesely measure of component importance
$I_{C_k}^{FV}$	Fussell-Vesely measure of minimal cut set k importance
I_c^{ST}	Structural measure of component importance
$I_{i,j}$	Importance measure for component i in phase j
$I_{i,j}^P$	Phase component importance measure for component i in phase j

$I_{i,j}^{T}$	Phase transition importance measure for component i in phase j
$\mathbf{M}(t)$	Vector of Petri net place marking at time t
$M_i(t)$	Number of tokens in place i at time t
m	Number of phases in mission
m_f	Number of failure modes
N_{C_i}	Number of basic events in cut set C_i
$N_{C_{i_j}}$	Number of basic events in cut set C_i in phase j
n_l	Number of wear levels
N_{mcs}	Number of minimal cut sets
N_{mcs_j}	Number of minimal cut sets in phase j
N_S	Total number of system states
P	Probability
$[\mathbf{P}]$	State probability vector
p_{c_j}	Component phase j availability (success probability)
$p_{C_{i_j}}$	Probability of success of cut set C_i in phase j
P_{l_a}, P_{l_b}	Lower and upper percentage bounds of wear level l
$P_{S_i}(t)$	Probability of being in state S_i at time t
PFC_j	Phase failure combinations for phase j
$Q(t)$	Unavailability function
$Q_j(t)$	Phase j failure probability (system phase j unavailability)
$\boldsymbol{q}(t)$	Vector of component unavailabilities
$q_c(t)$	Component unavailability (failure probability) function
$q_{c_j}(t)$	Component unavailability (failure probability) function in phase j
$q_{C_{i_j}}(t)$	Unavailability function of cut set C_i in phase j
$q_{c_{ij}}$	Unavailability of component c from phase i through to phase j inclusive
$Q_{IN\text{-}EX}$	Minimal unreliability bound using the inclusion-exclusion expansion
$Q_{IN\text{-}EX(CC)}$	$Q_{IN\text{-}EX}$ with cut set cancellation technique
Q_{MCB}	Minimal unreliability bound using the minimal cut set bound to estimate phase unreliability
$Q_{MCB(CC)}$	Q_{MCB} with cut set cancellation technique

Q_{MCSU}	Minimal cut set upper bound
Q_{MISS}	Mission unreliability
$Q_{SYS}(t)$	System unavailability function
$Q_S(t)$	Probability of being in state S at time t
$\dot{Q}_S(t)$	Rate of transition into state S
R	Risk
$R(t)$	Reliability function
$\mathbf{R}(t)$	Vector of system state reward rates
$r_{A_{i,j}}$	Probability of success of component A in phases i to j inclusive
$R_i(t)$	Probability of item i surviving to time t
R_j	Phase j reliability (success probability)
$R_{MISS}(t)$	Mission reliability (success probability)
$R_{SYS}(t)$	System reliability function
$S_i(t+u \mid t)$	Survivability of item i after interval length u given survival at time t
S_{nar}	Non-age-dependent survivability
t	Time
t_{F_c}, t_{R_c}	Component failure and repair time
t_{F_a}, t_{F_b}	Upper and lower bounds for predicted component failure time
t_i	Life of item i
t_{I_z}	Time of current inspection z
t_j	End of phase j and start of phase j+1
t_l	Time to enter wear level l
u	Length of MFOP
u_{max}	Design life of system (maximum MFOP length)
$\mathbf{V}_p$	Set of places in Petri net
$\lvert\mathbf{V}_p\rvert$	Number of places in Petri net
$\mathbf{V}_t$	Set of transitions in Petri net
X_j	Event of system being failed in phase j
x_c	Binary indicator variable for component states
$Y(t)$	Instantaneous accumulated reward
β_i	Weibull distribution shape parameter for component i
η_i	Weibull distribution characteristic life parameter for component i

λ_c	Conditional failure rate of component
μ_i	Mean time to failure (MTTF) for failure mode *i*
μ_{SYS}	Mean time to failure (MTTF) for the system
υ_c	Conditional repair rate of component
ρ_{c_j}	Conditional reliability of component *c* in phase *j*

Chapter 1 Introduction

1.1 Introduction to Reliability and Risk Assessment

On rare occasions an incident will occur with catastrophic consequences due to the failure of an industrial system. The industries within which these systems operate include aeronautical, nuclear, oil and gas, and transport. Incidents such as the explosion on the Piper Alpha oil platform in 1988 and the Air France disaster over the Atlantic in 2009 both resulted in multiple fatalities. Examples such as this demonstrate the importance of efforts to assess accurately and reduce the risk posed from complex systems. Applying system safety assessments at an early stage in design can reduce or prevent the possibility of unwanted incidents occurring once the system is in active use.

Developing techniques to assess the risk and reliability of systems has been a research interest for many years, with a great deal of progress made since the Second World War. These techniques provide the ability to evaluate the causes, and probability or frequency, of a hazardous event occurrence, taking into account, where necessary, the failure of safety systems to respond. The risk or 'expected loss', R, of any hazardous event is defined as the product of its consequence, C, and the probability or frequency of its occurrence, P:

$$\mathbf{R = C \times P} \tag{1.1}$$

In safety studies, a consequence can be, for example, number of fatalities, cost of compensation, or damage to buildings. The risk can be reduced either by reducing the associated consequences of the hazard, or by reducing the probability or frequency of its occurrence.

A quantitative risk assessment of a potential hazard involves four basic stages:

1. Identification of potential safety hazards.
2. Estimation of the consequences of each hazard.
3. Estimation of the probability or frequency of each hazard.
4. Comparison of the results against the acceptability criteria.

The consequences of a hazard are usually measured by the expected number of fatalities and indicate the severity of the incident. Consequence modelling is very much industry dependent, as systems and their modes of failure can vary significantly from one industry to another. Reliability assessment techniques, however, which are concerned with calculating the probability or frequency with which system failure can occur, are generic. Methods such as Failure Mode and Effect Analysis (FMEA), Event Tree Analysis, Markov Analysis and Fault Tree Analysis are used extensively in many industries.

Once a hazard has been assessed to find the consequences and probability of its occurrence, Equation 1.1 is used to determine the associated risk. In order to assess whether a level of risk is acceptable, the HSE (Health and Safety Executive) consider the use of a three-band approach known as the ALARP ("As Low As Reasonably Practicable") principle [1]. This is shown in Figure 1.1.

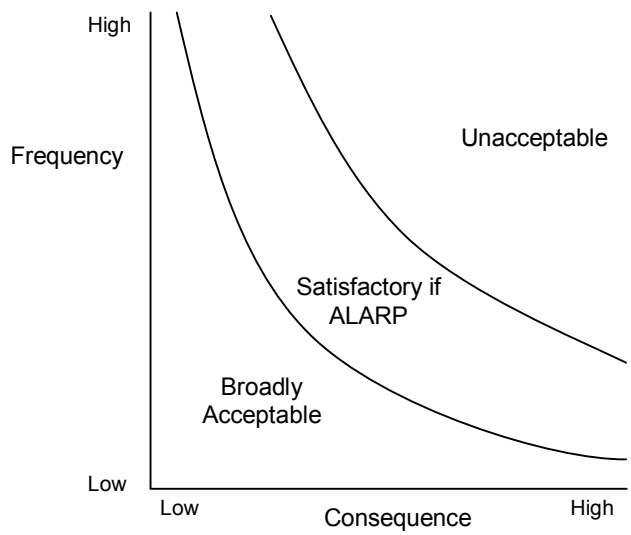


Figure 1.1 – The ALARP Principle [1]

Risks that fall into the "Broadly Acceptable" region are considered low enough to be permissible. Generally, they have a low probability of occurrence and do not have a severe hazard associated with them. Risks that fall into the "Unacceptable" region are not tolerated and either the probability or consequence of the event must be reduced. Between these bands is the "Satisfactory if ALARP" region, where risks are considered to be acceptable if they are as low as reasonably practicable. The test for reasonable practicability takes into account factors such as cost, effort, industry or legal standards, comparative risk prevention by competitors, and so on.

1.2 System Failure Quantification

It is possible to predict the reliability performance of a system from the known reliability performances of the components which make it up, by using appropriate techniques. This reliability performance of system or components can be expressed through the quantification of failure probabilities. The common parameters that are used throughout this thesis are defined below.

If a system or component can be repaired, and so its failure can be tolerated, a useful measure of performance is *availability*, $A(t)$. This is defined as:

> the fraction of the total time that a system or component is able to perform its required function.

This parameter can also be defined at a specified time point t as:

> the probability that a system or component is working at time t.

The complement of availability is unavailability, where:

$$\text{Unavailability} = 1 - \text{Availability}$$

Unavailability is defined as the probability that a component or system does not work at a given time t, and is denoted by $q_c(t)$ for a component and $Q_{sys}(t)$ for a system.

Reliability of a system or component, $R(t)$, can be defined as:

> the probability that a system or component will operate without failure for a stated period of time under specified conditions.

Reliability is typically more appropriate for systems where failure cannot be tolerated, and so the successful functioning of the system or component over a given time interval is an important performance measure. The probability that a system or component fails to work continuously over a given duration and under specified conditions is known as its unreliability, $F(t)$, where:

$$\text{Unreliability} = 1 - \text{Reliability}$$

If a component or system is non-repairable and it is working at time t, then it must have worked continuously since t=0. Therefore for the non-repairable case, the unavailability is equivalent to the unreliability.

The transition of a component or system to a failed state can be characterised by the conditional failure rate, $h(t)$, also known as the hazard rate. This parameter is a measure of the rate at which failures occur amongst those items which have not yet failed, i.e. those that are still functioning at time t. It is defined as follows:

> the conditional failure rate, $h(t)$, is the probability that a system or component fails in the interval $(t, t+dt)$, given that it has not failed in $[0,t)$.

A component family will typically have reliability characteristics which can be modelled by a "reliability bath-tub curve", as shown in Figure 1.2.

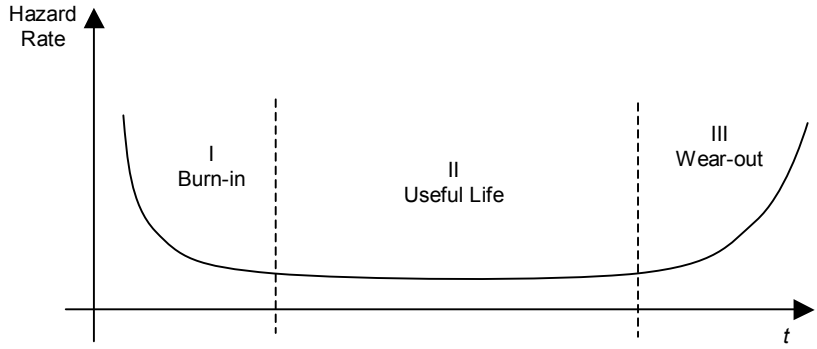


Figure 1.2 – Reliability Bath-Tub Curve

Phase I of the curve in Figure 1.2, called "Burn-in", sees the hazard rate reducing over time as weaker components fail and are removed from the overall population. During the second phase there is an almost constant hazard rate, leading to this period of life being known as the "useful life" of the components. The hazard rate Phase III increases as the components start to wear out and fail. Reliability assessment is typically performed on components which are thought to be in their useful life phase. The system reliability, $R_{sys}(t)$, can be expressed in terms of a constant hazard or failure rate λ by equation 1.2.

$$R_{sys}(t) = e^{-\lambda t} \tag{1.2}$$

Further component and system quantification methods can be found in [2] and [3]. Tools exist which allow the evaluation of the reliability parameters of a system in terms of those of its constituent components. Two of the typical methods, Fault Tree analysis and Markov methods, are discussed below.

1.3 Fault Tree Analysis

Fault Tree Analysis is a concept first introduced by H. A. Watson in the early 1960s [4], and is a deductive procedure for identifying the causes of a particular system failure mode using a "what could cause this" technique. The fault tree diagram provides a visual representation of the combinations of component failure events which result in the system failure mode occurring. The system failure mode under

consideration is referred to as the "top event" of the fault tree and branches of the tree are constructed below, each one of which represents a contributory cause. The events are continually redefined in terms of their causes, until each branch ends with a *basic event*, which is typically a component failure or human error. A fault tree can then be analysed to provide information on the causes of top event in terms of the basic events, with the probabilities of the latter allowing the calculation of the top event probability.

As the Fault Tree Analysis process starts with the top event and works downwards, building the fault tree beneath, this is known as an example of a "top-down" technique. This is in contrast to other methods, such as Failure Modes and Effects Analysis (FMEA), which are known as "bottom-up" approaches, since their starting point is a set of component failure conditions and the process allows the identification of the possible consequences using a "what happens if" approach.

Techniques which allow the quantitative analysis of a fault tree, known as Kinetic Tree Theory (KTT), were developed in the early 1970s by Vesely [5]. KTT allows the calculation of various system reliability parameters, such as probability and frequency of top event occurrence, using those of the basic events. This allows an assessment of whether the risk of system failure is acceptable and so if the required safety standards are being met.

The main disadvantage of KTT is that as fault trees grow larger, the analysis becomes more and more computationally intensive and so may require the use of approximations. This obviously leads to inaccuracies in the calculations. As the techniques are already so well developed, further refinement is unlikely to result in vast improvements.

1.4 Markov Methods

KTT is based on the assumption of statistical independence between the basic events in a fault tree. In many cases this is invalid, such as in systems which feature standby redundancy, common cause failures, secondary failures, or multiple-component states.

Markov analysis provides a means of analysing the reliability and availability of systems whose components express these dependencies. The method is a state-space approach. The likelihood of any event in the chain is determined only by the immediately preceding state and is independent of any other past events. As with fault tree analysis, the main disadvantage of the Markov approach is that as the number of components, and thus the number of system states, grows, the model can grow very rapidly in size.

1.5 Monte-Carlo Simulation

Sometimes, a system cannot be easily solved by graphical or simple mathematical methods. Another solution is to simulate the reliability performance of the system on computer. The method works by generating a computer model of the system failure logic in some way, and using a random number generator to simulate times to failure for each component. The failure times generated can be made to follow the distribution of times for each component, such that if a sufficient number of times were randomly generated and plotted with frequency of occurrence against time, the distribution would be recreated. These times are then fed into the failure logic for the system, and the combinations of component failures at generated times will lead to system failure after a certain time. As more simulations are performed, a description of the system reliability failure distribution is built up, which can then be used for analysis.

1.6 Phased Mission Systems

A common scenario in many real-world situations is one where a system undergoes missions, with distinct and differing objectives at consecutive time intervals. Such a concept is known as a *Phased Mission*, as first described by Esary and Ziehms [6], while each time interval with a unique objective is known as a *phase*. During the mission, in order to complete the different objectives for each phase, different sub-systems and components will need to be in use at different times, and as such the ways in which the overall system can fail changes with time. The phases in a mission may be identified in ways such as phase number, time interval, system configuration, task(s) to be undertaken, performance measure(s) of interest, or maintenance policy.

A multi-phased mission can be considered as a sequence of individual events. The success of each of these events means success of the overall mission. Thus, for the mission to be successful, each of the phases must not fail. Examples of systems which undergo phased missions include aircraft flights, rocket launches and many military operations for aircraft or ships. An aircraft mission can be thought of as containing the following phases: Pre-flight check, taxi to runway, take-off, ascent, transit to destination, descent and approach, landing, and taxi to the terminal. Components in the system may fail at any time, but their failure may only be critical for the failure of the system during a specific phase. As a result of this, the transition from one phase to the next may be the critical event which causes system, phase and mission failure, if component failures occurred earlier in the mission.

It is not possible to find the reliability of a mission by simply finding all of the phase reliability values and multiplying them together. This is due to the condition that at phase change times, the system must be in a state which is operational for both the current and the next phases. This creates statistical dependence between the phases. In order to understand how phases and missions can fail, it is necessary to express how the combinations of component failures can occur in each phase and thus cause system failure. Quantification of this allows mission failure probability and frequency to be determined. Mission unreliability is defined as the probability

that the system fails to function in at least one of the phases within it. An important problem is the calculation of the exact value of this parameter to a high level of efficiency.

The techniques which have been applied to solve for phased mission reliability parameters are predominantly those of fault tree, Markov or simulation analysis. Fault tree analysis is commonly used in industrial applications to find the probability of failure of their systems. It can be used to express the failure logic of the system in each phase. The combination of a different model for each of the potentially many mission phases makes the process of solving for reliability parameters considerably more complex than a single-phase system. If the assumption of independence between component failures is not applicable in a given situation, another method such as Markov analysis may be employed. If the modelling of a system by either of these methods is impractical or unfeasible, then it may be necessary to simulate the performance of the system. An instance where this may be required could be one where the system is too complex to use deterministic analysis, or if the component failure or repair rates are not constant. Simulation techniques typically offer a very high level of breadth in terms of representation and analysis, but suffer from often very expensive computational requirements.

1.7 Maintenance-Free Operating Periods

The concept of Maintenance-Free Operating Periods (MFOP) is an attempt to provide a more useful way of describing the reliability performance of a system. Created with the operators of military aircraft in mind, for whom the successful planning and execution of missions and maintenance is critical, it expresses the availability of an aircraft in a slightly different way. An MFOP can be defined as:

> The period of operation during which an item will be able to carry out all its assigned missions, without the operator being restricted in any way due to system faults or limitations, with the minimum of maintenance [7].

In other words, an MFOP is a time interval in which the system will perform as required without any emergency or unexpected maintenance being needed, with a specified confidence level. A period of maintenance follows each MFOP, and is known as a Maintenance Recovery Period (MRP). During each MRP, the system is recovered to a state whereby it can complete the next MFOP. It is allowable that the system may carry some faults into an MFOP, as long as they are not critical for that time period.

Research has indicated that the application of technologies such as prognostics (which allow estimation of future failure times) and redundancies (providing extra systems which are typically offline, only becoming operational when the primary system fails, covering for the loss of functionality), are critical in allowing an MFOP of a meaningful size. However, the concept is still relatively young and as such there is very little operational experience of the ways in which the technologies and procedures which will enable it can be applied to yield a satisfactory maintenance-free duration.

1.8 Research Objectives

The aim of this research is to develop methods which would allow an accurate analysis of Maintenance-Free Operating Periods and how they could be applied. This needs to take account of all of the technologies and maintenance schemes which would be critical in providing an MFOP value which is of use to the operator. Some of these reliability technologies are novel and very complex, and the sum total of their effect on a system may be similarly complex and not well understood. The analysis method developed should allow further insight into as yet unforeseen problems which may arise, and contribute to assessments into whether MFOP is a metric which will be useful in future applications.

Also important for consideration is the performance of the missions within an MFOP. These missions are performed sequentially, without emergency maintenance, and so each of them must be successfully completed. Considering each of these as single-phased would not provide an accurate enough figure for

mission reliability, and so a multi-phased mission approach is applied. The analysis method developed must take into account real-world considerations of the multi-phased nature of missions, such as alterations to the mission plan while it is being carried out.

The combination of phased missions analysis with the novel concept of MFOP is one which requires a very powerful way of modelling the reliability performance. The research objectives are to find a modelling tool which will provide this power, and present a modelling method using this tool.

Meeting these objectives involved the following:

- Review of research into Maintenance-Free Operating Periods: the creation and development of the concept, the methods by which it is expected to work, the mathematical and computational models which have been created to analyse it.
- Review of existing methods for phased missions systems: the development of the concept, and the ways in which it has been analysed for both the repairable and non-repairable cases.
- Establishment of a method which allows modelling of MFOP and phased missions in combination, to a satisfactory level of detail.
- Development of a modelling method which considers both these concepts in depth. The complexities of MFOP and phased missions, such as prognostics, redundant systems, and real-time mission reconfiguration should be taken into account.
- Application of the modelling method into a form which allows analysis of a real-world system. This is a computerised form, making it possible either to swiftly calculate end-values, or to simulate system reliability data.
- Demonstration of the modelling method to a real-world system, showing how it can be of use in designing new systems or developing current ones to achieve a high value of MFOP with a high confidence level.

Chapter 2 Reliability Analysis Tools

2.1 Introduction

Predicting the reliability performance of a system is typically done in terms of the reliability performance of the components which make it up. The methods most widely used for this purpose are Fault Tree Analysis (FTA), Binary Decision Diagrams (BDDs) and Markov analysis, which are discussed in this chapter.

2.2 Fault Tree Analysis

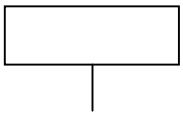
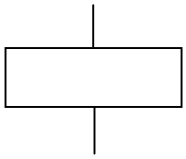
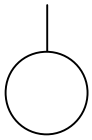
In a fault tree, a specific system failure mode is expressed using a visual symbolic representation of how component failure events can combine to cause it. In this way, FTA allows the causes of system failure and corresponding reliability parameters to be established by a logical and structured process. The method is often employed at the design stage of a system, in order to highlight potential areas for improvement, both in its inherent reliability and the way it is used by operators.

2.2.1 Fault Tree Construction

A fault tree is typically constructed in a top-down fashion. This means that it must begin with the identification of the system failure mode to be analysed. If a system has more than one failure mode, then a different fault tree for each mode is required. The system failure mode becomes the *top event* of the tree, with the branches below it describing the events which could cause this. Each event in the tree has the causes for its occurrence ascertained, gradually increasing in resolution until each branch terminates with the failure of a component, known as a *basic event*. *Intermediate events* are those which are between basic events and the top event. Each basic event can be described in terms of its failure probability over time, and the combination of these in the specific logic visualised by the fault tree establishes the failure rate of the top event.

Events in a fault tree are linked logically by *gates*. Each gate is a way of establishing how an intermediate event can be caused by basic or other intermediate events of a higher resolution. The symbols for events used in fault trees are shown in Table 2.1. Table 2.2 shows the symbols for gates, of which there are three types: AND, OR, or NOT, which represent the Boolean operations of "intersection", "union" and "complementation" respectively. A fourth type is also shown, which is a simplified way of representing an output event occurring when *k* out of *n* inputs have occurred.

Table 2.1 – Event Symbols

Event Symbol	Explanation
	Top Event
	Intermediate event developed by gate below
	Basic event

A system in which failure can only be caused by component failures and in either AND or OR logic is known as a *coherent* system. If at least one component success is used anywhere to define system failure, a NOT gate is required for each success and the system is known as *non-coherent*.

Analysing a fault can be done in two ways. The first identifies the logical combination of component failures which cause system failure, and is known as *qualitative analysis*. The second allows calculation of system reliability parameters using those of components, and is known as *quantitative analysis*.

Table 2.2 – Gate Symbols

Gate Symbol	Gate Name	Causal Relation
	AND Gate	Output event occurs only when all input events have occurred.
	OR Gate	Output event occurs when at least one input event occurs
k n inputs	*k*-out-of-*n* Voting Gate	Output event occurs if at least *k* out of the *n* input events occur
	NOT gate	Output event occurs if the input event does not occur

2.2.2 Qualitative Analysis

This type of analysis is used to logically express the failure of a system in a particular way in terms of the component failures which can combine to cause it. Each individual combination of basic events which will definitely lead to system failure is known as a *cut set*, defined as:

> A cut set is a collection of basic events such that if they all occur, the top event also occurs.

A cut set may contain events which do not contribute towards the system failure mode. If the cut set {A B C D} ensures that the system will fail, but components B and D could cause this alone, then the states of A and C are irrelevant. This allows the definition of a *minimal cut set*:

> A minimal cut set is the smallest combination of basic events, such that if any basic event is removed from the set, the top event will not occur.

If one fault tree shares the same minimal cut sets as another, then they are logically equivalent. The number of basic events within a minimal cut set defines the cut set's *order*. Typically, minimal cut sets of a lower order influence the system failure rate most, and so at the design stage, it is these which should be focused on and removed. If NOT logic is used anywhere within a fault tree, the combinations of basic events which cause system failure are known as *implicants*. Minimal sets of these are called *prime implicants*.

Determining a fault tree's minimal cut sets can be done either in a "top-down" or a "bottom-up" approach, which results in a Boolean expression for system failure in terms of component failures. The "top-down" approach continually substitutes Boolean events lower down the tree for those higher up the tree, so the top event is consistently "broken-up" into smaller events which are all eventually basic. The "bottom-up" approach begins with each set of events for each gate at the bottom of the tree and gradually combines these until the top event is reached. The product symbol, '·' is used to represent 'AND', while the sum symbol, '+' represents 'OR' in logical expressions. The logic is typically expressed in a sum-of-products (s-o-p) fashion from which the cut sets can be found. In order for only minimal cut sets to be found, the s-o-p expression must be minimised by removing any redundancies in accordance with Boolean algebraic laws. See [2] for a more detailed explanation, with examples, of constructing fault trees.

For a complex system there may be thousands of minimal cut sets which can only be analysed by intensive and expensive computation. In these conditions, it is possible to make approximations by removing cut sets above a certain order, or removing those of a low probability while they are being quantified. While this may be faster, it reduces the accuracy of the analysis and produces further error in the quantitative analysis.

2.2.3 Quantitative Analysis

This method of analysing a fault tree allows parameters to be evaluated which describe the reliability performance of the overall system. Important parameters which are widely used in industrial applications are the probability, frequency and expected number of top event occurrences.

2.2.3.1 Top Event Probability

The probability of the top event occurring, which is also the unavailability of the system, can be obtained by combining the failure probability for each event in a minimal cut set in a specific method, known as the *inclusion-exclusion expansion*. The probability that a minimal cut set C_i exists is equal to the product of the probabilities that each basic event within it exists. For instance, if cut set C_i = {A,B}, its probability of occurrence equals the probability of A failing multiplied by the probability of B failing. In general, this is expressed by equation 2.1.

$$P(C_i) = \prod_{c=1}^{N_{C_i}} q_c(t)$$

2.1

As the top event is caused by any of the minimal cut sets occurring, a system with N_{mcs} minimal cut sets will have failure probability at time t, $Q_{SYS}(t)$, equal to the probability of the union of the set of minimal cut sets, as shown in equation 2.2.

$$Q_{SYS}(t) = P\left(\bigcup_{i=1}^{N_{mcs}} C_i\right)$$

2.2

This may be expanded as shown in equation 2.3.

$$Q_{SYS}(t) = \sum_{i=1}^{N_{mcs}} P(C_i) - \sum_{i=2}^{N_{mcs}} \sum_{j=1}^{i-1} P\left(C_i \cap C_j\right) + \ldots + (-1)^{N_{mcs}-1} P\left(C_1 \cap C_2 \cap \ldots \cap C_{N_{mcs}}\right)$$

2.3

If a top event is made up of many minimal cut sets, using the inclusion-exclusion expansion to calculate the top event probability may be unfeasible due to the large number of calculations required. In this case, approximation techniques are employed to find bounds for the parameter.

2.2.3.1.1 Upper and Lower Bounds for System Unavailability

In the inclusion-exclusion expansion shown in equation 2.3, as the number of cut sets being combined increases, the contribution of each term towards the exact figure reduces. The series can be truncated to give upper and lower bounds for the system unavailability, shown in equation 2.4.

$$\sum_{i=1}^{N_{mcs}} P(C_i) - \sum_{i=2}^{N_{mcs}} \sum_{j=1}^{i-1} P(C_i \cap C_j) \leq Q_{SYS}(t) \leq \sum_{i=1}^{N_{mcs}} P(C_i)$$

2.4

Lower Bound Exact Upper Bound

The upper bound is also known as the *rare event approximation* since the rarer the component failures, the closer this figure comes to the exact system unavailability.

2.2.3.1.2 Minimal Cut Set Upper Bound

The Minimal cut set upper bound, Q_{MCSU}, is more accurate than the rare event approximation. It is given by equation 2.5.

$$Q_{MCSU} = 1 - \prod_{i=1}^{N_{mcs}} [1 - P(C_i)]$$

2.5

2.2.3.2 Importance Measures

A valuable exercise is the calculation of how dependent the system is on a particular component, in terms of its reliability. *Importance measures* may be calculated, which assign a value to basic events or cut sets. These express in various ways the likelihood of a component failure causing system failure. There are several different types, used in different ways, which can be broadly categorised in two ways: *deterministic* and *probabilistic*.

Deterministic importance measures do not consider component failure probabilities. Instead, the structure of the system is analysed and the value of the importance measure stems from that. An example of one of these is the *Structural Importance Measure*, given by equation 2.6.

$$I_c^{ST} = \frac{\text{number of critical states for component } c}{\text{total number of states for the } (n-1) \text{ remaining components}}$$

2.6

A critical state for component c is one where the other (n–1) components are in a state such that the failure of c will cause the system to transition from a working to a failed state.

A major drawback of this type of importance measure is the lack of consideration shown to the component failure rates. As such, probabilistic measures of importance are typically of more use than deterministic measures.

Probabilistic importance measures take account of the contribution of component failures to system failures in terms of the component failure and minimal cut set occurrence probabilities. As mentioned above, they come in two types: those concerned with either system unavailability or system unreliability. The Birnbaum measure of importance, $G_c(\boldsymbol{q}(t))$, also known as the criticality function, is defined as the probability that the system is in a critical system state for component c. It can be determined using equations 2.7 or 2.8.

$$G_c(\boldsymbol{q}(t)) = Q_{SYS}(1_c, \boldsymbol{q}(t)) - Q_{SYS}(0_c, \boldsymbol{q}(t))$$

2.7

where $(1_c, \boldsymbol{q}(t)) = (q_1, \ldots, q_{c-1}, 1, q_{c+1}, \ldots, q_n)$ component c failed

$(0_c, \boldsymbol{q}(t)) = (q_1, \ldots, q_{c-1}, 0, q_{c+1}, \ldots, q_n)$ component c working

$$G_c(\boldsymbol{q}(t)) = \frac{\partial Q_{SYS}(\boldsymbol{q}(t))}{\partial q_c(t)}$$

2.8

The criticality measure of importance is defined as the probability that the system is in a critical state for component c, and component c has failed. This is weighted by the system unavailability and is shown in equation 2.9.

$$I_c^{CR} = \frac{G_c(\mathbf{q}(t))q_c(t)}{Q_{SYS}(\mathbf{q}(t))}$$

2.9

The Fussell-Vesely importance measure [8] is defined as the probability of the union of the minimal cut sets C_k containing c given that the system has failed. Equation 2.10 shows this.

$$I_c^{FV} = \frac{P\left(\bigcup_{k|c\in k} C_k\right)}{Q_{SYS}(\boldsymbol{q}(t))}$$

2.10

The Fussell-Vesely Measure of Minimal Cut Set Importance is defined as the proportion of times a cut set C_k causes system failure. This is calculated as in equation 2.11.

$$I_{C_k}^{FV} = \frac{P(C_k)}{Q_{SYS}(\boldsymbol{q}(t))}$$

2.11

2.3 Binary Decision Diagrams

Fault trees and the analysis methods used to solve them can become very large and inefficient when the systems they are applied to are themselves large or complex. Instead of analysing in this way, it may be more efficient and worthwhile converting the fault tree to a Binary Decision Diagram (BDD), and then performing analysis. Initially developed by Rauzy [9], a BDD in reliability analysis allows the top event failure logic to be expressed in the form of a Boolean equation. This Boolean equation can be solved in a much simpler way than the inclusion-exclusion expansion shown in equation 2.3, and so they are a preferred option in reliability

analysis. The method allows for both qualitative and quantitative analysis, with exact solutions being found without the need for approximations as are standard in the fault tree approach.

2.3.1 BDD Properties

A BDD is a directed acyclic graph, made up of both terminal and non-terminal nodes (also called vertices) connected by branches. Its acyclic nature means that paths are followed in one direction only, without any loops. Non-terminal nodes represent basic events of a fault tree, while terminal nodes express the final state of the system. This state is binary, where a 0 represents "system works", and 1 represents "system fails". An example of this is shown in Figure 2.1.

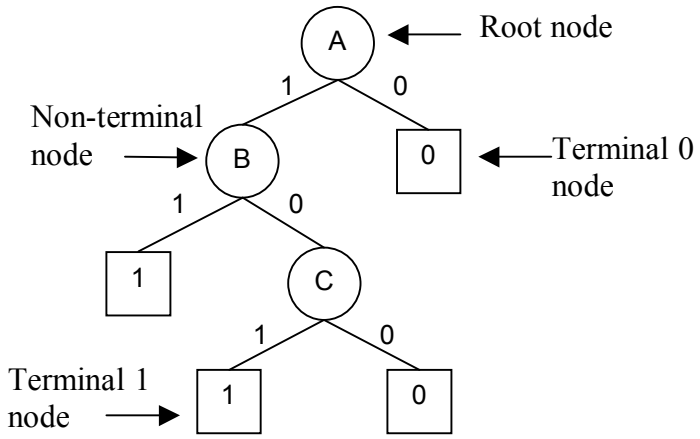


Figure 2.1 - Example Binary Decision Diagram

Every root and non-terminal node has two branches underneath. The left branch, labelled '1', represents the occurrence of the basic event (that is, a component has failed). The '0' branch represents the component's success. The size of a BDD is expressed in terms of the number of non-terminal nodes.

Each path through a BDD begins at the root node and runs through a series of non-terminal nodes until a terminal node is reached. If a path ends with a '1' node, a cut set of the system can be described by including in it all those basic events through which a '1' branch was followed. For instance, in Figure 2.1, two paths end in

terminal 1 nodes, A→B and A→ $\overline{B}$ →C. By ignoring the success of B in the second path, the cut sets become {A,B} and {A,C}.

In order to create a BDD for a system, the basic events need to be considered in an order. An order can be chosen which will create the best size of BDD and speed up analysis; however, if the order is poorly chosen, a large BDD can result. Sinnamon and Andrews [10, 11] have researched both the qualitative and quantitative aspects of BDDs.

2.3.2 Formation of a BDD using If-Then-Else Structure

Developed by Rauzy [9], the **i**f-**t**hen-**e**lse (ite) method of converting fault trees to BDDs takes each gate in the fault tree in turn, beginning with the top event. These are expressed in the form of a Boolean function, $f(\mathbf{x})$, pivoted about any variable X1. Shannon's formula can be expressed as in equation 2.12:

$$f(\mathbf{x}) = X1 \cdot f1 + \overline{X1} \cdot f2 \qquad \textbf{2.12}$$

where f1 and f2 are functions with X1=1 and X1=0 respectively.

The **ite** structure for this is represented as **ite**(X1,f1,f2), and is explained as "**if** X1 fails, **then** consider f1, **else** consider f2". In the BDD, a '1' branch of X1 will achieve f1, while a '0' branch achieves f2. These represent "X1 occurs" and "X1 does not occur" respectively, as shown in Figure 2.2.

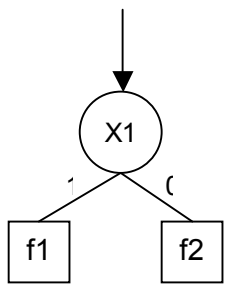


Figure 2.2 – BDD Vertex of ite(X1,f1,f2)

When beginning the process of constructing a BDD, each basic event x is given the structure **ite**(x,1,0). Each gate in the fault tree is then considered using a bottom-up approach, with the following rules applied:

If $X < Y$ $\quad J \oplus H =$ **ite**$(X, f1 \oplus H, f2 \oplus H)$

If $X = Y$ $\quad J \oplus H =$ **ite**$(X, f1 \oplus g1, f2 \oplus g2)$

where $J =$ **ite**$(X, f1, f2)$ and H = **ite**$(Y, g1, g2)$ are the gate inputs.

An advantage of this method is the automatic elimination of any repeated nodes.

2.3.3 BDD Minimisation

If the ordering of the variables is not optimal, the BDD and the calculated cut sets will not be minimal. Rauzy [9] developed a minimisation process which will create a minimal BDD which represents the minimal cut sets of the fault tree. Each node in the BDD has a general **ite** representation of

$$F = \textbf{ite}(x, G, H) \tag{2.13}$$

If δ is a minimal solution of G, and δ is not a minimal solution of H, then the intersection of δ and x, ($\{\delta\} \cap x$), will be a minimal solution of F. The set of all minimal solutions of F, $sol_{min}(F)$, will also include the minimal solutions of H, and can be expressed as in equations 2.14 and 2.15.

$$sol_{min}(F) = \{\sigma\} \tag{2.14}$$

where

$$\sigma = [\{\delta\} \cap x] \cup [sol_{min}(H)] \tag{2.15}$$

Rauzy also defined the *without* operator, where all minimal solutions of G which are also minimal solutions of H are removed. By removing all minimal solutions of G which are also minimal solutions of H, equation 2.15 is minimal.

2.4 Markov Analysis

Markov methods are used where components of a system have strong dependencies between each other. They allow the calculation of the reliability and availability, but are more useful for smaller systems, due to a size explosion as systems grow larger. The Markov approach assumes that a system's future state depends only upon its present state, not on any states it has gone through previously. Thus, each event is dependent only upon the present system state, and not upon any previous events. The approach also assumes that the rate of switching from one state to another is constant, and thus allows a solution for the reliability characteristics to be calculated.

A Markov model consists of two parts: states and transitions. States describe the operation or failure of components in the system at the current point in time, while transitions describe the rate at which the switch from one state to another is possible. If states are not linked by a transition, the system cannot go from one state to the other directly. Some states have no outgoing transitions, in which case they are described as *absorbing* states.

The method described below assumes that there are a fixed number of discrete system states and that the transition rate between them is either zero (impossible) or constant. These rates thus imply that component failure and repair times follow the exponential distribution. Thus, the Markov method used here is discrete in space and continuous in time.

The starting point of Markov analysis is typically the construction of a Markov model. This is a directed graph where each node represents a system state, and the edges between them represent the transitions in the direction of the arrow, with associated rates. This results in a set of equations of the form of equation 2.16.

$$[\dot{\mathbf{P}}] = [\mathbf{P}][\mathbf{A}]$$

2.16

where $[\dot{\mathbf{P}}]$ is the instantaneous probability per unit time of being in each state.

In general, the matrix **A** is found from the Markov diagram, by using the following rules:

- The matrix is square, with the number of rows and columns equalling the number of states.
- The sum of each row is zero.
- All non-diagonal elements in row *i* and column *j* represent the transition from state *i* to state *j*.
- All diagonal elements *ii* represent the total transition rate out of state *i*.

As the Markov state equations are linear differential equations with constant coefficients, one method of solving them is by using Laplace transforms. However, where a Markov model is large or complex, it is typically more suitable to solve the resulting large set of N_s differential equations using numerical methods. Equation 2.16 may be expanded to give equation 2.17.

$$\begin{bmatrix}\dot{P}_1 & \dot{P}_2 & \cdots & \dot{P}_{N_S}\end{bmatrix} = \begin{bmatrix}P_1 & P_2 & \cdots & P_{N_S}\end{bmatrix}[\mathbf{A}]$$

2.17

Because $$\dot{P}_i(t) = \frac{P_i(t+dt) - P_i(t)}{dt},$$

2.18

Equation 2.17 may be rewritten as equation 2.19.

$$\begin{bmatrix}P_1(t+dt) & P_2(t+dt) & \cdots & P_{N_S}(t+dt)\end{bmatrix} = \begin{bmatrix}P_1(t) & P_2(t) & \cdots & P_{N_S}(t)\end{bmatrix}[\mathbf{I} + [\mathbf{A}]dt]$$

2.19

The general numerical solution to the set of differential equations is therefore given by equation 2.20.

$$[\mathbf{P}(t+dt)] = [\mathbf{P}(t)][\mathbf{K}] \qquad \text{where} \quad [\mathbf{K}] = [\mathbf{I} + [\mathbf{A}]dt] \tag{2.20}$$

This leads to a recursive solution to the differential equations over a duration of time.

2.5 Summary

This chapter has introduced methods which allow the reliability of a system to be predicted, based on the reliability of its components.

Fault tree analysis provides a logical method of gradually breaking down a specific system failure mode into the sub-system and component failures which may build up to cause it. This approach considers that a failure typically occurs either when one of many possible causes occurs (OR logic), or all of several causes occur (AND logic). This structure allows the calculation of cut sets, which are combinations of basic events which will lead to system failure. Minimal cut sets are the necessary and sufficient forms of these, with repetitions and unnecessary events removed. The failure probabilities of the cut sets can be combined in an inclusion-exclusion expansion to calculate system failure probability. Importance measures, which describe the contribution of a component or cut set to system failure, can be calculated using the qualitative and quantitative expressions of system failure.

Evaluating system failure probability in an inclusion-exclusion expansion for a large or complex system is time consuming and inefficient. A Binary Decision Diagram method has been created to speed up the process. A fault tree can be converted to a BDD, which expresses the top event failure logic in the form of a Boolean equation. This equation can then be solved much more simply and quickly than a standard fault tree evaluation technique, without approximations being needed.

Markov analysis expresses the state of a system in terms of the combinations of the states of the components within it, typically categorising them as “operational” or “failed” states. The initial conditions and the rates between the states determine the

probability, at a given point in time, of being in any of them. A matrix set of equations can be used to express the set of state probabilities and transition rates, which can then be solved using analytical techniques such as Laplace transforms or numerical methods.

Chapter 3 Maintenance Free Operating Periods

3.1 Introduction

In 1996, the Ministry of Defence introduced the idea of the Maintenance Free Operating Period (MFOP). This was to satisfy what it saw as the needs of the typical military customer, which were centred on improved operational reliability & availability and the need to reduce costs. The supplier guarantees the customer that for a specified proportion of the operational time, the overall system will not completely fail or lose enough functionality such that it cannot carry out all its assigned missions. After this there is a period of maintenance where the system is recovered to full capability.

This chapter aims to expand in detail the aspects of MFOP, both basic information and the technologies which will allow it to work. It will explain some of the advantages and disadvantages that introducing the concept may create, and state some of the arguments and opinions regarding MFOP. Also presented is an investigation of the factors that will affect its implementation, in design, manufacture, operation and maintenance and how the literature published to date has attempted to mathematically and computationally model MFOP, with explanations of some of these analysis methods.

Note: The concept of MFOP was originally intended to be applied to the field of military aircraft, although it is general and can be applied to many other fields. Confusion can, however, arise in the use of terminology: a “system” can mean either an overall item, such as a single aircraft, or a sub-system of it (such as landing gear). In keeping with the aerospace industry, and to avoid this confusion, the term “platform” is used to refer to the overall system, such as an aircraft. “System” refers to one of the individual parts of the platform, such as a fuel or a weapons system.

3.2 Explanation of MFOP

The constant drive for improvement in all areas of engineering has resulted in an increased focus on reliability and availability. As a result, in 1995 the Ministry of Defence (MoD) looked at the needs of the modern military aircraft customer and looked at ways of providing a platform with the necessary technologies to satisfy these. These needs include dependable systems that work when required for a long period of time, with failure- and maintenance-free periods in order to carry out all missions without any loss in capability or functionality. The equipment must be affordable, not just in terms of acquisition costs but whole life cost (WLC), considering reduced cost of operation and support to match reduced funding and number of personnel (i.e. value for money). Also important is the ability to plan and predict, with high probability of success, all aspects of missions and mission support, allowing for quick deployment and the spreading of fighting across several areas, providing for increased emphasis on expeditionary warfare. The overall goal is, in summary, smaller, resourceful fleets that are more capable and cheaper to maintain.

Some of the needs mentioned above, such as dependable systems and good WLC are already important considerations of the current schemes in operation. Maintenance Free Operating Periods were first proposed by the MOD at a seminar in 1996 as a way of providing for all of the needs listed above [12]. The MFOP is defined, in [7], as:

> The period of operation during which an item will be able to carry out all its assigned missions, without the operator being restricted in any way due to system faults or limitations, with the minimum of maintenance.

In effect this is a period in which there are no failures of any type which can cripple the platform or prevent it from carrying out its mission specifications. Accompanied with this is a probability of the MFOP being completed, sometimes called MFOPS (MFOP Survivability). MFOP considers the overall platform survival, so as long as

sufficient failure tolerance is built in, individual system failures can occur if they do not affect operations. Although MFOP may become a prime metric in reliability engineering, it will not necessarily improve the **reliability** of the platform's systems – it may in fact be reduced.

A particular MFOP would require, for example, 12 days of continuous operation. After this time it is anticipated that the platform will have built up component or system failures that have not affected its functionality or caused mission failure but which require attention prior to the next MFOP. The period during which the majority of maintenance will take place is called the Maintenance Recovery Period (MRP). It is intended that this period will not end up being so long as to inhibit operations but will efficiently consolidate all necessary maintenance actions into the same time interval, hence saving on costs. After each MRP the platform is ready for completion of another full MFOP.

An MFOP is not defined to contain absolutely no maintenance; rather it is considered that minor actions, such as refuelling, rearming and repairing important safety related features, will still need to take place.

3.3 MFOP Basics

Over time the definition of MFOP has changed and is now more pragmatic in nature. The Committee for Defence Equipment Reliability and Maintainability (CODERM) and the Ultra Reliable Aircraft (URA) consortia agreed definition is:

> A period during which the system will operate without failure and without the need for any maintenance, however, faults and minor planned, contractually agreed maintenance are permissible. [21]

This definition may further mature over time and may need to be specifically documented in contracts.

MFOP is driven by a “weakest-link” style hierarchy – the lowest system MFOP will decide the platform MFOP. Improving this will therefore improve the platform MFOP [13]. As Figure 3.1 shows, the platform takes the MFOP level of system D, which has the shortest length, rather than A B or C. By improving system D, the platform MFOP will increase.

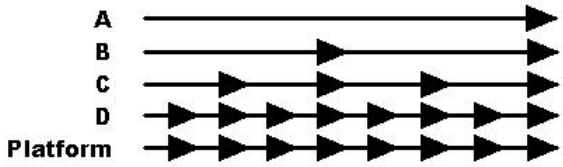


Figure 3.1 – Nature of MFOP Platform

The changes to the reliability specification are largely technology driven and so are likely to be carried out over the longer term. A platform would need to be designed to operate MFOPs and so it will be realistically impossible to introduce MFOP into the current generation [14]. Use of MFOP does not remove the need for a “faults per hour” parameter, even if faults are not causing system failure.

3.4 Impact of MFOP

3.4.1 Technologies Within MFOP

MFOP has the potential to be used as a means of significantly improving the operational capability and platform-level reliability of the platform applying it and therefore provide a way of better meeting the customer’s needs. There are various potential improvements, listed below and outlined in detail in Section 3.5.1. These are regarded as beneficial in creating a consistent period during which high likelihood, critical system failures will not occur.

One of these improvements involves understanding and increasing the reliability and availability of each component and system. Detailed research, investigation and effort will allow failure modes to be removed or improved, while knowledge of components' failure time distributions can be updated. Where this method is

restricted by cost or time constraints, redundant components and systems can be used to back up those that are more prone to failure, or which have unacceptable consequences upon failure. With mechanical equipment this may be inapplicable due to an unacceptable weight or bulk penalty, but avionics are able to use this idea. An extension of this is reconfigurable systems, which can adapt their behaviour either to take over the operation of a failed module or sub-system, or continue their current operation, allowing for the fact that another module has failed.

Other technologies include prognostic systems to predict the future failure of a component or system, using Health Monitoring and accurate failure time distribution knowledge to allow improved estimation of the time of failure. Similarly, diagnostic systems detect and locate the components that are currently failed, perhaps with the capability to relay this information back to main base to allow efficient planning of maintenance operations. Providing maximum support for inspection, such as ease of access to systems and improved Built In Test Equipment, will shorten the MRP, increasing availability and reducing costs.

These technologies are not new and are not exclusive to MFOP, but the expectation is that the setting of an ambitious goal, such as a guaranteed maintenance-free period, will drive all those involved with the development and operation of the platforms towards the ambition of having one which requires no maintenance at all.

3.4.2 Advantages

The key advantages of introducing MFOP (from the supplier's point of view) will be those that create or increase consumer interest in their product. In this case the primary advantages will be linked to improving the purchase and whole life costs of the platform or providing operational benefits.

Examples of the advantages which using MFOP will confer include the predictability of maintenance periods which allows for overheads such as manpower and facilities to be used less often and with more planning, with little probability of emergency or reactive maintenance. Facilities need not be available throughout a

mission or series of missions, saving on costs or allowing for greater flexibility in mission planning. Another cost-related benefit will be realised if maintenance is made to be more efficient, by employing more powerful diagnostic tools, or planning repairs and spares better. Details of failures discovered mid-mission could be transmitted back to the repair facility, allowing for spares and manpower to be made available.

Successful application of MFOP will create a better chance of completing a mission and will reduce the time spent in the failed state (improving availability). If this is the case, it may be possible to perform missions to the same standard with a smaller fleet, as fewer backup platforms are necessary to cover for any one becoming inoperable. Also, if the platform is more reliable then its availability and effectiveness will be increased. This is key for the military customer whose funding partially rests on ability to perform.

Having a system which is very reliable and well understood will allow good prediction of its failure time. The key of MFOP is that the platform only needs to maintain its ability to operate for the next period. If prognostic systems predict that a system is not going to last for the duration of the next MFOP then it can be replaced or corrected before dispatch. Although this will lead to components or systems potentially being scrapped before failure, it is thought that the consequent improvement in reliability and reduction in necessary manpower can reduce costs enough to overcome the increase in throughput of spares. The resultant increase of "wasted" life of a component and overuse of spares can be reduced by more accurate health monitoring.

3.4.3 Disadvantages

While there is much that could be gained from the use of MFOP, there are also potential downsides. For instance, as shown in Figure 3.1, MFOP is based on a hierarchy whereby the system with the smallest MFOP dictates the MFOP of the platform. This allows for the most unreliable system to be improved, after which the system with the next lowest MFOP is improved, and so on. This method of

improvement was adopted by the Germans during WW2 while working on their V2 rockets. Their eventual quality was poor, as they did not focus on the overall reliability of the platform but its constituents. This approach therefore has a historical basis to be considered as weak.

The large level of improvements and technologies that will have to be developed to allow the MFOP concept to be successful is very ambitious and will take much time, money and effort to bring about. Contracts requiring MFOP before understanding of it is mature will only lead to strained relations between customer and supplier. All stakeholders must have knowledge of the starting situation and the timeframe of improvements. However, the addition of complex technologies to a system may well lead to a reduction of quality and availability unless the complexity is understood and allowed for. New technologies typically feature new components whose reliability is not well understood, and so a reduction in reliability or availability in the short term may result. Also, there may well be unforeseen problems which appear when MFOP is being implemented.

3.4.4 Arguments Surrounding MFOP

The novelty of the concept of MFOP will inevitably lead to a debate within the target industry (military aircraft) over whether it is a good thing or not. There are several aspects of this that are worth stating. The MoD papers that have been published to promote the concept of MFOP, particularly Knowles [15], Appleton [13] and Turner *et al* (1997) [14], have argued that the use of the term "Mean Time Between Failures" (MTBF) has a negative effect on reliability engineering. Some of their arguments include

- Use of MTBF implies the use of the negative exponential distribution and constant Rate of Occurrence of Failure (ROCOF)
- Use of the term indicates that engineers have no desire to understand the causes of random failures, and that as a mean level of failures per unit time is contracted for, failures are 'allowable'

- As a result of not understanding when failures are going to occur, maintenance support is required to be available round the clock, wasting time and money.

MTBF is the average time across any given distribution of failure times, not just the exponential distribution. Efforts to improve the MTBF level require an improved understanding of cause, and given that it is impossible to completely eradicate failures, allowances must be made for this. Even with an MFOP methodology in place, it is unrealistic to expect a perfect, accurate prediction of exactly when a failure will occur. As such it will always be necessary to have maintenance crews on standby for situations where emergency repairs must take place. Stating that the designers do not consider failures to a full extent or that the motivation to understand failures is lacking (a charge made by Knowles [15] against electronics designers), will irritate members of the industry who *are* working to these goals, and help to instil a resistance to the proposed idea.

The lack of objectivity of some of the statements made does little to convince others in the industry that MFOP is a worthwhile cause. It is likely that the best way to inspire confidence in MFOP is to outline the proposed benefits, rather than to focus on the perceived failures of the current methods. The most successful reports in getting the necessary ideas across are those that concentrate purely on MFOP, its advantages, how it would work and how it is to be studied, evaluated and implemented.

3.5 Using and Achieving MFOP

The published research investigating MFOP can be split into two broad camps: those advocating a sea-change in the current methods and cultures by adopting MFOP in its entirety; and those that want to use the current reliability information, such as hazard rate, to estimate MFOP.

Several mathematical models (detailed in Section 3.6.1) have been put forward which aim to estimate MFOP based on a components' MTBF or Weibull

distribution variables [16, 17]. However, these models do not consider many issues which are of concern in application, such as environmental effects. This is understandable given their infancy, but there will always be a limit on the level of detail that can be included in mathematical models.

Others [12, 14, 18] tend towards the argument that every component's exact distribution and parameters, should be found as accurately as possible and the approach should be more investigative and considered, with a design-led desire to improve reliability in all areas of engineering. Whilst this would be worthwhile, there are several constraints such as cost and lack of availability of failure data which prevent the possibility of this being fully realised.

3.5.1 Enablers of MFOP

There are many different ideas and technologies that will contribute to prolonged maintenance-free operation of a platform. These are outlined in the sections below.

3.5.1.1 Inherent Reliability (IR)

Bringing about a lengthy MFOP will require, in part, that the components and systems have the best reliability levels that are feasible to obtain with the time and financial constraints in place. It is thought that the technology currently employed would not yield a long MFOP at an acceptable probability [19]. Improving component reliabilities through tests and investigations into failure modes, etc., will allow the supplier not only to understand their components better but will also assist the other enabling technologies in providing a more integrated set of reliability management systems.

The desired outcomes of any tests and investigations into component reliabilities will be:

- To find all the failure modes, and for frequent or catastrophic events, to eradicate either the effects or the mode itself. This is generally performed by

Failure Modes Effects (and Criticality) Analysis (FME(C)A) [19] and would ideally lead to a high mean and a low variance in the reliabilities of items.

- To understand the physics of failure [15] – that is, the absolute root causes of or descents to failure – of every component.
- To find the true failure distribution of each component in the system, and obtain accurate parameters for these distributions. If this were achieved, a much better assessment of the overall system failure rate could be realised.
- To understand the effects of environmental parameters, such as vibration or temperature, in degrading a component or causing it to fail. This will allow more complete computational or mathematical models to be produced for analysis of system reliability.

These outcomes do not stand alone – they will link into one another as they also interact with the other enablers. Investigating failure modes and then attempting to minimise them or eradicate them will require investigations into how a component is to be used in service and how the environment it is subjected to can contribute to its failure; these environmental issues may affect which distribution is thought to be the governing one, and so on.

The testing of components to find all this information is a difficult problem – it is not typically feasible to wait for a sufficiently large number of components to fail in normal operation, even though this would provide the best quality information, due to the large time to failure of most components. Other ways of testing components in laboratories to estimate this information do not always simulate operational conditions and involve some way of "speeding up" the process. These methods include Reliability Demonstration or Reliability Growth Testing, Accelerated Life Testing and Fatigue Testing [20].

The problem of investigating times to failure has led to the creation of several groups with this purpose in mind. These are such as the Society of British Aerospace Companies (SBAC) and the Center for Advanced Life Cycle Engineering (CALCE) in the USA.

3.5.1.2 Prognostics

Prognostics systems are those that allow the future failure of a component to be predicted. They are considered an important part of the MFOP concept. Prognostics technology draws on information regarding components' failure life characteristics (FLCs), if they exist, and good knowledge of their physics of failure and failure time distributions. In order to detect a future failure, there needs to be either a understanding of the signs that are shown by components before they fail or, in the event of invisible or no signs of deterioration, an understanding of the conditions under which the component is likely to fail faster. Electrical components are more likely to fail spontaneously, whereas mechanical parts are prone to visible wearing. Items that visibly wear will be regularly inspected, and if the remaining life for a part is thought to be too low, that part is replaced.

Lack of signs of wear is being dealt with by the introduction of Condition or Health Monitoring (HM) systems. These systems consider various aspects of a component's usage, such as the value of the component's relevant reliability factor (flights, hours, etc), the types of excursions taken, environmental factors and so on. The HM system should then be able to use these figures and calculate the effects on the aeroplane system, and allow for better scheduling of replacement. A reliable HM system should keep the level of premature replacement and overuse of spares to a minimum. More research into the effects of various factors on the health of systems is needed before HM becomes accurate [21].

According to Cini & Griffith [20], because mechanical components are generally more bulky and difficult to replace than other components, any decisions made by a HM system in requiring replacement must be both trusted and accurate ("full authority") in their analysis of whether to remove a mechanical item – the advice must not be merely advisory. They state that the helicopter industry has good experience in condition monitoring of mechanical parts and may be able to lend itself to the MFOP philosophy.

3.5.1.3 Diagnostics

Diagnostic systems are those which check for and locate current failures in a system or platform. Many failures are found through inspection by a maintenance engineer. Built In Test (BIT) systems [20] have been introduced which perform tasks such as identification of faults, without the need for extra personnel. As the name suggests, they are built in to the system, thereby easing the diagnosis of faults. However, a problem can occur with BIT systems if they have a high "No Faults Found" (NFF) rate. This is a figure which indicates the accuracy of the BIT equipment, as they sometimes report faults when none have occurred, increasing the cost of maintaining the platform through unnecessary replacement of operational items. As is the case with some commercial aircraft, such as the landing gear control and interface unit of the Airbus A320, BIT equipment can be set to control the automatic change-over to a standby system.

To assist inspection and maintenance, the most unreliable components and those that require inspection from an engineer will need to be easily accessed and removed, well-mounted and connected to allow ease of test [22]. Warrington *et al* claim that if a technician has access to more information, they have to perform less work in diagnosing and fixing faults and so need not be as capable [23]. The authors imply that too much reliance on technician's skill in handling faults will lead to more mistakes through human error than automating the process.

3.5.1.4 Redundancy/Reconfigurability

The MFOP concept allows for items to fail as long as the overall platform continues operation. Failure tolerant items, which involve either redundancy or reconfigurability, will consequently be a key aspect.

The time of failure of a particular component cannot be specified exactly. If the risk of a component failure is high, either due to probability or consequence, redundant or backup systems can be provided in parallel to take over operation in the event of a failure, such as two electrical power supplies. Some components already contain redundant sections (e.g. dual rams on a hydraulic actuator). However, redundancy

for mechanical components is more difficult to implement than electronics due to the mass/volume/cost implications [24].

Accompanied with redundancy is firstly a diagnosis system that can reveal the failure if it has not been revealed through impairment of performance, followed by some method of switching off the failed system and switching on the backup. This could be manually performed by an operator or automatically managed by a computer. The backup can be either 'cold' (cannot fail while remaining unused), 'hot' (operates with normal reliability while in backup state), or 'warm' (reliability figures are in between hot and cold). Newton [21] suggests that hot standby would give the smoothest switching, but mentions that this creates problems with establishing which system is faulty.

Despite being widely regarded as a 'good' concept, redundancy has not been extensively researched and so possible problems have not been fully investigated [21]. Potential difficulties include a common cause fault damaging both the main and back-up systems (e.g. a power surge) or switching problems. The cost/benefit of redundancy and its balance with the cost/benefit of improving inherent reliability will need to be carefully considered as sometimes money invested in redundancy could be better used elsewhere Throughput of spares would be expected to rise with increased redundancy, so the costs of this may be deemed unacceptable [24].

Reconfigurability is a more high-tech area of redundancy provision. Reconfigurable items are capable of adapting their functionality either to carry out the function of a failed item or continue functioning, taking the item's failure into account. Integrated Modular Avionics (IMA) is an area which seeks to exploit this idea, by having standby processing modules which can be switched over or combined in the event of failure of a different module. The tasks performed by each module must be managed by an intelligent management or overview system to control the switching [24].

3.5.2 Considering MFOP in Design

Those endorsing MFOP are keen to emphasise the importance of the design process. Money spent on improvements or considerations at the design stage goes much further than that spent at later stages in the development of new technologies. The various papers published on aspects of MFOP indicate a range of factors and possible problems affecting design and reliability.

Achieving a good balance between the various enablers will reduce costs. Efforts to improve inherent reliability will only go so far before they become too expensive, and so effective prognostic systems or good redundancy may need to be employed [24]. Some component failure modes may have no effect on the system at different phases of flight – a fuel valve failing open mid-flight may not cause a problem while a valve failed closed could; an open-valve failure may however present difficulties for refuelling where a closed failure would go unnoticed. FME(C)A can find root failure modes and causes, and coupled with a detailed knowledge of the parameters and effects of the intended operating environments and typical usage, these can be exploited to great effect [14].

Parameter variations such as tolerances, drift, ageing, etc. will naturally occur during the life of the platform, and can cause failures. The ability to cope with varying environmental stresses must be built in: as benign an environment as possible should be provided for the equipment. Simplicity in design is important, even with the various enablers considered - if new technology or components provide no discernible advantage over older technology, then the proof of the older items in service means it should be retained. Conversely, equipment with known problems should be removed from the design at the earliest opportunity [22].

Design engineers should consider the MRP and attempt to ensure low variation in its length [19]. Maintainability is a key part of MFOP – equipment which is more prone to failures will need to be accessed easily and be well-mounted and connected to allow for quick replacement [22].

Relf [24] considered a design methodology with relevance to MFOP. The process consists of:

1. Laying out the system architecture of the platform so it can be analysed
2. Finding the platform and system MFOP values
3. Identifying the "weak link" system MFOP; that is, the lowest-value system MFOP
4. Applying one or more enablers to it or adapting the system architecture to improve the low system MFOP and thereby improve the platform MFOP
5. Another analysis of the platform MFOP can then be performed and the next weak link found and improved.

As discussed in section 3.4.3, this may not be the most effective way of designing systems to perform MFOPs.

3.5.3 Maintenance Recovery Period

The MRP is a period during which the platform is brought back to a state where it can fully complete the next MFOP with the required level of confidence. All maintenance, except specific activities noted in the customer's contract, will take place during the MRP. It will consist of inspection, monitoring, overhaul or testing.

The diagnostics systems on the aeroplane, when they detect a failure, should display the failure data to the pilot. This can then be communicated back to base, either by an automated procedure (such as Airbus' Future Aircraft Navigation System transmitting Centralised Maintenance System data [20]) or by the pilot. A thorough knowledge of failures, and their effect on current and future missions will allow the operators to plan what maintenance will be needed at the end of the MFOP. This will allow deployment of manpower, spares and facilities 'just in time' [13]. Advance knowledge is key to maximising efficiency, in terms of both missions and maintenance planning, although failures occurring towards the end of the MFOP will not be handled with as much efficiency. The diagnostics systems (such as a

“simple” Troubleshooting Manual or an inbuilt system) have a strong requirement placed on them to be accurate.

The MRP should not be planned to be too long or too short. Due to the different nature of problems that will arise, the length will vary, especially when performing structural inspections. The integration of these inspections (which fit the MFOP philosophy very well) into the MRP will need investigation. The MRP may get longer as the vehicle ages, especially if it is thought to have a non-constant hazard rate [20]. The quality of inspections and the effect of false positives and negatives on the cost of maintenance within the MFOP framework is investigated by Dagg & Newby [25]. The maintenance itself has various challenges, including prioritisation of tasks, allocation and scheduling of resources, and fault visibility and isolation. Priority can be ordered based on the criticality of systems or components with regard to system failure; or by taking into account issues such as the duration of repair, access limits, resources and costs.

Line Replaceable Items (LRIs) are groups of components or other LRIs created to ease repair, and may exist across several systems. An LRI containing other LRIs is known as a ‘parent’, while one which exists in another LRI is a ‘child’. Replacing the parent replaces all the children, while a child can be replaced individually [26]. However, the concept of an LRI means that if one is found with a failure, it is replaced without requiring much knowledge of the actual problem, making it difficult to design out. The removal of LRIs will lead to them being stripped, rebuilt and tested which can lead to more component failures than might have occurred if the LRI had not been removed. More maintenance may therefore lead to more spares being required [16]. Not isolating a failure down to its root cause due to time or cost restrictions often results in several Line Replaceable Items (LRIs) being replaced at the same time and a higher NFF rate in BIT systems. This means a restriction on the number of LRIs replaced at one particular time, or a ban on certain types of maintenance at certain times, may be needed [23].

3.6 Analysing MFOP

Moving immediately to the MFOP concept with the necessary degree of confidence within defence budgets and timeframes is neither credible nor achievable for current platforms – high-confidence, high-value MFOPs cannot be expected for some time. In the meantime, efforts are being made to estimate a realistic MFOP for a typical platform.

There have been several attempts to produce mathematical models for or simulate MFOP in order to gauge what problems may arise, how easy it would be to predict and determine, how achievable the customer's requirements are and the feasibility of the overall concept. The following sections detail the current research into mathematical modelling of MFOP, and the current URAM MFOP simulation project.

3.6.1 Mathematics of MFOP

The research into mathematically expressing MFOP and discovering it from a system of components of mixed distributions is largely adapted from current methods. It will be necessary for any flaws in the models to be identified well in advance so new methods of theoretically evaluating MFOP can be researched [16].

The mathematics establishes a link between the MFOP and its probability of survival, referred to as MFOPS (MFOP Survivability), $S_i(t+u \mid t)$. The notation represents the probability that item i will survive u units of time (the MFOP) given survival at time t.

The probability, $R_i(t)$, of item i surviving from 0 to t is given in equation 3.1.

$$R_i(t) = \int_t^{\infty} f_i(t)dt$$
$$= 1 - \int_0^t f_i(t)dt = 1 - F_i(t) \text{ for } t \geq 0$$

3.1

And the probability of i then surviving to time $t+ u$ is shown in equation 3.2.

$$S_i(t+u \mid t) = \frac{\int_{t+u}^{\infty} f_i(t)dt}{\int_t^{\infty} f_i(t)dt}$$
$$= \frac{R_i(t+u)}{R_i(t)} \text{ for } u \geq 0$$

3.2

A system comprising of n independently distributed components in series has a survival probability from time t to time $(t + u)$ of:

$$S_{sys}(t+u \mid t) = \prod_{i=1}^{n} \frac{R_i(t_i + u_i)}{R_i(t_i)}$$

3.3

where u_i *and* t_i are individual MFOP and life values for each component i.

Components whose mode of failure is not dependent on age have the survivability function, S_{nar}, as shown in equation 3.4.

$$S_{nar}(t+u \mid t) = e^{-\frac{u}{\mu_{sys}}}$$
$$\text{where } \mu_{sys} = \frac{1}{\sum_{i=1}^{m_f} \frac{1}{\mu_i}}$$

3.4

Increasing survivability is possible either by reducing the MFOP duration u or introducing redundancy. Replacement will have no effect. Also note that if a high-confidence MFOP is to be achieved, the MFOP duration must be much smaller than

the relative MTTF value as Figure 3.2 shows (this applies purely to components with exponentially-distributed failure times). The data used to create Figure 3.2 is taken from [21].

The various enablers that will be in place should lengthen the components' typical times to failure and produce systems whose own times to failure are much greater than this, due to prognostics and redundancy.

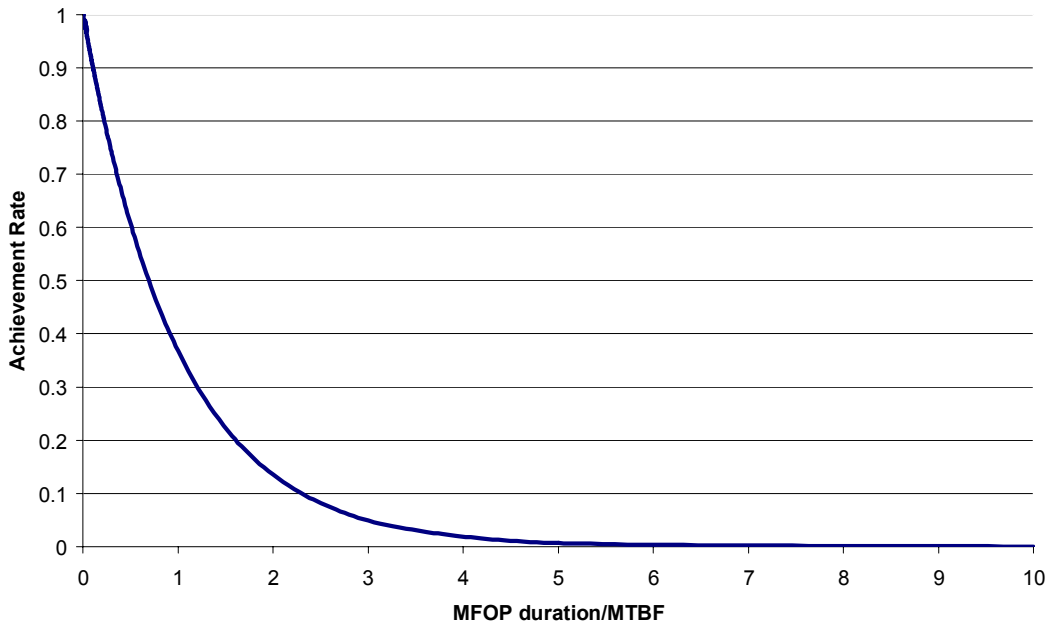


Figure 3.2 – Chart showing Probability of Achieving a Given MFOP Duration

Components whose times to failure are dependent on age are often modelled with the Weibull distribution. The survivability function for component i is given by equation 3.5.

$$S_i(t+u \mid t) = \exp\left[\frac{t^{\beta_i} - (t+u)^{\beta_i}}{\eta_i^{\beta}}\right] \qquad \textbf{3.5}$$

Combining equations 3.4 and 3.5, the survival probability of the system can be estimated as in equation 3.6.

$$S_{sys} = S_{nar} \prod_{i=1}^{n} S_i \text{ (only if components are in series)}$$
$$= \exp\left[-\frac{u}{\mu_{sys}} + \sum_{i=1}^{n} \frac{t_i^{\beta_i} - (t_i + u)^{\beta_i}}{\eta_i^{\beta_i}} \right]$$

3.6

The design life of the system, also called the maximum MFOP length, $u_{\max}$, for a stated value of S_{sys} is the age at which the reliability falls below the designed reliability value, given for the Weibull Distribution [17] by:

$$u_{\max} = \eta_{sys} \left[\ln\left(\frac{1}{S_{sys}} \right) \right]^{\frac{1}{\beta_{sys}}}$$

3.7

Equations 3.4 and 3.5 can be rearranged, to give the survival period for a given level of confidence:

$$u = -\mu_{sys} \ln(S_{nar}) \text{ for } \beta = 1 \qquad \text{[non age-related failures]}$$

3.8

$$u = (t^{\beta_i} - \eta_i^{\beta_i} \ln(S_i))^{\frac{1}{\beta_i}} - t$$
$$\text{for } \beta_i \neq 1 \,\&\, \beta_i > 0 \qquad \text{[Weibull-distributed failures]}$$

3.9

The relationship of MFOP to the Weibull distribution changes with β (the shape parameter). MFOP generally decreases with age, as β is generally greater than 1, but can have other behaviours, as Figure 3.3 shows.

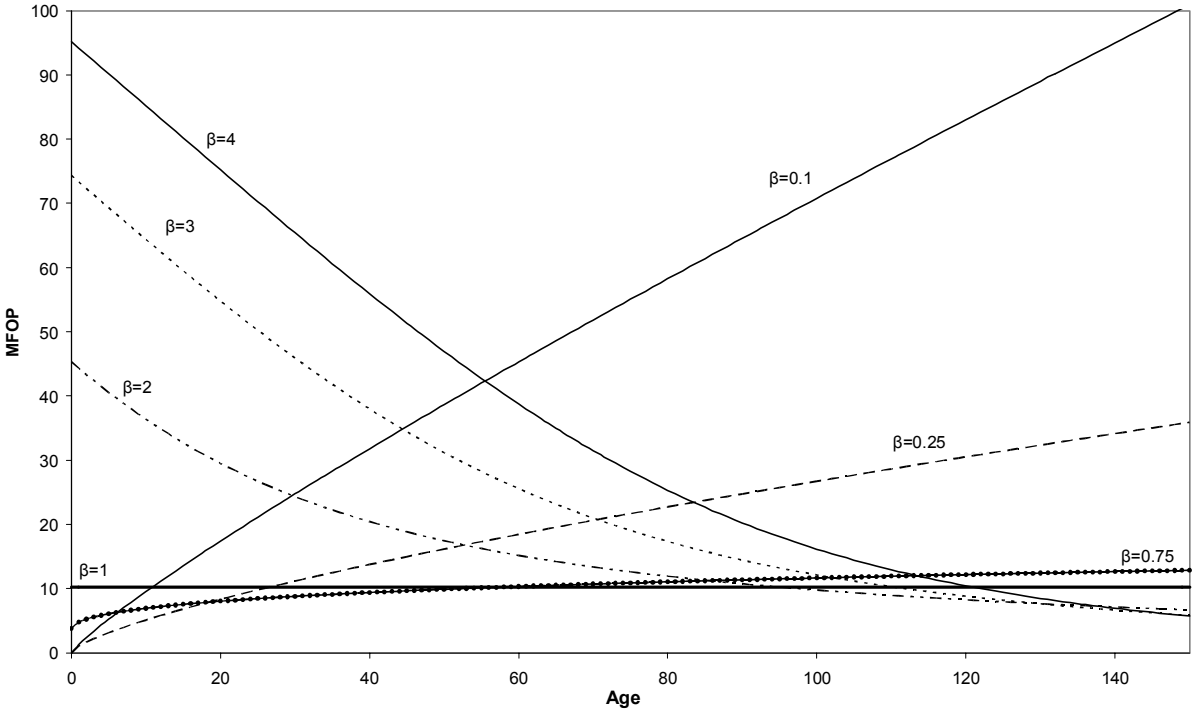


Figure 3.3 - Graph Showing MFOP for Varying Values of β (η = 200, S = 0.95)

- If $\beta < 1$, MFOP curve is convex starting from zero – MFOP increases with age.
- If $\beta = 1$, MFOP is constant throughout life.
- If $\beta > 1$, MFOP curve is concave tending towards zero – MFOP decreases with age.

Note that if the MFOP is considerably shorter than the characteristic life, survivability does not decrease much over consecutive MFOPs for age-wearing components. Also note that the units for MFOP (u), characteristic life (η) and age (t) are undefined here but will be the same as each other. Common units are hours, days or years.

The assumptions and limitations of this model are rather restrictive:

- All components use the same units of aging.
- All Weibull distributions are two-parameter, not three.
- This model cannot consider redundancy as the components must all be in series.

- The normal and lognormal distributions have not been considered.

Todinov [27] uses the broader measure Minimum Failure-Free Operating Period (MFFOP). This is not directly applicable to the MFOP concept mentioned here as it is based on the homogeneous Poisson process, and only exponential failure times are considered. Also, being based on FFOPs, no consideration is made of redundancy.

3.6.2 Ultra Reliable Aircraft Model (URAM)

The mathematical models that have been produced are limited in their scope. An alternative approach to evaluate other aspects of MFOP, including more realistic, complex systems, is Discrete Event Simulation (DES). This allows many factors of a realistic MFOP scenario to be considered.

URAM [23, 26] is a user-configurable tool that not only assesses the MFOP capability of an item at all levels (platform down to component), but also considers other factors such as operating scenarios, maintenance regimes, logistics support, etc. It can consider most enablers of MFOP, except for reconfigurability, which is still in development. It outputs a variety of reports, most importantly the probability of completing an MFOP of a certain length. The program allows designers to investigate possible options and provides a theoretical method of experimentation.

URAM considers various factors relating to maintenance [23]. These include forward planning, such as prediction of future failures, scheduled replacements and changing mission plans based on the system state. Diagnosing faults takes account of visibility of system failures and varying maintenance regimes. It also includes the ability to group components and systems in LRIs. Other factors, such as balancing the importance of controlling costs, minimising repair time and achieving maximum mission effectiveness are considered. Manpower, spares and other resources can be scheduled according to need.

URAM uses “mode maps” to model the relationships between components having several failure modes, the potential for systems to have various functions within a given mission, and the potential effects of system failures. These maps can also allow failures to impact on other systems and components. Consideration is given to the impact of environmental stresses and how it can vary with location. Different components can age in different ways, and the environment in which a platform is operating can speed up or slow down this aging.

URAM is a very large, complex simulation program and is capable of handling many different aspects of the proposed MFOP idea. However, there are still some areas where improvement is possible. For instance, the requirement to build the Reliability Block Diagrams within the program, as is currently the case, is very time-consuming for large systems, and can lead to over-simplification of the failure modes of the platform. Similarly, the capability of the software to cover the many different aspects of MFOP operation is reflected in the complexity and time consumption of setting up simulations. Even a simple system set-up can take a long time to implement in URAM.

URAM has only a limited ability to analyse systems which perform phased missions (see Section 4.2 for more explanation of this subject). While it is possible to perform a basic analysis of an MFOP system performing phased missions, it cannot consider more complex real-world factors such as abandonment of missions, choice of next phase or choosing between a primary mission objective (such as destruction of a target) or performing a less important, secondary mission objective (e.g. reconnaissance of target area) due to time, fuel or system failure constraints. In addition, the simulation results produced for phased missions are not tailored to offer the data required for in-depth analysis of relevant factors such as phase and mission failure probability, criticality of component failures to phase and mission failures, and speed of convergence of the phase unreliabilities.

As outlined above, the modelling of the specific scenario of MFOP systems performing phased missions needs extending. This is an area addressed in this thesis.

3.7 Summary

An MFOP is a period where the platform can be guaranteed for a given (high) percentage of the time to survive without loss of functionality from failures. A Maintenance Recovery Period follows each MFOP. This puts the platform back in a state to complete the next MFOP. The concept of MFOPs has been proposed to more effectively fulfil the requirements of the current and future aeronautical industries by bringing together many technologies for a common purpose. MFOPs are thought to offer considerable advantages in operational availability and capability, saving the operators time and money. Considerable effort is required to overcome problems involved with the complexities of implementation.

It is expected that a platform with successful MFOPs will contain the following features:

- Good inherent reliability – although balanced with cost, it is essential that the overall reliability of components and systems is improved. Also important is an understanding of the modes of failures.
- Prognostics – the ability to predict when a system is going to fail, to an accurate timeframe, would be critical in an MFOP platform and would enable early replacement to prevent any onboard failures.
- Diagnostics – understanding the precise location and cause of a component failure would make the provision of maintenance faster and more efficient, saving costs.
- Redundancy/Reconfigurability – backup systems and the ability to recreate the functionality of a failed system would act as a cover for less reliable equipment and further enhances the confidence in the platform. This has to be managed by a capable diagnostics system.

The current efforts to produce values for MFOP have been restricted to mathematical models and simulation software. MFOPs are a very complex idea and so the mathematical models produced thus far have not been able to adequately consider the effects of combining the various enablers. Much more work is needed

before these can be used for accurate evaluation of a platform's MFOP. The only published MFOP simulation software is the URAM project, which aims to simulate all aspects of a highly reliable aircraft. This includes maintenance, operations, missions planning and so on. It is able to consider many more factors than the analytical mathematical models. However, there are limitations in URAM's ability to simulate MFOP systems performing phased missions. For example, as lack of necessary figures (e.g. phase and mission failure probabilities and speed of convergence of these probabilities), and inability to model real-world phased missions scenarios, such as choosing between several options for the next phase, and reconfiguration of the mission profile to suit time, fuel or system failure constraints. Hence, there is a need for further research into the combination of MFOP with phased missions.

Chapter 4 Phased Missions and Petri Nets

4.1 Introduction

A conceivable and common scenario in real world applications of reliability engineering is one where a system performs missions, in order to complete one or more objectives. Often, these missions can be broken up into distinct time intervals, each of which has at least one goal which must be completed. These time intervals are known as *phases*, and due to the necessity of completing different phase objectives, the system may need to operate different sub-systems at different times. The failure logic of the system is therefore subject to change as time progresses. This chapter gives details on how these types of systems have been analysed.

Also investigated in this chapter is the development of a modelling method known as Petri nets. These are a very versatile way of combining the flexibility of state-space models with the logical function of combinatorial models. Their development from first concept through to variations in modern-day use is reviewed. Their application in the field of reliability, particularly with respect to phased missions, is also explained.

4.2 Phased Missions

4.2.1 Non-Repairable Systems

In 1975 Esary and Ziehms published a paper [6] outlining the first consideration of methods of analysing phased missions. A mission is split up into sequential phases, each of which has a specific task to perform. Various combinations of the non-repairable basic events may cause phase failure, and these combinations may differ from phase to phase. If all phases are completed successfully, the mission is a success. The probability of this is known as *Mission Reliability*, R_{MISS}.

Mission reliability cannot be calculated exactly by simply multiplying the individual phase reliabilities; this will lead to an overestimate of R_{MISS}. This is because the twin assumptions that all components are in the working state at the beginning of each phase, and that no components are shared between the phases (and are hence independent) are both false.

The consideration of non-repairable components means that any component will work continuously until its failure, thence remaining in the failed state. Esary and Ziehms' method transforms a multi-phased mission into a single-phase mission (assuming zero-duration phase boundaries), following which existing techniques can be used to calculate mission reliability.

4.2.1.1 Transformation of a Multi-Phased Mission to an Equivalent Single-Phased Mission

A non-repairable system undergoing a multi-phased mission will require its components to function through all previous phases in order to be operational in the current phase. As such, a single component c operating in phase j may be represented by j components in series. Each "sub-component", c_1, c_2,...,c_j, corresponds to the behaviour of component c in each phase up to j. Figure 4.1 shows this transformation.

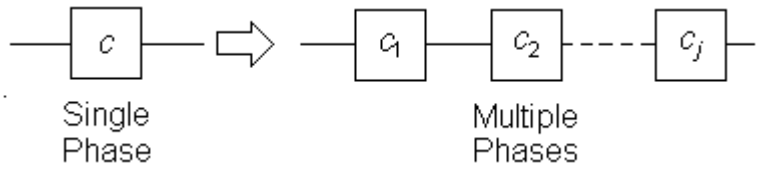


Figure 4.1 – Single and Multiple Phase Component Block Diagrams

Similarly, when using Fault Tree Analysis to display the failure of component c, this can be replaced by an OR combination of the failure of component c in any phase up to and including phase j.

Following the component transformation, the phase configurations can be joined in series. Figure 4.2 shows this for a simple three phase and three component mission.

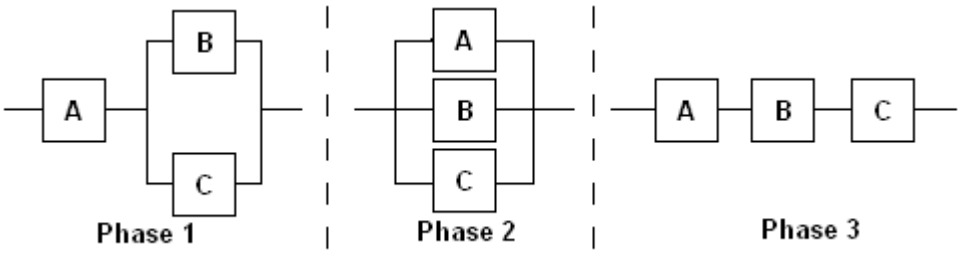


Figure 4.2 – System Component Configuration in Three Phases

In Figure 4.2, components A, B and C combine in various ways to form the system in each of the three phases, thus also changing the failure logic in each. The transformation of this as described yields the equivalent single-phase mission profile in Figure 4.3.

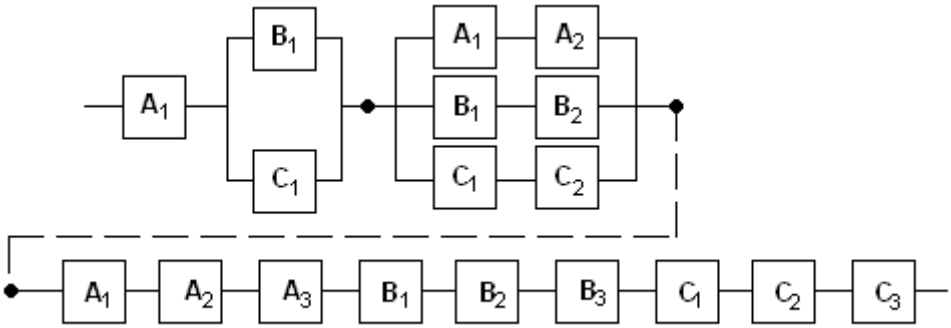


Figure 4.3 – Equivalent Single Phase Mission to System in Figure 4.2

The single-phase network is created from each of the transformed original systems (with expanded sub-components as in Figure 4.1) placed in series in the order of the phases from the start of the mission to the end. Due to sharing common components, the sub-systems are not independent, and so sub-system reliabilities cannot be multiplied to produce the mission reliability. These sub-system reliabilities, R_i, are:

$$
\begin{aligned}
R_1 &= \rho_{A_1}(\rho_{B_1} + \rho_{C_1} - \rho_{B_1}\rho_{C_1}) \\
R_2 &= \rho_{A_1}\rho_{A_2} + \rho_{B_1}\rho_{B_2} + \rho_{C_2}\rho_{C_2} - \rho_{A_1}\rho_{A_2}\rho_{B_1}\rho_{B_2} - \rho_{A_1}\rho_{A_2}\rho_{C_1}\rho_{C_2} - \rho_{B_1}\rho_{B_2}\rho_{C_1}\rho_{C_2} \\
&\quad + \rho_{A_1}\rho_{A_2}\rho_{B_1}\rho_{B_2}\rho_{C_1}\rho_{C_2} \\
R_3 &= \rho_{A_1}\rho_{A_2}\rho_{A_3}\rho_{B_1}\rho_{B_2}\rho_{B_3}\rho_{C_1}\rho_{C_2}\rho_{C_3}
\end{aligned}
\tag{4.1}
$$

where $\rho_{c_1} = P[x_c(t_1) = 0]$ then, $\rho_{c_i} = P[x_c(t_i) = 0 \mid x_c(t_{i-1}) = 0]$ for $i = 2,...,m$

$$x_c = \begin{cases} 0 \text{ if component } c \text{ is working} \\ 1 \quad \text{if component } c \text{ is failed} \end{cases}$$

The true mission reliability, $R_{MISS} = \rho_{A_1}\rho_{A_2}\rho_{A_3}\rho_{B_1}\rho_{B_2}\rho_{B_3}\rho_{C_1}\rho_{C_2}\rho_{C_3}$, is found by finding the simplest form, discussed in the next section, of Figure 4.3, as shown in Figure 4.4. This reliability value is greater than the product of the subsystem reliabilities.

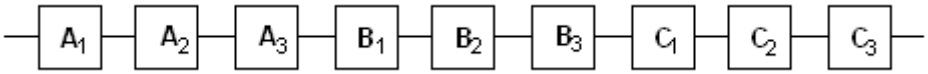


Figure 4.4 – Simplest form of Figure 4.3

4.2.1.1.1 Cut Set Cancellation

It is possible to simplify the multiple phase configurations before transformation to a single-phase structure, by using the technique of cut set cancellation. If a minimal cut set in a phase contains any minimal cut sets of a later phase, they may be removed from the earlier phase. This is because the failure logic for later phases automatically includes mention of the failure of components in all phases up to that phase, and so these earlier phase cut sets are superfluous.

Continuing the example in Figure 4.2, the fault trees for the phases are shown in Figure 4.5. The minimal cut sets for each phase are:

Phase 1	Phase 2	Phase 3
{A}	{A,B,C}	{A}
{B,C}		{B}
		{C}

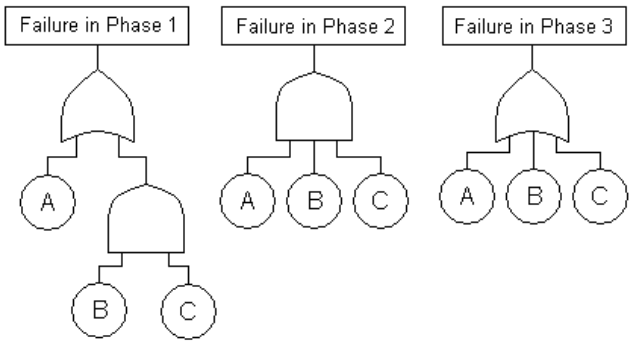


Figure 4.5 – Phase Fault Trees of System in Figure 4.2

Cut set {A} can be removed from phase 1 as A is also a cut set of phase 3, and A's failure in phase 1 will also cause failure in phase 3. Similarly, if component B fails during phase 1, this will cause mission failure during phase 3, making C's state irrelevant. Cut set {B,C} can therefore be removed from phase 1. {A,B,C} can be removed from phase 2 using the same logic, making the phase minimal cut sets:

Phase 1	Phase 2	Phase 3
-	-	{A}
-		{B}
		{C}

In spite of the cancellations of phase cut sets, the systems are equivalent and will result in the same figure for mission reliability. The latter approach is easier to convert to a single-phase mission structure, however.

4.2.1.2 Obtaining Bounds for Mission Unreliability

Mission unreliability is defined as the probability of failure of successful system operation during at least one of the phases of a mission. It is important, however, that this figure or bounds for it are calculated as quickly or efficiently as possible. Burdick *et al* [28] have reviewed the method of Esary and Ziehms and present methods to approximate mission unreliability for non-repairable systems with statistically independent components. The methods are applicable to systems with large numbers of components, where calculation of an exact mission reliability figure would be costly.

Esary and Ziehms' method of transforming each basic event c into a series of events $c_1,\ldots,c_i$ for each phase i creates a large increase in the number of cut sets for the mission and thus also in the time and cost of an exact calculation of mission unreliability. The methods of Burdick *et al* to estimate mission unreliability avoid the need for basic event transformation, of which four of the most accurate and conservative are presented here.

- **Inclusion-Exclusion Expansion of Phase Unreliabilities**

  Each phase fault tree is used to find that phase's minimal cut sets. These are then used in an inclusion-exclusion expansion (equation 2.3) to find the unreliability of phase i, Q_i, using unconditional basic event unreliabilities. Equation 4.1 shows an expression to find the conditional basic event c reliability ρ_{c_j}. The unconditional basic event c reliability p_{c_j} derives from this, as shown in equation 4.2.

$$p_{c_j} = P\left[x_c\left(t_j\right) = 0\right] = \prod_{i=1}^{j} \rho_{c_i} \quad \text{for } j = 1,\ldots,m$$

4.2

By multiplying these individual phase reliabilities, an approximation to the mission reliability, $\overline{Q}_{IN-EX}$, can be reached (equation 4.3).

$$\overline{Q}_{IN-EX} = \prod_{j=1}^{m} R_j$$

4.3

Instead, by summing the individual phase unreliabilities, an approximation to Q_{IN-EX}, the mission unreliability, can be reached, as shown in equation 4.4.

$$Q_{IN-EX} \leq \sum_{j=1}^{m} Q_j$$

4.4

If the technique is applied after the cut-set cancellation method mentioned in the previous section is implemented, another approximation to the mission unreliability, $Q_{IN-EX(CC)}$, can be found. Due to there being generally fewer cut sets in each phase due to the cancellation, the resulting approximation is usually less than Q_{IN-EX}.

- **Minimal Cut Set Bound**

  In this approximation, after the minimal cut sets are found, Equation 4.5 allows calculation of the probability of failure of cut set C_i in phase j, $q_{C_{i_j}}$.

$$q_{C_{i_j}} = \prod_{c=1}^{N_{C_{i_j}}} P\{c\} \qquad \textbf{4.5}$$

where c is the occurrence of basic event c in cut set C_i of phase j

The reliability of phase j can thus be estimated using the minimal cut set bound in equation 4.6.

$$R_j = \prod_{i=1}^{N_{mcs_j}} p_{C_{i_j}} \qquad \textbf{4.6}$$

This method can be used to find an approximation to the mission reliability, $\overline{Q}_{MCB}$ also, by applying equation 4.3. Once again, the cut set cancellation technique can be applied before the approximation to give another approximation to the mission unreliability, $Q_{MCB(CC)}$.

The four approximations to the mission unreliability can be ordered in terms of their accuracy, as shown in equation 4.7.

$$Q_{MISS} \leq Q_{IN-EX(CC)} \leq \begin{matrix} Q_{MCB(CC)} \\ Q_{IN-EX} \end{matrix} \leq Q_{MCB}$$

4.7

As the approximations do not consider the outcomes of previous phases, the bounds are only estimates. These are useful, for instance, in systems where a large amount of components prevents an exact calculation of the mission reliability due to cost or difficulty.

Veatch [29] presents another technique for estimating the unreliability of a phased mission system. The method constructs a lower bound structure function for application to periodic systems.

4.2.1.3 Expected Number of Failures

Finding the expected number of failures for a multi-phased mission is a more difficult task than finding the same parameter in a single-phase mission. Phase transitions can cause system failure without a component failure occurring at that time. Montague and Fussell [30] describe a method for finding the expected number of failures for a phased mission.

However, their method does not consider the outcome of previous phases. The combination of all of the cut sets and path sets, to find the expected number of phases across the phase boundary, would be very time consuming and inefficient for large systems, and approximation techniques would not be accurate. Due to the method ignoring the outcome of earlier phases, the value of the expected number of failures it gives cannot be accurate.

4.2.1.4 Laws of Boolean Phase Algebra

The performance of a component *c* has been considered by other methods to be a separate event in different phases, while the system reliability parameters are based on the product of the event probabilities. Dazhi and Xiaozhong [31] have established a set of Boolean algebraic laws to combine and simplify expressions of component behaviour.

If phases j and k are ordered such that $1 \leq k \leq j$, the rules of logical AND and OR for events as expressed in equations 4.8, 4.9 and 4.10 can be applied to phased mission systems.

$$\begin{aligned} A_{(j)} &= A_1 \vee A_2 \vee \ldots \vee A_j \\ &= \bigcup_{i=1}^{k} A_i \bigcup_{i=k+1}^{j} A_i \\ &= A_{(k)} \bigcup_{i=k+1}^{j} A_i \end{aligned}$$

4.8

$$\begin{aligned} A_{(k)} \wedge A_{(j)} &= A_{(k)} \wedge \left(A_{(k)} \bigcup_{i=k+1}^{j} A_i \right) \\ &= A_{(k)} \bigcup_{i=k+1}^{j} \left(A_{(k)} A_i \right) \\ &= A_{(k)} \end{aligned}$$

4.9

$$\begin{aligned} A_{(k)} \vee A_{(j)} &= A_{(k)} \vee \left(A_{(k)} \bigcup_{i=k+1}^{j} A_i \right) \\ &= A_{(k)} \bigcup_{i=k+1}^{j} A_i \\ &= A_{(j)} \end{aligned}$$

4.10

The event of a system being failed in phase j, X_j, could have occurred in any phase from the first to j inclusive, as shown in equation 4.11.

$$X_{(j)} = X_1 \vee X_2 \vee \ldots \vee X_j$$

4.11

where $X_i = \bigcup_{i=1}^{N_{mcs_j}} C_{i_{(j)}}$

Mission unreliability can then be found from equation 4.12.

$$
\begin{aligned}
Q_{MISS} &= P\lfloor X_{(m)} \rfloor \\
&= P\left(\bigcup_{j=1}^{m} X_j \right) \\
&= P\left[\bigcup_{j=1}^{m} \left(\bigcup_{i=1}^{N_{msc_j}} C_{i_{(j)}} \right) \right]
\end{aligned}
$$

4.12

The combination of the different phase events in equation 4.12 contains an automatic implementation of the cut set cancellation technique presented in Section 4.2.1.1.1. Kohda *et al* [32] supply further Boolean laws which use the minimal cut sets and path sets for each phase which removes any need to convert the mission into a single phase system. The use of Boolean laws to solve phased mission reliability removes the false assumption made by Esary and Ziehms which is that the performance of the component through different phases is independent.

Somani and Trivedi [33] introduced another method whereby Boolean algebraic methods were employed to find phased mission system reliability without converting all the phases into a single mission. This solution necessitates the transfer of information from one phase to the next, as phases are not independent.

Their method uses cumulative distribution functions (CDFs) with a mass at the origin. A random variable X has a CDF with time t given by equation 4.13.

$$
q_X(t) = (1 - e^{-\lambda T_1}) + e^{-\lambda T_1}(1 - e^{-\lambda t})
$$

4.13

where T_1 is the time at the start of the phase.

The function's mass at the origin is given by $P(X = 0) = (1 - e^{-\lambda T_1})$, which is the probability that the component exists in the failed state at the start of the phase. The other part of the equation, $e^{-\lambda T_1}(1 - e^{-\lambda t})$, represents the continuous part of the function which is the failure probability distribution of the component in the current phase. These distribution functions can be used to represent the failure probabilities of components.

The situation considered by Somani and Trivedi is the simplest, with each phase having the same system configuration and failure criteria. Each phase differs from the rest by having differing component failure rates. Phase- and age-dependent failure rates are considered in addition to random phase durations. Consideration is also given to situations where the system configuration varies between phases. The research considered four plausible scenarios across a phase boundary:

1. Accumulation of component failures is insufficient to cause system failure in either phase j or j+1.
2. Accumulation of component failures causes system failure in both phase j and j+1.
3. Accumulation of component failures does not lead to system failure in phase j but does lead to system failure in phase j+1.
4. Accumulation of component failures leads to system failure in phase j but not in phase j+1.

Cases 1 and 2 do not require a change in failure combinations between phases, while the combinations in case 3 can be thought of as failure in both phases, as the failure will occur at the transition point. The mission reliability for these three cases is found in the same way as for a mission where all phase configurations are identical, by solving the fault tree for the final phase.

A method is given to solve the fourth situation which accounts for the probability of occurrence of failure combinations in phase j. System unreliability can be divided into two parts – common failure combinations and phase failure combinations.

- Common Failure Combinations – considers the component failure combinations that are common to all phases. If phase j+1 can fail through a particular combination, that combination can also fail phase j. The unreliability due to these combinations are found using the same method mentioned above, that is, finding the failure distribution for each component and solving for the last phase fault tree.

- Phase Failure Combinations – considers the component failure combinations which can cause failure in phase j but in no subsequent phases. Phase failure combinations for phase j (PFC_j) that are treated as success combinations for all the subsequent phases are given by equation 4.14.

$$PFC_j = (\ldots((E_j \wedge \overline{E_{j+1}}) \wedge \overline{E_{j+2}})\ldots \wedge \overline{E_m})$$

4.14

This is simplified in equation 4.15.

$$PFC_j = E_j \wedge (\overline{E_{j+1}} \vee \ldots \vee \overline{E_m})$$

4.15

The phase failure combinations for phase j use the same notation as for the Esary and Ziehms' method where a separate symbol (a_j) is used to denote the occurrence of an event in each phase j. A new notation is defined, where A_j represents the failure of component A in any phase up to and including phase j,

$$A_j = a_1 \vee a_2 \vee \ldots \vee a_j$$

and $\overline{A_j}$ represents the success of component A from the start of the mission to the end of phase j,

$$\overline{A_j} = \overline{a_1} \wedge \overline{a_2} \wedge \cdots \wedge \overline{a_j}$$

As the expressions for phase failure combinations represent both failure and success events, simplification will merge combinations of both terms. Algebraic rules are introduced to simplify these combinations. If i and j are two phases in a mission where $i < j$, the Boolean laws can be summarised as in equations 4.16.

1.	$\overline{A_i} \wedge \overline{A_j} \rightarrow \overline{A_j}$	5.	$A_i \vee A_j \rightarrow A_j$
2.	$A_i \wedge A_j \rightarrow A_i$	6.	$\overline{A_i} \vee A_j \rightarrow 1$
3.	$A_i \wedge \overline{A_j} \rightarrow 0$	7.	$A_i \vee \overline{A_j} \rightarrow$ no physical meaning
4.	$\overline{A_i} \vee \overline{A_j} \rightarrow \overline{A_i}$		

4.16

The laws presented in equations 4.16 contain an error. The sixth law represents the situation where component *A* succeeds up to and including phase *i* OR component A fails in any phase up to and including phase *j*. This is combined to become 1, which is incorrect as for this to be true, the former term would need to be the complement of the latter, which is not the case. The correct expression would be $\overline{A_i} \vee A_i \rightarrow 1$, which is not the same as rule 6 in equations 4.16. Inaccurate results will result if these simplification techniques are used uncorrected.

The method becomes even more complicated when terms are combined which cannot be simplified, such as $\overline{A_i} \wedge A_j$. This represents the event where component *A* is operational until the end of phase *i*, and then fails between the end of phase *i* and the end of phase *j*. This is not the same as A_j, which is the event that component *A* fails in any phase up to and including phase *j*. The probability of this combination is given in equation 4.17.

$$\begin{aligned} P\left(\overline{A_i} \wedge A_j = 1\right) &= E\left[\overline{A_i} \wedge A_j\right] = E\left[\overline{A_i} \wedge \left(1 - \overline{A_j}\right)\right] \\ &= E\left[\overline{A_i}\right] - E\left[\overline{A_i} \wedge \overline{A_j}\right] = P\left(\overline{A_i} = 1\right) - P\left(\overline{A_j} = 1\right) \end{aligned}$$

4.17

It can be seen from equation 4.17 that the probability of event $\overline{A_i} \wedge A_j$ is the same as the probability that component fails between the end of phase *i* and the end of phase *j*. It would be of use to find the value of this term without requiring the use of the component success probabilities.

System unreliability is found by evaluating the PFCs for all phases and is given in equation 4.18.

$$Q_{MISS} = P(E_m) + \sum_{j=1}^{m-1} P(PFC_j)$$

4.18

The unreliability at the end of each phase *j* can be expressed in equation 4.19.

$$Q_j = \sum_{i=1}^{j} P(PFC_i)$$

4.19

At the time of phase change, the unreliability figure may jump, due to more stringent failure criteria in a later phase. This is referred to as a latent failure.

Somani and Trivedi's method successfully identifies the possible situations that can occur across a phase boundary. The Boolean laws defined in their work allow for added simplification of component failure and success events in a phased mission system, despite the noted error in the laws. All previous methods consider that a phase failure would result in mission failure, thus preventing the continuation of the mission after the phase failure event. Phase *j* failure could not, then, combine with phase *j*+1 success. Somani and Trivedi's method allows the phases of a mission to occur in any order, allowing this situation to occur. The calculation of the system performance parameters involves the combination of the current phase failure combinations with the success combinations for all subsequent phases. This leads to lengthy calculations for situations where there are numerous phases or cut sets in each phase.

Ma and Trivedi have extended this work by using a computational algorithm to find the mission unreliability in the form of the sum of disjoint products [34].

4.2.1.5 Binary Decision Diagrams

BDDs have previously been used to represent single-phase systems, using the method shown in Section 2.3. For adaptation to the multi-phased scenario, the dependence of a component's failure state on its performance in all previous phases must be considered. This complicates the BDD technique.

Zang *et al* [35] present a method to apply the BDD technique to multi-phase missions. Component behaviour in the current and all previous phases is represented by using a series of sub-components, as described in Section 4.2.1.1.

The failure function for component c in phase j, $q_{c_j}(t)$, is the probability that component c fails in phase j given that it worked at the start of phase j, and is expressed in equation 4.20.

Since all c_i, $i = 1, 2, \ldots, j$ are in series,

$$q_{c_j}(t) = \left[1 - \prod_{i=1}^{j-1}\left[1 - q_{c_i}(T_i)\right]\right] + \left[\prod_{i=1}^{j-1}\left[1 - q_{c_i}(T_i)\right]\right] \cdot q_{c_j}(t)$$

4.20

The first term of equation 4.20 represents the probability that the component has already failed during the previous 1, …, j–1 phases. The second term represents the failure probability distribution of the component in phase j.

As for a single-phase mission, a BDD requires that the basic events be ordered in a sequence before it can be constructed. Zang *et al* present two possible ordering schemes, expanding each component c into its series of sub-components in the following ways:

- Forwards Phase-Dependent Operation (PDO): The variables are ordered in the same pattern as the phase order, $c_1, c_2, \ldots, c_m$.
- Backwards PDO: The variables are ordered in the reverse of the phase order, $c_m, c_{m-1}, \ldots, c_1$.

The **ite** structure of the performance of component c in two phases i and j can be represented by E_i and E_j respectively,

$$E_i = \mathbf{ite}(c_i, G_1, G_2)$$
$$E_j = \mathbf{ite}(c_j, H_1, H_2)$$

Logic operations between E_i and E_j can be represented by BDD manipulations as:

$$\text{Forwards PDO : } \mathbf{ite}(c_i, G_1, G_2) \oplus \mathbf{ite}(c_j, H_1, H_2) = \mathbf{ite}(c_i, G_1 \oplus H_1, G_2 \oplus E_j)$$
$$\text{Backwards PDO : } \mathbf{ite}(c_i, G_1, G_2) \oplus \mathbf{ite}(c_j, H_1, H_2) = \mathbf{ite}(c_i, E_i \oplus H_1, G_2 \oplus H_2)$$

The order of events before BDD construction has a great effect on the resultant size of the diagram. Methods may be used for the selection of the most appropriate or efficient ordering sequence of variables in the BDD. Once the order has been selected, each component is replaced by a series of sub-components in either of the orders mentioned above. Generally speaking, backwards PDO results in a smaller BDD and common component cancellation is performed without any additional calculations.

An algorithm is presented for construction of a BDD for a phased-mission system:

1. Obtain the failure function for each variable using equation 4.20.
2. Order the mission components using an heuristic method.
3. Generate the BDD for each phase using logic equations.
4. Use phase algebra and the backwards PDO to combine each phase BDD using OR logic to obtain a mission BDD.
5. Calculate the unreliability of the PMS from the mission BDD.

In a backwards PDO BDD, the '0' branches (non-occurrence of the basic events) always link two variables that belong to different components. The '1' branches (basic event occurrence) can connect either variables of the same component, or those of different components.

Considering a BDD for function G,

$$\mathrm{G} = \mathbf{ite}(c_j, G_1, G_2) = c_j \cdot G_1 + \overline{c_j} \cdot G_2$$

Since the '0' branch always links events of different components, G_2 cannot represent any event of component c. It follows that c_j and G_2 are always statistically independent events, and so,

$$\mathrm{P}(\overline{c_j} \cdot G_2 = 1) = \mathrm{P}(c_j = 1) \cdot \mathrm{P}(G_2 = 1)$$

If a '1' branch links nodes from different components, G_1 also cannot represent any event of c, and the same method can be applied as for a single-phase system shown in equation 4.21.

$$\begin{aligned}
\mathrm{P}(G = 1) &= \mathrm{E}[\mathrm{G}] = \mathrm{E}[c_j \cdot G_1 + \overline{c_j} \cdot G_2] \\
&= \mathrm{E}[c_j] \cdot \mathrm{E}[G_1] + \mathrm{E}[\overline{c_j}] \cdot \mathrm{E}[G_2] \\
&= \mathrm{E}[G_1] + (1 - \mathrm{E}[c_j]) \cdot (\mathrm{E}[G_2] - \mathrm{E}[G_1]) \\
&= \mathrm{P}(G_1 = 1) + (1 - \mathrm{P}(c_j = 1)) \cdot (\mathrm{P}(G_2 = 1) - \mathrm{P}(G_1 = 1))
\end{aligned}$$

4.21

For a '1' branch linking nodes belonging to the same component, G_1 will be dependent on a variable of c. Due to lack of independence between the two node events, the following structures apply:

$$\mathrm{G} = \mathbf{ite}(c_j, G_1, G_2) = c_j \cdot G_1 + \overline{c_j} \cdot G_2$$

$$G_1 = \mathbf{ite}(c_i, H_1, H_2) = c_i \cdot H_1 + \overline{c_i} \cdot H_2$$

Also,

$$\begin{aligned}
\mathrm{P}(\mathrm{G} = 1) &= \mathrm{E}[\mathrm{G}] = \mathrm{E}[c_j \cdot G_1 + \overline{c_j} \cdot G_2] \\
&= \mathrm{E}[c_j \cdot (c_i \cdot H_1 + \overline{c_i} \cdot H_2) + \mathrm{E}[\overline{c_j}] \cdot \mathrm{E}[G_2] \\
&= \mathrm{E}[c_j \cdot c_i \cdot H_1 + c_j \cdot \overline{c_i} \cdot H_2) + \mathrm{E}[\overline{c_j}] \cdot \mathrm{E}[G_2]
\end{aligned}$$

Using the rules of phase algebra in equations 4.16, a branch linking two nodes belonging to the same component is given in equation 4.22.

$$\begin{aligned}\mathrm{P}(\mathrm{G}=1) &= \mathrm{E}[c_j \cdot c_i \cdot H_1 + c_j \cdot \overline{c_i} \cdot H_2) + \mathrm{E}[\overline{c_j}] \cdot \mathrm{E}[G_2] \\ &= \mathrm{E}[G_1] + \mathrm{E}[\overline{c_j}] \cdot (\mathrm{E}[G_2] - \mathrm{E}[H_2]) \\ &= \mathrm{P}(G_1 = 1) + (1 - \mathrm{P}(c_j = 1)) \cdot (\mathrm{P}(G_2 = 1) - \mathrm{P}(H_2 = 1))\end{aligned} \tag{4.22}$$

Depending on whether the '1' branch links events of different or the same components, equations 4.21 or 4.22 would apply respectively.

As failures are possible at phase transition time, the unreliability may jump at this instant. Phase BDDs can be used to find the system unreliability just before and just after this time so that the size of the 'jump' may be calculated.

Xing and Dugan [36, 37] noted the limitations of the approach given by Zang *et al*. The developed PDO will only generate the correct phased mission BDD if the following rules are adhered to:

1. Orderings implemented in the generation of each phase BDD must be consistent or the same for all phases.
2. Variables belonging to the same component in different phases must stay together in the ordering scheme. This is achieved by expanding each component into sub-component form after the ordering of components has been defined using heuristics.

If any random ordering scheme is used, the PDO is not complete enough to combine the single phase BDDs into an equivalent mission BDD. However, a BDD with backwards PDO may represent an impossible scenario, such as the success of an event in a later phase ordered before the failure of that event in an earlier phase. An example of this is shown in Figure 4.6.

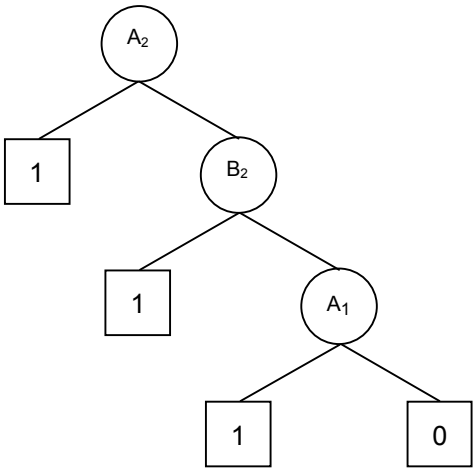


Figure 4.6 – BDD Using Backwards PDO Pattern

If A is working in phase 2, it cannot have failed in phase 1. If impossibilities such as these are shown by the BDD, they can be removed. The incoming branch to each impossible node is instead passed to the node on its right '0' branch, as for any component to work in a later phase, it must have worked through earlier phases. Nodes below the left son are removed. Thus Figure 4.6 becomes as shown in Figure 4.7.

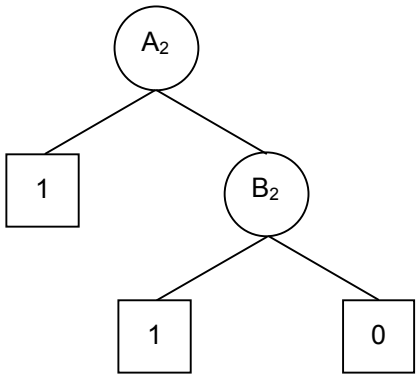


Figure 4.7 – BDD as in Figure 4.6 but with Impossible Nodes Removed

Similarly, with the forwards PDO, another method must be applied to remove impossible nodes. Once all impossible node combinations are removed, any ordering scheme may be used to find the final phased mission BDD.

The method presented by Zang *et al* reveals a very simple, efficient way of representing the failure logic of a phased mission, despite the limitations identified by Xing and Dugan. However, each phase *j* BDD is only constructed from the

failure logic for phase *j*, which discounts the potential outcomes of previous phases and so the phase BDDs will be incorrect.

Dunnett and Andrews [38] outline a method which allows the formation of a full mission BDD from each of the phase BDDs. The phase BDDs are found by converting phase fault trees to BDD form. If a component operates successfully through a phase, it is possible for it to then go on to fail in a subsequent phase. The construction of the mission BDD therefore considers each phase in turn, with all BDD paths which end in a terminal '0' vertex being connected to the top node for the next phase BDD. Each path which ends in system failure is marked to indicate the first phase which would be failed by that combination of component failures. The initial overall mission BDD is thus very large, but rules are provided for its reduction, by removing paths which represent impossible component conditions such as failing more than once or working in a phase following a failure.

The BDD method has also been applied [39-41] to scenarios where several different platforms, some of which may be unmanned, execute individual missions which contribute towards the completion of an overall objective. The completion of the mission objective may require the platforms to perform different phases simultaneously, or to finish or commence certain tasks at the same time. In order to allow the completion of these goals, accurate and fast information is needed on the state of components at the current point in time, both before and during a mission (diagnostics), and the likelihood of mission success based on this information (prognostics). The research uses BDDs as a way of quickly finding reliability parameters in order to make judgements on whether the overall mission should be abandoned, amended or substituted for a alternate objective.

4.2.1.6 Non-Coherent Fault Trees

The methods provided so far allow calculation of mission reliability parameters, but do not allow the system failure probability for each phase to be found. La Band and Andrews [42] established a method of finding these values using a fault-tree approach. Their approach continues the fault tree convention of the event of a component failing in phase *i* being that where the component could have failed in any phase from 1 to *i*. System failure in phase *i* is represented as the AND

combination of the system successfully completing phases 1 to i-1 and failing in phase i. This can be shown in fault tree form as in Figure 4.8.

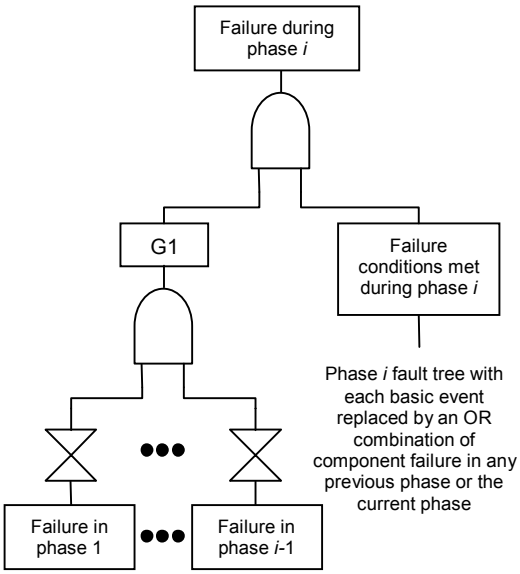


Figure 4.8 – Generalised Phase Failure Fault Tree

If the phase failure probabilities, Q_i, i = 1,...,n are found, the mission unreliability, Q_{MISS}, is found by their sum:

$$Q_{MISS} = \sum_{i=1}^{n} Q_i \qquad \textbf{4.23}$$

For every phase other than the initial one, its fault tree as shown in Figure 4.8 will be non-coherent, due to the inclusion of a NOT gate. This gate is used in representing the ways in which the previous phases will have survived. The combinations of basic events which lead to top event occurrence are referred to not as cut sets but as *prime implicants*. In order to find qualitatively the ways in which the system could fail in phase i, the top event failure must be expressed in terms of the prime implicants. Section 4.2.1.4 established notation for the success or failure of a component, which is extended by La Band and Andrews to include the event where a component A fails at some point from the start of phase i to the end of phase j, A_{ij}. The probability of this event can be found by integrating the component's failure time probability density function, $f_A(t)$, as in equation 4.24.

$$q_{A_{ij}} = q_A(t_{i-1}, t_j) = \int_{t_{i-1}}^{t_j} f_A(t)dt$$

4.24

This notation allows the algebraic laws in equations 4.16 to be updated and extended as in equations 4.25, where $i < j$.

1. $A_i \wedge A_i = A_i$
2. $A_i \wedge A_j = 0$
3. $A_i \wedge A_{ij} = A_i$
4. $\overline{A_i} \wedge A_i = 0$
5. $\overline{A_i} \wedge A_{ij} = A_{i+1,j}$
6. $\overline{A_i} \wedge \overline{A_{i+1}} \wedge \ldots \wedge \overline{A_j} = \overline{A_{ij}}$
7. $A_i \vee A_{i+1} \vee \ldots \vee A_j = A_{ij}$

4.25

The prime implicants occurrence probabilities are combined in an inclusion-exclusion expansion as in equation 2.3. This method allows phase reliabilities to be found in terms of the prime implicants which cause phase failure. This is useful for the analysis of phase risk, where consequences of phase failure must be taken into account (consider the difference in consequence between a plane failing while grounded or while in the air). However, although ways of reducing the resulting fault trees are also provided, the inclusion of previous phase success in each phase fault tree can very quickly lead to a large fault tree for each subsequent phase or extra component. If this is then used with an inclusion-exclusion expansion, the calculation for phase unreliability may require a great deal of computer power. Because of this, the authors present methods for converting the phase fault trees to binary decision diagrams. Each phase fault tree is converted to a BDD, using the standard **ite** method explained in section 4.2.1.5. The prime implicants are found from each disjoint path which leads to a '1' node, and these are simplified using the techniques in equations 4.25. This makes the procedure much more efficient.

4.2.1.7 Importance Measures

Importance measures, as previously explained in section 2.2.3.2, are a way of expressing the contribution of a specific component to the system failure. If the failure of a particular component is found to highly contribute to system failure, this allows resources to be concentrated on improving the reliability performance of that component. Andrews [43] extends the concept to find importance measures for components in systems which undergo phased missions.

Birnbaum's measure of importance, also known as the criticality function, considers the probability of the system being in a state such that it is critical for a given component. This function is also applied to phases, with a critical state for component *i* in phase *j* being defined as "a state of the remaining components through the previous and current phases such that the system is working in phase *j* and the failure of component *i* will then cause phase (and mission) failure". The conditions for this to be true are that all phases prior to *j* must have successfully completed, and component *i* must be working at the start of phase *j*.

A tabular approach is used to demonstrate the philosophy of the method, but this is impractical for large systems or missions. A faster method of calculating the phase criticality function, based on the phase failure likelihood function, is given in equation 4.26.

$$G_{i,j} = \frac{\partial Q_j}{\partial q_{i_j}}$$

4.26

While the phase criticality function is useful in qualifying and quantifying the probability of the system being critical for a given component, a drawback of it is that, for each component, the function does not depend on its own failure probability. As such, other importance measures exist which consider the component's failure probability in conjunction with the criticality function.

A system failure occurs when it is in a critical state for a component and that component subsequently fails. If the probability of this is divided by the system

failure probability, the result is the proportion of times the component failure will cause system failure. This is a useful expression of component *i*'s contribution to system failure. There are two ways in which the criticality and subsequent phase failure can occur:

1. The system can be in a critical state for a component *i* in phase *j* and component *i* then fails during the phase causing phase failure (Phase Importance).
2. The failure conditions for phase *j* exist prior to that phase beginning, and phase failure occurs on transition to phase *j* (Transition Importance).

For transition failure to occur, the system must be in a critical state for component *i* in a phase prior to phase *j*, with *i* also failing prior to phase *j*. Andrews then defines two measures of component importance:

- *Phase Component Importance* for component *i* in phase *j* is the probability of the system being critical for component *i* in phase *j*, and the component then fails in that phase (equation 4.27):

$$I_{i,j}^{P} = \frac{G_{i,j} q_{i_j}}{Q_j} = \frac{\frac{\partial Q_j}{\partial q_{i_j}} q_{i_j}}{Q_j}$$

4.27

- *Phase Transition Importance* for component *i* in phase *j* is the probability of the phase failing as it starts, due to the failure of component *i*. Q_j is the probability of failure in phase *j*. It accounts for the probability of either failing during the phase or on transition into the phase. Thus, to find the probability of failure on phase transition, the logic equations for this must remove any of the failure events which occur in phase *j*. Thus, the causes of phase *j* failure must occur in phases 1 to *j*-1, without any of the earlier phases having failed.

The phase criticality function for phase j transition failure due to component i in a phase k prior to j, $G^T_{i,j,k}$ is shown in equation 4.28.

$$G^T_{i,j,k} = \frac{\partial Q^T_j}{\partial q_{i_k}}$$

4.28

The phase transition importance measure is the proportion of system failures which are caused by entry to that phase, as shown in equation 4.29.

$$I^T_{i,j} = \frac{\left(\sum_{k=1}^{j-1} G^T_{i,j,k} q_{i_k}\right)}{Q_j} = \frac{\left(\sum_{k=1}^{j-1} \frac{\partial Q^T_j}{\partial q_{i_k}} q_{i_k}\right)}{Q_j}$$

4.29

The total importance contribution of component i to failure in phase j is the sum of the phase and transition importance measures, as shown in equation 4.30.

$$I_{i,j} = I^P_{i,j} + I^T_{i,j}$$

4.30

A measure of the total contribution of the failure of component i to failure of the whole mission is therefore as shown in equation .

$$I_i = \frac{\sum_{\substack{all \\ j}} \left\{ \frac{\partial Q_j}{\partial q_{i_j}} q_{i_j} + \left(\sum_{k=1}^{j-1} \frac{\partial Q^T_j}{\partial q_{i_k}} q_{i_k} \right) \right\}}{Q_{MISS}}$$

4.31

This method is extended by Andrews in another paper [44] with the use of Ternary Decision Diagrams (TDDs, an extension of BDDs) to allow fast calculation of Birnbaum's phase and transition criticality measures.

4.2.1.8 Markov Methods

It is not necessary to use combinatorial approaches such as Fault Tree methods in order to solve multi-phased missions. Instead, Markov methods may be used either by solving for each phase individually, or by analysing the entire mission in a single model. The former approach requires each phase Markov model to be solved independently, and linked to that of the next phase through a state probability vector. Dugan [45] considered the latter method, presenting a method of constructing a single continuous-time discrete-space Markov model for phased-mission systems, with a state space equal to the size of the union of the components in each phase model. Phase fault trees are used in its construction.

Continuing the example shown in Figure 4.2, a Markov model may be constructed, assuming the following:

- Failure rates for the components are constant for the duration of the phase, but can be different for each phase.
- The system fails due to failure in any phase of the mission.
- Phase change times are deterministic.

If the set of components is not consistent throughout the phases, or if a component cannot fail in a given phase, the Markov model generation will encounter problems, as the system states will not match. The phase fault trees in Figure 4.5 can be converted to Markov chains for further analysis, with system states showing the states of components A, B and C in the form {A B C} with 0 as working and 1 as failed, as shown in Figure 4.9.

The combination of these three Markov models into a single model requires the use of a multiplicative factor, Φ_i, which is appended to each phase i transition. Transitions are thus defined by the sum of corresponding phase transitions, as shown in Figure 4.10.

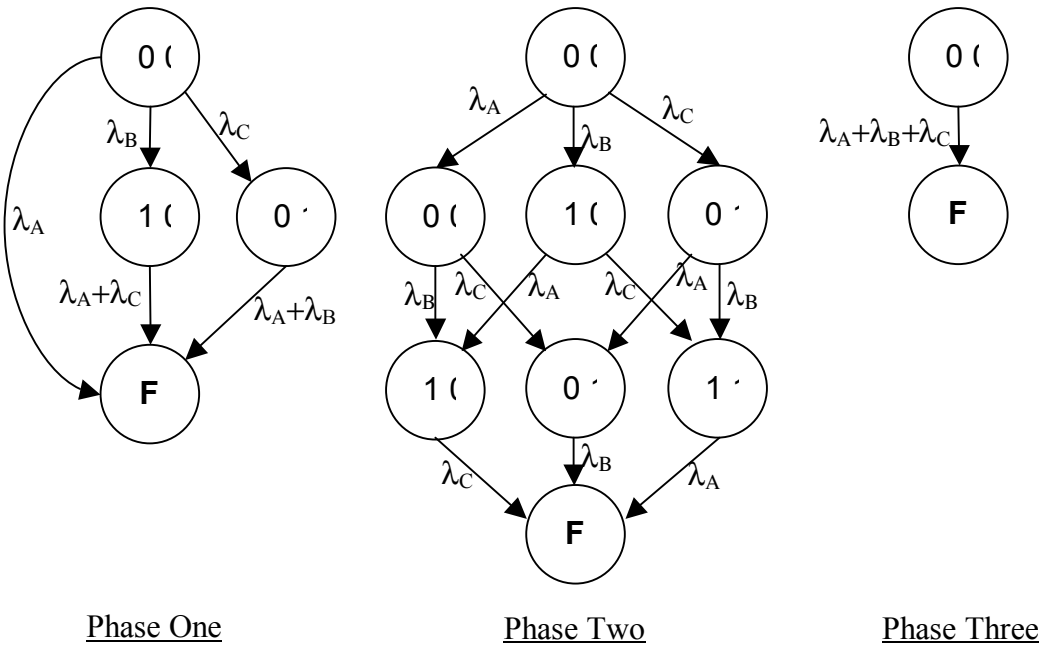


Figure 4.9 – Phase Markov Models for Figure 4.2

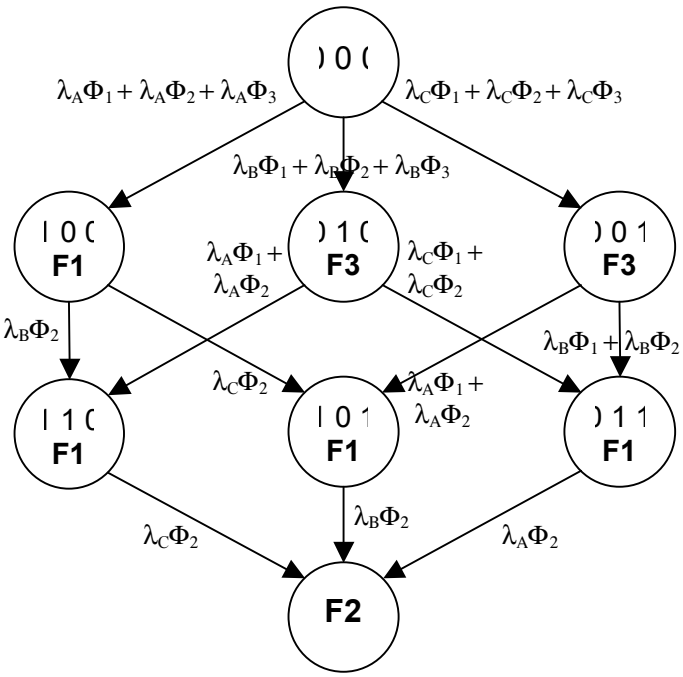


Figure 4.10 – Combined Mission Markov Model

Standard numerical techniques can be employed to solve the combined model in Figure 4.10. For solutions to phase i ($t_{i-1} \leq t \leq t_i$), $\Phi_i = 1$, while $\Phi_j = 0$, $i \neq j$. This removes any transition not belonging in the current phase. The state space does not change – only the state transitions vary from phase to phase.

If the components used differ from phase to phase, the model can still be formed by creating a full Markov state listing. This is an expansion of all components which have some contribution at any point of the mission. It is possible for a system state in a particular phase to be a failed state, and for this not to be true in the earlier phases. If, however, a failure state is reached in any given phase, it becomes absorbing for all future phases. The system states can be defined as "operational for all phases" or "failed in phase *i*", where *i* is the first phase in which the system fails.

The combined Markov model gives a set of differential equations, for which a numerical solution method such as Runge-Kutta must be used [45]. The initial conditions for the first phase are known, while the failure probability for each phase is found by using the Markov state probabilities at the end of the previous phase. The final phase state probability vector is passed directly to the following phase for further analysis.

The state space of the combined model is defined by the total number of components in use at any point of the mission. Due to this, a multi-phased system using a high number of components will cause a very large Markov model to be created, and will increase the number of differential equations that require solving. This potential inefficiency for large systems is a drawback to the Markov method.

4.2.2 Repairable Systems

All of the methods reviewed thus far have been applicable only to non-repairable systems. While repairs are typically unfeasible on a spacecraft or an in-flight aircraft, an analysis where repairs are frequently made would be useful. There are, however, situations where intra-mission repairs are possible, and so it is necessary to understand the associated reliabilities of these systems. The following sections review the published methods of analysing repairable multi-phased missions.

4.2.2.1 Combinatorial Approaches

Somani extended his earlier work with Trivedi [33] by applying a combinatorial method to the situation of repairable multi-phased missions [46]. His approach

considers the probability that a component is failed or working at the start of a phase, and the probability that it has changed state (by failing or being repaired during that phase). By evaluating these, it is possible to calculate the probability that a component is in either state after time *t*. However, his method is restricted by the condition that only components which are not active in the phase being analysed may be repaired.

Vaurio [47] put forward a method of approximating the unavailability and failure intensity for each phase of the mission. These values for each component *c* at the start of the mission are obtained by using Laplace transforms, and are then used to calculate phase unavailability and failure intensity, as well as the expected number of mission failures. However, his method does not model the dependencies that arise in repairable systems, while the values for phase unavailability and failure intensity are only approximations of the exact values. Also, the phase calculations do not consider the outcome of previous phases.

Combinatorial methods, therefore, have not so far been able to allow a satisfactory solution to finding mission reliability for a repairable multi-phased system. Other approaches must be considered instead.

4.2.2.2 Markov Methods

As Markov methods allow a consideration of the repeated change of state that a component's repairability produces, they are more tuned to the solution of repairable multi-phased systems than combinatorial methods. Much research has been undertaken in this area using Markov approaches.

4.2.2.2.1 Homogeneous Markov Model

For a Markov model to be homogeneous, the state transitions must not be dependent on time, being governed instead by a constant rate. Other assumptions are:

- The system comprises elements which may be working or failed with independently, exponentially distributed failure and repair times.
- Repair of a component makes it as new, i.e. perfect condition.

- The purpose of a phase is not considered, and so each phase may have more than one. The only criterion for mission failure is failure of the system in any phase.
- Phase change is instantaneous.

Clarotti *et al* [48] were among the first to investigate the applications of Markov methods to repairable phased missions.

Continuing the example in Figure 4.2, there are eight possible states of the system, based on the two states of each of the three components, as Table 4.1 shows.

Table 4.1 – System States of Component Combinations

State	A	B	C
S_1	0	0	0
S_2	1	0	0
S_3	0	1	0
S_4	1	1	0
S_5	0	0	1
S_6	1	0	1
S_7	0	1	1
S_8	1	1	1

where 0 = working
1 = failed

Considering each of the three phases,

Phase 1 $(0,t_1)$

The *probability vector* expresses the probability of the system being in any of the eight possible states. The mission begins with the assumption that each component is in the working state, which results in an initial condition probability vector, **P**(0), as shown in equation 4.32.

$$\mathbf{P}(0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.32}$$

Phase 1 success requires A and either of B and C to be working. Therefore, from Table 4.1, the states which will cause phase failure are states S_2, S_4, S_6, S_7 and S_8. The matrix form as defined in equation 2.16, the progression of phase 1 can be represented by the matrix equation 4.33.

$$\begin{bmatrix} \dot{P}_{S_1}(t) \\ \dot{P}_{S_2}(t) \\ \dot{P}_{S_3}(t) \\ \dot{P}_{S_4}(t) \\ \dot{P}_{S_5}(t) \\ \dot{P}_{S_6}(t) \\ \dot{P}_{S_7}(t) \\ \dot{P}_{S_8}(t) \end{bmatrix}^T = \begin{bmatrix} P_{S_1}(t) \\ P_{S_2}(t) \\ P_{S_3}(t) \\ P_{S_4}(t) \\ P_{S_5}(t) \\ P_{S_6}(t) \\ P_{S_7}(t) \\ P_{S_8}(t) \end{bmatrix}^T \begin{bmatrix} -\sum_1 & \lambda_A & \lambda_B & 0 & \lambda_C & 0 & 0 & 0 \\ - & -\sum_2 & 0 & \lambda_B & 0 & \lambda_C & 0 & 0 \\ \upsilon_B & 0 & -\sum_3 & \lambda_A & 0 & 0 & \lambda_C & 0 \\ 0 & - & - & -\sum_4 & 0 & 0 & 0 & \lambda_C \\ \upsilon_C & 0 & 0 & 0 & -\sum_5 & \lambda_A & \lambda_B & 0 \\ 0 & - & 0 & 0 & - & -\sum_6 & 0 & \lambda_B \\ 0 & 0 & - & 0 & - & 0 & -\sum_7 & \lambda_A \\ 0 & 0 & 0 & - & 0 & - & - & -\sum_8 \end{bmatrix}$$

4.33

where 0 represents an impossible state transition

– represents an absorbing state (no transition is possible out of the state)

Σ_i is the sum of the non-diagonal entries in row i.

At phase change time t_1, the system must be in a state which fails neither phase 1 nor phase 2. For success in phase 2, the system must not have all three components failed. Thus, states S_1, S_3 or S_5, as shown in Table 4.1, are required. The probability of the system residing in each of these states at time $t = t_1$ is represented by $P_{S_1}(t_1)$, $P_{S_3}(t_1)$ and $P_{S_5}(t_1)$ respectively. The sum of these probabilities gives the probability of completing phase 1 and successfully entering phase 2, as shown in equation 4.34.

$$R(t_1) = P_{S_1}(t_1) + P_{S_3}(t_1) + P_{S_5}(t_1)$$

4.34

Phase 2 (t_1,t_2)

Phase 2 can only start if the system is in states S_1, S_3 and S_5, and will progress to other states from these. The vector of initial phase 2 system state probabilities has all entries equal to zero except those corresponding to working states for phase 1 and phase 2, as shown in equation 4.35.

$$\mathbf{P}(t_1) = \begin{bmatrix} P_{S_1} & 0 & P_{S_3} & 0 & P_{S_5} & 0 & 0 & 0 \end{bmatrix}$$

4.35

The matrix equations for solution of phase 2 are given in equation 4.36.

$$\begin{bmatrix} \dot{P}_{S_1}(t) \\ \dot{P}_{S_2}(t) \\ \dot{P}_{S_3}(t) \\ \dot{P}_{S_4}(t) \\ \dot{P}_{S_5}(t) \\ \dot{P}_{S_6}(t) \\ \dot{P}_{S_7}(t) \\ \dot{P}_{S_8}(t) \end{bmatrix}^T = \begin{bmatrix} P_{S_1}(t) \\ P_{S_2}(t) \\ P_{S_3}(t) \\ P_{S_4}(t) \\ P_{S_5}(t) \\ P_{S_6}(t) \\ P_{S_7}(t) \\ P_{S_8}(t) \end{bmatrix}^T \begin{bmatrix} -\Sigma_1 & \lambda_A & \lambda_B & 0 & \lambda_C & 0 & 0 & 0 \\ \upsilon_A & -\Sigma_2 & 0 & \lambda_B & 0 & \lambda_C & 0 & 0 \\ \upsilon_B & 0 & -\Sigma_3 & \lambda_A & 0 & 0 & \lambda_C & 0 \\ 0 & \upsilon_B & \upsilon_A & -\Sigma_4 & 0 & 0 & 0 & \lambda_C \\ \upsilon_C & 0 & 0 & 0 & -\Sigma_5 & \lambda_A & \lambda_B & 0 \\ 0 & \upsilon_C & 0 & 0 & \upsilon_A & -\Sigma_6 & 0 & \lambda_B \\ 0 & 0 & \upsilon_C & 0 & \upsilon_B & 0 & -\Sigma_7 & \lambda_A \\ 0 & 0 & 0 & - & 0 & - & - & -\Sigma_8 \end{bmatrix}$$

4.36

After the phase change time, any state other than S_8 is allowable, due to the failure logic in phase 2 (see Figure 4.2), where no components are working. Therefore, as long as any one component is operational, the system will not fail. Once the time for transition from phase 2 to phase 3 is reached, however, the system must occupy a state where it is working in both of the phases. For this to be true, the system must be in state S_1, with all components working, at time $t = t_2$. The probability of this is given in equation 4.37.

$$R(t_2) = P_{S_1}(t_2)$$

4.37

Phase 3 (t_2,t_3)

For the mission to complete successfully, the system must remain in state S_1 for the entire length of phase 3. The initial phase 3 probability vector is given in equation 4.38.

$$\mathbf{P}(t_2) = \begin{bmatrix} P_{S_1}(t) & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.38

The matrix equation for the solution of phase 3 is given in equation 4.39.

$$\begin{bmatrix} \dot{P}_{S_1}(t) \\ \dot{P}_{S_2}(t) \\ \dot{P}_{S_3}(t) \\ \dot{P}_{S_4}(t) \\ \dot{P}_{S_5}(t) \\ \dot{P}_{S_6}(t) \\ \dot{P}_{S_7}(t) \\ \dot{P}_{S_8}(t) \end{bmatrix}^T = \begin{bmatrix} P_{S_1}(t) \\ P_{S_2}(t) \\ P_{S_3}(t) \\ P_{S_4}(t) \\ P_{S_5}(t) \\ P_{S_6}(t) \\ P_{S_7}(t) \\ P_{S_8}(t) \end{bmatrix}^T \begin{bmatrix} -\Sigma_1 & \lambda_A & \lambda_B & 0 & \lambda_C & 0 & 0 & 0 \\ - & -\Sigma_2 & 0 & \lambda_B & 0 & \lambda_C & 0 & 0 \\ - & 0 & -\Sigma_3 & \lambda_A & 0 & 0 & \lambda_C & 0 \\ 0 & - & - & -\Sigma_4 & 0 & 0 & 0 & \lambda_C \\ - & 0 & 0 & 0 & -\Sigma_5 & \lambda_A & \lambda_B & 0 \\ 0 & - & 0 & 0 & - & -\Sigma_6 & 0 & \lambda_B \\ 0 & 0 & - & 0 & - & 0 & -\Sigma_7 & \lambda_A \\ 0 & 0 & 0 & - & 0 & - & - & -\Sigma_8 \end{bmatrix}$$

4.39

The success probability of the mission is the probability that it does not enter any of the absorbing states (states S_2 to S_8) and is in S_1 at the end of the mission ($t = t_3$). This is shown in equation 4.40.

$$R_{MISS} = P_{S_1}(t_3)$$

4.40

The method identifies the need to find the initial conditions for each phase. Gray [49] also identifies this requirement, and uses parallel subgroups with identical components to solve it. The deficiency of the Clarotti *et al* method is that every phase uses the same state space to represent the system. Some missions will have phases where success does not depend on the performance of some components, or where a high number of components is used. This will result in a very large and complex Markov model which will be difficult and expensive to solve. Also, the

Clarotti *et al* method considers phase reliabilities to be found which include a successful transition at the end of the phase. This does not give true phase reliabilities, however, as the successful transition *out* of a phase does not affect its reliability; as long as the system has remained in a successful state for that phase alone, the phase will be successful. Failure upon transition to the next phase should be considered by that phase's reliability.

Random Phase Durations

The methods presented thus far have had an assumption that each phase has a deterministic length which is defined at the start of the mission. However, this is not necessarily realistic, as some missions, such as those of a military aircraft, will have phase lengths prone to variation depending on the circumstances of the flight. Wells and Bryant [50] first considered this, but in application to a single-phase system.

Alam and Al-Saggaf [51] described two approaches to satisfactorily describe the marginal distributions of the mission phase change times (MPCTs) when the phase change times are random variables. The first is a general formula for the joint probability density function of the MPCTs, while the second models the MPCTs as order statistics of a continuous random variable. The solution is found similarly to the method by Clarotti *et al*, using the expected probability values of being in each phase. Kim and Park [52] developed this idea by using the system eigenvalues to solve the differential equations created by the Markov model.

Somani *et al* [53] also consider random phase durations using Markov models. Their method investigates the change in system failure criteria between phase Markov models. The system states in the model for phase i is mapped to that for phase i+1, using computer software (Hybrid Automated Reliability Predictor, or HARP) to perform this if there are problems. Examples of these are a state not having an equivalent in the following phase, or several states sharing an equivalent state in the next phase.

State Dependent Phase Sequences

A possible mission profile is one where each phase has a pre-determined length, but the order of them is subject to the system state at the time of switching. Mura and

Bondavalli [54] present a method which analyses a phased mission system on two levels. The higher level method models the mission structure with regard to the phases, while the lower level models the state of the system in the phases.

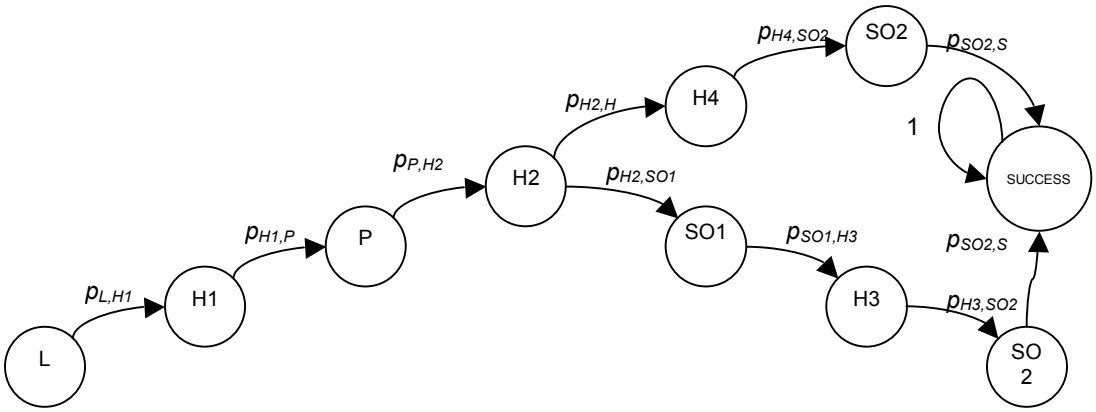
Components within the system can undergo different failure and repair rates in different phases, as long as they remain constant within each phase. This satisfies the requirement for homogeneousness within phases. The phase order can be changed mid-mission, depending on the state of the system at the end of a phase. An example system is presented of a spacecraft mission:

Launch (L):	Launch of spacecraft. Short, stressful phase.
Hibernation (H):	Large periods of dormant behaviour, with minimal activity, for cruise navigation.
Planet (P):	Landing of spacecraft on planet. Short, stressful phase.
Scientific Observations (SO):	Performed in proximity to stellar objects. Main goal of mission.

Figure 4.11 shows the higher-level model of the spacecraft mission. While the failure of any active phase will cause mission failure, this model ignores this and shows merely the possible phase sequences.

Each phase of the mission relies on a given number of computer processors in order to operate properly. If this number of processors is not available, the phase will fail. The numbers are:

Phase		No. Processors Required
Hibernation		1
Launch		3
Planet		3
Scientific Observations	Cruise	3
	Perform Observations	1



where $p_{S1,S2}$ is the probability of executing state *S2* after state *S1*.

Figure 4.11 – Higher-level Model of a Spacecraft Mission

Four processors are provided for the completion of the mission. Phase SO2 is the main objective of the mission, and therefore its performance is essential. SO1 is a secondary goal, to be performed if possible. The choice of whether it is performed or not rests on whether, at the end of H2, any processors are faulty. If this is true, SO1 is skipped and a second successive hibernation phase (H4) is entered. If the system has no faults, SO1 is carried out as normal. The phase carried out after H2 is therefore state dependent. Reward rates can be applied to phases, to indicate the benefits of carrying it out.

The lower level models can be represented by generalised stochastic Petri nets (for an explanation of these, see section 4.3). These Petri nets can be translated into a continuous time Markov model, which indicates the number of working (#W), spare (#S) and failed (#F) processors in the form {#W, #S, #F}. {F} is the absorbing failure state. In the SO phases, all components are required, and so the states are represented by {#W, #F}.

A separate phase transition model is presented which maps the final state probabilities of one phase to the initial state probabilities of the next phase. Where no choice is possible, such as when phase H1 makes a transition to phase P, a deterministic model is applied. The phase transition model for this change is shown in Figure 4.12, where *c* represents the probability of successful reconfiguration of the system ready for the new phase. The Figure shows the mapping only between the non-failure states.

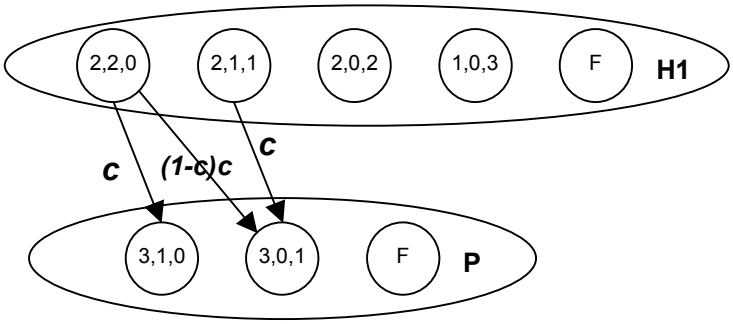


Figure 4.12 – Deterministic Phase Transition Model from H1 to P

Because phase P requires three processors to operate, the successful transition from H1 to P will only be true if the number of working and spare processors is at least equal to this number. The deterministic model shown can be represented by a phase transition matrix, with the number of rows equalling the number of operational states in the exiting phase, and the number of columns being the number of operational states in the entering phase. The elements of the matrix equal the probability of switching from one state in the starting phase to another in the subsequent phase.

If the phase transition is not deterministic, and there are state dependencies between the phases, a probabilistic phase transition model represents the possible system states at the end of the switching phase, as shown in Figure 4.13.

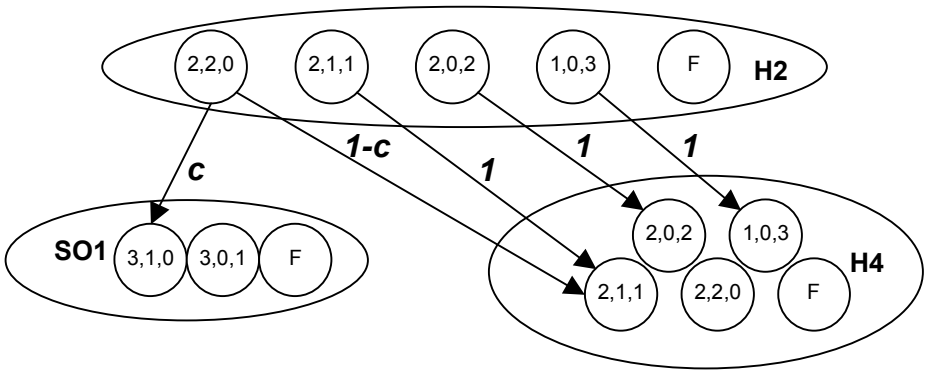


Figure 4.13 – Probabilistic Phase Transition Model from H2 to SO1 or H4

The probabilistic state transition matrix is generated as above, except that the number of columns equals the total number of possible states in all possible entering phases.

The lower level models are solved in the order in which their corresponding phase appears in the higher-level model. The initial phase state probability vector can be found by applying the appropriate transition matrix to the state probabilities at the end of the finishing phase. Solving the higher-level model then allows calculation of reliability parameters. Mura *et al* discuss further methods of solutions to this problem using deterministic and stochastic Petri nets [55].

4.2.2.2.2 Non-Markovian Models

The standard Markov method presented thus far requires each phase to be considered individually, so that a state probability vector at the time of phase change can be calculated. A linear transformation of this vector then creates the initial condition for the subsequent phase. This process works through each of the phases in the mission until the state probability vector for the last phase is found, which represents the mission reliability estimate. A limit of this approach is that the phases are assumed to instantaneously change at specific times, with the condition that the system must be operational in both the exiting and the entering phases. This model cannot represent the amount of work performed or the importance of certain tasks compared to others in a typical mission.

If the transition matrix [**A**] as shown in equation 2.16 contains globally time-dependent coefficients, the model is known as non-Markovian. If these rates are constant, the model is homogeneous. Smotherman and Zemoudeh [56] identified problems with the homogeneous method. Firstly, phase changes and their timing are dependent only upon the current phase, and not the state of the system. A feasible scenario is one where the system has deteriorated, but not failed, such that it takes longer to complete a given phase than a fully functional system would. Another issue is that the number of phases with a random time duration is limited, or requires the computation of order statistic integrals. Also, failure and repair rates must be constant within a phase. This, therefore, ignores the burn-in and wear out stages of a component life, as shown in Figure 1.2.

They present a generalised method of finding system reliability performance parameters using a continuous time finite-state Markov process. Phase change times are remodelled as non-overlapping uniform distributions ordered in the same way as the phases. Failure and repair rates are assumed to be globally time-dependent. Transitions in the model represent phase changes as well as component failures and repairs. Thus, arbitrary distributions of phase change times can be expressed in hazard rate form as time-varying transition rates. The model is solved numerically, using a fifth-order Runge-Kutta method with the non-Markovian matrix set of differential equations. This is extended for time-dependent transition rates, and includes information on each type of phase change: exiting state, entry state(s) and the branching possibility for multiple entry states. If a phase changes at a fixed-time, the probabilities are transferred instantaneously from the exiting state to the entry state(s).

Smotherman and Geist [57] introduce the idea of reward rates which model the effectiveness of the system in particular states. This is applied to a single non-Markovian mission model, and can be used in situations where component failures are not exponentially distributed.

If $\{X(t) \mid t \geq 0\}$ is a finite state stochastic process with the state probabilities $P_{S_i}(t) = P[X(t) = i]$, the set of state differential equations can be expressed by equation 4.41.

$$P_{S_i}(t) = \sum_{j=1}^{N_S} P_{S_j}(t) a_{ij}(t)$$

4.41

The matrix form of this is similar to equation 2.16, with the difference that the transition matrix is time dependent, [**A**(t)]. The possible system states for each phase are given as a subset of the overall set of states in the single model. Phase changes are time-variant and state dependent, with non-instantaneous phase changes being modelled by the inclusion of an intermediate state. Thus, missions with more than one objective and phase change durations greater than zero can be represented.

In the reward model, each system state, S_i has a weight, called a reward rate, $R_{S_i}(t)$, which expresses the "reward" for being in that state for a given length of time. The accumulated reward, $Y(t)$ is calculated instantaneously. The vector of system state reward rates, $\mathbf{R}(t)$, is defined as in equation 4.42.

$$\mathbf{R}(t) = \left[R_{S_1}(t), R_{S_2}(t), \ldots, R_{S_{N_S}}(t)\right]^T$$

4.42

The instantaneous reward rate of the system at time t is given by $\mathbf{P}(t)\cdot\mathbf{R}(t)$. The expected value of $Y(t)$ is given as in equation 4.43.

$$E[Y(t)] = \int_0^t \mathbf{P}(u)\mathbf{R}(u)du = \sum_{i=1}^{N_S} P_{S_i}(u)R_{S_i}(u)du$$

4.43

This type of information may give details such as the expected time in a given subset of states. This allows it to be used to calculate parameters such as expected time on duty or expected time under repair.

A disadvantage of this type of non-Markovian model is the requirement for it to be solved numerically. A standard initial-value solution algorithm may perform this, but the transition matrix must be re-evaluated at each time step, as the transition rates vary with time. If a particular transition rate approaches a discontinuity, the time steps must become smaller and smaller, and so the problem takes much more computing time to solve. Because the model accounts for every state in every phase, the state space is potentially very large, and so the flexibility of the model is counteracted by the increase in computational power required to solve it.

Smotherman and Geist provide a worked and quantified example to explain the method further.

4.2.3 Summary

Various methods for analysing systems which undergo phased missions have been presented. The methods are split into those which consider repairable or non-repairable systems. Whatever the type of system being analysed, the main parameter of interest is that of mission reliability, or the likelihood of mission success. This is equal to the probability that no phases within the mission fail.

The bulk of the literature detailing analysis methods for non-repairable systems has focused on combinatorial methods. The original analysis method was concerned with the formation of a single mission fault tree from those of each of the phases. Basic events were broken down into sub-events representing the failure of the component in each phase. Thus, existing fault tree techniques can be employed to solve the single mission model. This method can yield very large mission fault trees if the number of components or phases increases by only a small amount, and so the computation of the reliability parameters may be expensive. Approximation and reduction techniques have therefore been researched and presented, in order to make the analysis easier to compute. Boolean algebra phase laws can be employed to simplify expressions for phase and mission failure, combining component success and failure throughout multiple phases.

Binary Decision Diagrams have been employed to speed up the process of evaluating the mission's reliability performance. Parameters other than mission reliability, such as importance measures for each component's contribution to the failure of the mission, or the expected number of mission failures in a given time interval, can be calculated. Markov methods have been employed with non-repairable systems to a small degree.

For repairable systems, the bulk of the research has focused on state-space approaches such as Markov methods, with combinatorial approaches typically being unable to account for the dependencies that arise with repairable systems. State-space methods such as Markov consider each phase to have a set of states relating to the operation or failure of each of the components in the system. Homogeneous

Markov methods consider that the transitions between these states are constant with respect to time. Homogeneous Markov methods express the set of transitions from state to state in the form of a matrix. The probabilities of being in any state, some of which will represent phase and mission failure, is calculated. The state probabilities at the end of one phase become the initial conditions for the next phase. By considering each phase in turn, the state probability vector for the final phase gives the mission success probability.

Further developments, such as state-dependent phase sequences, or the consideration that phases may have random durations, have been considered. Non-Markovian methods, which do not require each state change to have a constant rate, have been employed. Their increased flexibility in analysis is countered by the requirement to solve them numerically.

4.3 Petri Nets

4.3.1 Introduction

Petri nets (PNs) were first created in 1962 by Carl Adam Petri [58]. Since that time, many people have researched extensions and adaptations to the original technique, often for application to their specific area of concern. The most important developments, such as the addition of timed elements, and a form of NOT logic, are explained within this chapter.

Since around 1987 the Petri net has been adapted to analyse system reliability. The versatility of PNs allows a PN reliability model to incorporate fault trees, event trees, Markov models and other system representations. Scenarios capable of being modelled by PNs include phased missions, various maintenance policies, MFOPs, fleets performing various missions, and non-exponential component failure time distributions. Prof. Petri himself feels that reliability modelling is an excellent application of PNs [59].

This part of this chapter aims to review the development of Petri nets from their initial conception to the status quo, and their application to the field of reliability analysis.

4.3.2 Formal Definition

This formal definition of a Petri net is taken from Schneeweiss [60]:

A Petri net, $\mathbf{G}_{PN}$, is a bipartite digraph with markings of nodes and edges as shown in equation 4.44:

$$\mathbf{G}_{PN} = \left(\mathbf{V}_p, \mathbf{V}_t, \mathbf{E}; \mathbf{M}(0), \mathbf{D}, \mathbf{W}\right), \quad \mathbf{E} \subseteq (\mathbf{V}_p \times \mathbf{V}_t) \cup (\mathbf{V}_t \times \mathbf{V}_p) \tag{4.44}$$

where $\mathbf{M}(0)$ is the initial marking vector of the set $\mathbf{V}_p$ of places,

The components of **M** and **W** are integers and those of **D** are non-negative real numbers. Thus, a PN is an ordered 6-tuple of two sets of nodes ($\mathbf{V}_p$ and $\mathbf{V}_t$), edges (which are ordered pairs of nodes), vector integers (**M** and **W**) and a random non-negative real vector (**D**).

4.3.3 Basics of Petri nets

Two sources offering explanations of the basic operation of Petri nets are Schneeweiss [60] and Bobbio [61].

4.3.3.1 Original Petri net Concept

The original 1962 Petri net concept is a directed bipartite graph with two distinct types of node: *places*, drawn as circles, and *transitions*, shown as bars. These nodes are connected to one another by directed edges or *arcs*. These arcs are usually considered to be *input arcs* if they go from a place to a transition, and an *output arc* if they are directed the other way, although this convention is not universal.

Places contain *marks* or *tokens*, shown graphically as bullet points, which represent the dynamic aspect of the net: the *marking* (the position of all tokens throughout the entire PN) at any point shows the current state of the system being modelled. This marking changes when any transition *switches*. This switching can only take place when the transition is *enabled.* This process is shown in Figure 4.14.

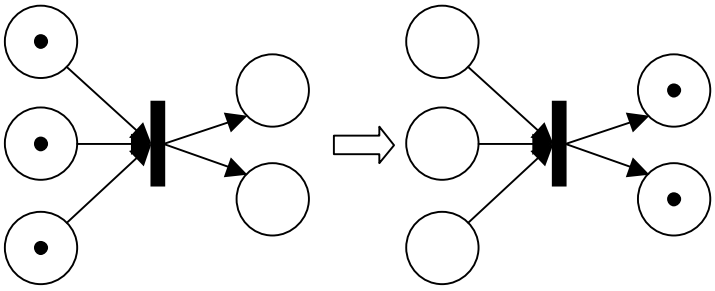


Figure 4.14 – Enabled Transition Firing

As Figure 4.14 shows, a transition (the bar in the middle of the net) is enabled when each of the places (circles) which input to the transition contain one or more tokens. More specifically, there must be a token available for each arc having an input to the transition. Once the transition is enabled, it fires, taking one token from each input place and depositing one token in each place with an arc leading away from the transition. The marking of the net has thus been altered: the system state represented by the first marking has now changed to the one shown by the second.

4.3.3.2 Timed Petri nets

The limited applicability of the original Petri net model, where each transition fired the instant it was enabled, led to the incorporation of temporal modelling. Various proposals were consequently put forward to model transient system behaviour [62]:

- In Ramchandani's doctoral thesis [63], it was proposed that each transition have a delay associated with it. This became the accepted practice.
- Merlin [64] and Berthomieu & Menasche [65] suggested using two values, minimum and maximum times, to define a range of delays for each transition.
- Sifakis [66] suggested associating the delays with places. Coolahan and Roussopoulos [67] used a similar approach. This does not increase the

modelling power over Ramchandani's approach, but does retain the immediate firing feature of the 'standard' PN model.

- Razouk [68] proposed using firing and enabling times – in his model, tokens are "absorbed" by the transition after a given enabling time has passed, and tokens are not output until the firing time has elapsed.

The time associated with delaying transitions can be either deterministic or stochastic. If a PN is required to model processes that have a random (or pseudo-random) nature to them, and this randomness follows a certain pattern such as a statistical distribution, the transitions can sample their switching times from this distribution. S*tochastic Petri nets* (SPN) were created [69, 70], where the firing times for each transition follows the negative exponential distribution. Ajmone-Marsan *et al* [71] put forward *Generalised Stochastic Petri nets* (GSPN) which also allow transitions to have zero firing time (they fire as soon as they are enabled), thereby reincorporating the original 'standard' Petri net into the more developed technique. Investigations into applying other distributions to the transitions' switching times are fewer in number than those using only exponential times [72].

Graphically, while an immediate transition is shown as a single, thick line as Figure 4.14 shows, a timed transition is typically shown as a hollow rectangle – see Figure 4.15.

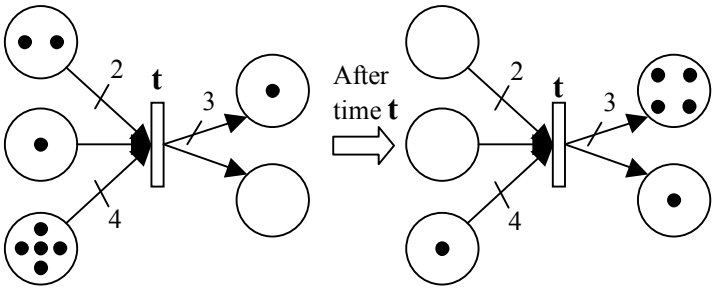


Figure 4.15 – Timed Transitions and Arc Multiplicities

In Figure 4.15, *t* is the time delay associated with the transition. The Figure shows the transition to be enabled, but only after *t* units of time have passed will the transition fire.

It is conceivable that situations may arise which cause a conflict. Examples of these include:

- If two or more transitions are enabled from a common place, then the one with the earlier time delay will switch first. If the input place's token is taken, the other transition will be disabled.
- If two transitions sharing a common input place are immediate or share the same switching time, both will try to take the place's token at the same time. If this occurs, there are several options to resolve the situation:
    - Transitions are given a certain level of preference as to which takes it (say, transition A has a 60% chance of taking it, while B has a 40% chance)
    - The preference could be based on the number of tokens in the input place(s) for the transitions
    - One transition automatically overrides the other.
- If a transition is disabled while waiting for its switching time to pass, there is often a choice as to how this time is affected upon re-enabling [73]:
    - The time can be renewed from the previous enabling
    - A completely new time could be generated
    - The old time could be regenerated

Different transitions may use different resolutions to these problems within the same Petri net.

4.3.3.3 Arc Multiplicities

A widely accepted extension to the standard PN is having more than one arc between a place and a transition. Each place has to have a token for each arc before the transition would be enabled [62]. It is more normal now for the arcs to be contracted into one, with the single arc crossed and a number written next to it which indicates the *multiplicity* or *weight* of the arc; that is, either how many tokens are needed on that arc before a transition will be enabled; or how many tokens will be put in an output place [74]. Note that if no multiplicity is explicitly shown, it is always assumed to be 1.

Figure 4.15 shows the effect that arc multiplicities have on the transfer of tokens. In the Figure initially, the three places which input to the transition have, from the top down, two, one and five tokens respectively, while the places to which the transition outputs have one and no tokens. Because each of the input places has a number of tokens greater than or equal to the multiplicity of its arc to the transition (these are 2, 1 and 4 respectively), the transition is enabled. After a time delay of *t*, the transition switches, removing the weight-number of tokens from each of the input places and putting the weight-number of tokens (three and one) in the output place. This leaves the net with a marking of (0 0 1) in the input places and (4 1) in the output places.

4.3.3.4 Inhibitor Arcs

Like arc multiplicities, inhibitor arcs have become an integral part of the Petri net technique. Inhibitor arcs go only from places to transitions, and act such that if the place inhibiting the transition contains a token, the transition cannot switch, regardless of whether it is otherwise enabled. These arcs can also be associated with multiplicities, so that the place must have a weight-number of tokens before it will inhibit the transition [73].

The inhibitor arc is shown in Figure 4.16 by a normal arc with a circle on the end rather than an arrow. Any multiplicity is shown in the same way as normal arcs. In the initial net, the transition's input places have enough tokens to enable the transition, but the inhibiting place is marked, and thus disables the transition. Hence, after time *t* has passed, the transition does not switch.

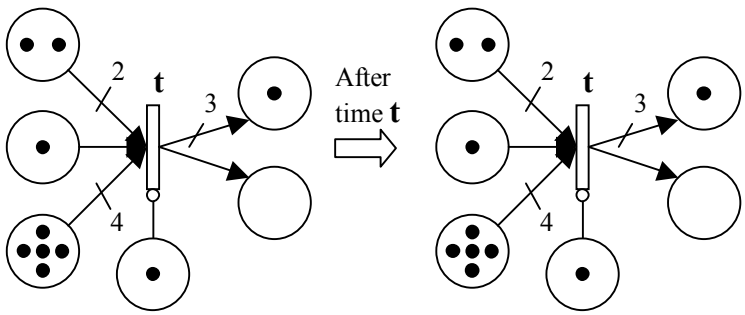


Figure 4.16 – Inhibited Transition

Inhibitor edges are useful in resolving possible clashes between transitions (see Section 4.3.3.2), allowing one to be disabled while the other takes the token. They are also useful for ensuring that, if an immediate transition has a particular place which is both an input and an output, it does not repeatedly fire and place an infinite number of tokens in any other output places.

4.3.3.5 Reachability Graphs

A common way of expressing all the possible markings of a Petri net from a given initial marking is the reachability graph (RG). This is a state graph in which each node corresponds uniquely to a given marking. For example, consider the Petri net in Figure 4.17. The Figure shows a PN with three numbered places and three numbered transitions. Associated with transitions 1 and 2 are time delays D_1 and D_2. Transition 3 is immediate.

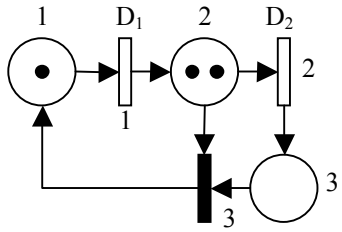


Figure 4.17 – Example Cyclic PN [59]

When one considers the initial marking (1 2 0) as shown in Figure 4.17, there are two possible ways for the marking to change:

- If $D_1 < D_2$, transition 1 will switch first. Place 2 will then have three tokens, and the others none.
- If $D_2 < D_1$, then transition 2 will switch before the others, and all places will have one token. Transition three immediately switches, placing a token in place 1, emptying places 2 and 3, giving a marking of (2 0 0).

The full RG of the PN in Figure 4.17 is shown in Figure 4.18.

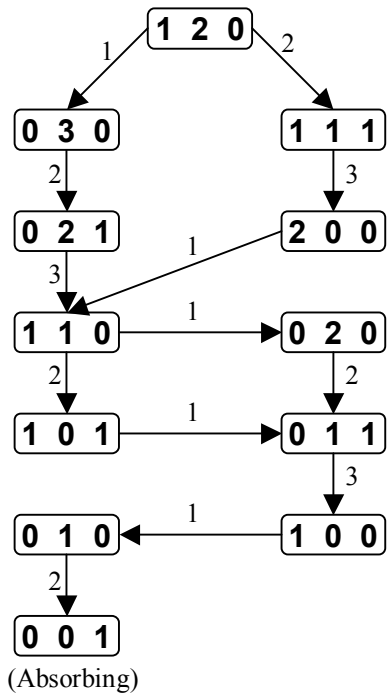


Figure 4.18 – Reachability Graph of Example PN

Schneeweiss [60] gives a mathematical notation of RGs. The marking in a given RG node, as shown in Figure 4.18, is displayed in the form:

$$\mathbf{M}(t) = (M_1(t), M_2(t), ..., M_{|\mathbf{V}_p|}(t))$$

4.45

In a RG, the movement from state to state is linked by an arc which shows the transition that switches to make the state change. The RG in Figure 4.18, no matter which transitions switch first, always ends in the *absorbing state* (0 0 1), with places 1 and 2 empty and place 3 containing 1 token. An absorbing state is one where no more transitions can switch, and the marking cannot change.

As a RG describes all the possible markings that a Petri net can reach from a given initial marking – if a marking is not on this RG, then it is described as *unreachable* from the initial state. Probabilities or sampling times attached to the switching of each transition can therefore govern the probabilities of the PN being in a given marking at any time. This is useful, for example, in giving the probability of a place marked "System Up"/"System Down" being marked. Reachability graphs of a Petri net are closely related to Markov models (see section 4.3.4.1).

4.3.3.6 Coloured and Aging Petri nets

Coloured Petri nets (CPNs) have had some application in the field of reliability, by Volovoi [73, 75] and Mura & Bondavalli [76]. Created by Kurt Jensen [77], each token can affect the firing of a particular transition in a different way, depending on its colour. The colour is merely a graphical way of distinguishing between different tokens, by giving them a label.

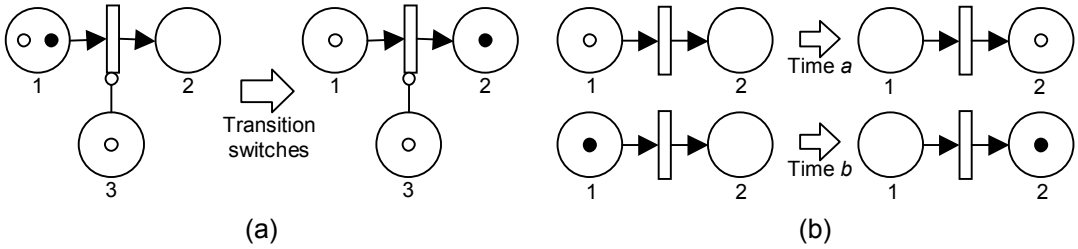


Figure 4.19(a) – Inhibiting Coloured Tokens; (b) – Varying switching times for coloured tokens

Figure 4.19 shows two examples of CPNs. Figure 4.19(a) shows place 1 as input to a transition which outputs to place 2, while place 3 inhibits this transition. If the tokens were indistinguishable, the transition would never be able to fire, due to the token in place 3. In this CPN, a token can only inhibit the switching of a token which is the same colour. In Figure 4.19(a), once the transition's enabling time has passed, the transition will switch the dark token, but not the white token, as a white token is inhibiting the transition. Figure 4.19(b) shows the effects of two different coloured tokens on the time to switch a common transition. The white token will switch it after time *a*, while the black token is switched after time *b*. These switching times can be sampled from dissimilar time distributions.

In a standard Petri net, tokens in places which input to transitions are destroyed when the transition switches, while tokens are "created" to be put in the output places: the two events are considered to be independent of one another. These actions can instead be considered as a single action of moving a token from the input to the output. For standard PNs the difference is indistinguishable, while the second interpretation is core to the notion of CPNs [73]. It is possible for the label or colour of a token to change once it has passed through a transition. All rules pertaining to the effects of specific tokens on transitions, and vice versa, are decided

for the specific modelling situation, and may differ for transitions within the same net.

Volovoi introduced *aging tokens* in the papers [73, 75] as an extension to CPNs, allowing tokens to have 'counters' as labels. These counters can change continuously throughout the time the token enables a transition, rather than once the transition switches, as in standard CPNs. For instance, the value of a counter might be between 0 and 1, to indicate the proportion of its life that has expired. The primary application of this in reliability engineering is in modelling damage to components or systems, such as in warm spares, or the effects of different environments on a particular equipment.

4.3.3.7 Other Variations

Other variations in Petri net design include:

- *Bi-directional transitions* have one associated distribution of times to switch when enabled in one direction, and another distribution when enabled in the other. These transitions connect to places using edges with no arrowheads. This is an attempt to minimise the size of the overall model, and offers no extra modelling power [78].
- *Stochastic Reward Nets* – These attempt to minimise the often confusing nature of large Petri nets. By establishing Boolean logical statements between certain places, transitions which offer only a logical function may be removed. Although the authors claim this offers more clarity and more power than Petri nets, they have not been widely used [74, 79].

4.3.4 Application of Petri nets to Reliability

Petri nets have been applied and adapted to modelling different situations, including workflow management [80] and concurrent computing [81]. One of the original investigations of using Petri nets for reliability prediction was with assessing their ability with respect to the modelling of computer software system reliability [62]. A large number of papers using the Petri net for reliability analysis consider Markov

Regenerative Stochastic Petri nets. This involves producing a reachability graph, which is converted back to a Markov model, allowing an analytical solution to the problem [76, 82]. This section aims to review the ways in which Petri nets have been applied in the field of reliability engineering.

4.3.4.1 Markov Analysis

The concept of Markov analysis was explained briefly in Section 2.4. It is possible to represent a Markov model using a Petri net, as shown by Figure 4.20. The system states are shown as places, while a Markov arc is replaced by two arcs and a transition, using the same rate or distribution of times. The Markov PN will only ever have one token in it, in the place representing the current system state. Upon state change, one of the transitions switches and moves the token to a different place.

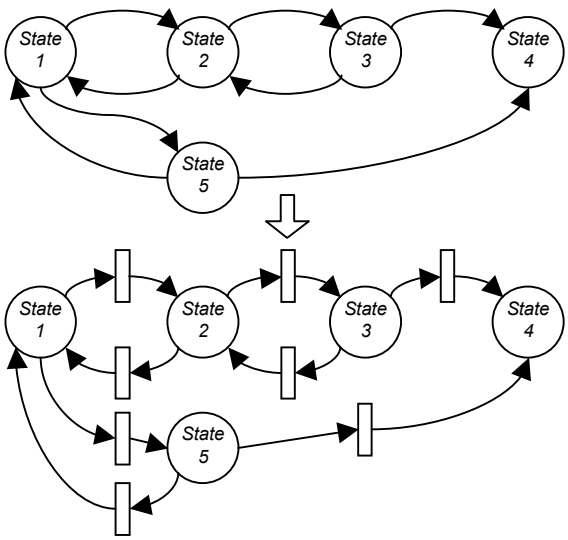


Figure 4.20 - A PN representation of a Markov model

The relationship between PNs and Markov is not restricted to a conversion from one model to the other. As explained in Section 4.3.3.1, the markings of a PN represent the current system state. Continuing the example in Figure 4.17, the initial marking of (1 2 0) represents the operation of the system in a particular way. If transition 1 switches, the marking changes to (0 3 0), which represents the operation of the system in a different way. The RG in Figure 4.18 shows all the possible system states that can be reached from the initial marking, and the transitions that must switch to reach those states. If the time delays of transitions 1 and 2, D_1 and D_2, are

randomly sampled from the negative exponential distribution, then the RG is equivalent to a Markov model, and Markov state equations can be created from which the required system reliability characteristics can be evaluated [2].

The Markov state equations for a Markov model are given by equation 2.16. Considering as an example the RG given in Figure 4.18, firstly it is necessary to combine states which are only reached instantaneously into the following state. As shown in Figure 4.17, transition 3 is an immediate transition, and so any markings enabling this need not be considered. Applying this, there are nine states, including the initial marking (1 2 0), that are reachable for a fixed time. These are given in Table 4.2.

Table 4.2 – State numbers of markings of PN in Figure 4.17

State No.	Marking	State No.	Marking	State No.	Marking
1	1 2 0	4	1 1 0	7	1 0 0
2	0 3 0	5	0 2 0	8	0 1 0
3	2 0 0	6	1 0 1	9	0 0 1

Applying the state markings in Table 4.2 to equation 2.16 gives the set of simultaneous system equations in equations 4.46, where λ_1 and λ_2 are the rates of occurrence of transitions 1 and 2 switching, respectively, gives equations 4.46.

$$\begin{aligned}
\dot{Q}_1(t) &= -(\lambda_1+\lambda_2)Q_1(t) & \dot{Q}_6(t) &= \lambda_2 Q_4(t) - \lambda_1 Q_6(t) \\
\dot{Q}_2(t) &= \lambda_1 Q_1(t) - \lambda_2 Q_2(t) & \dot{Q}_7(t) &= \lambda_2 Q_5(t) + \lambda_1 Q_6(t) - \lambda_1 Q_7(t) \\
\dot{Q}_3(t) &= \lambda_2 Q_1(t) - \lambda_1 Q_3(t) & \dot{Q}_8(t) &= \lambda_1 Q_7(t) - \lambda_2 Q_8(t) \\
\dot{Q}_4(t) &= \lambda_2 Q_2(t) + \lambda_1 Q_3(t) - (\lambda_1+\lambda_2)Q_4(t) & \dot{Q}_9(t) &= -\lambda_2 Q_8(t) \\
\dot{Q}_5(t) &= \lambda_1 Q_4(t) - \lambda_2 Q_5(t)
\end{aligned} \qquad \textbf{4.46}$$

When the equations 4.46 are combined with equation 4.47,

$$\sum_{i=1}^{9} Q_i(t) = 1, \qquad \textbf{4.47}$$

results can be reached for the values of $Q_i(t)$ with respect to λ_1 and λ_2, by using a Laplace Transform to convert the differential equations into a set of solvable algebraic equations.

A criticism that has been levelled at PNs [61] is that they can become very large when modelling complex systems. While this is true, the number of different states within a large PN is also very large, and hence a representation of this in a RG or a Markov model would be enormous. Referring specifically to Stochastic Petri nets (SPN), Volovoi mentions a "commonly shared belief about SPNs being a mere preprocessor to Markov chains" and "associated assumptions about the limitations of SPNs" [73]. By stating this he is criticising the approach of converting PNs into Markov chains, thus limiting the modelling possibilities of the PN technique. Volovoi mentions alternatives to this such as discrete event simulation or "direct numerical solutions of SPNs balance equations", citing Horton [83] for further reading on this latter technique.

4.3.4.2 Fault Tree Modelling

Fault trees have been introduced in section 1.3 and explained further in section 2.2. While Fault Trees are useful for breaking down the occurrence of a top event into basic events, they require the basic events to be independent of one another in order to find an exact analytical solution to the top event probability. Other common reliability scenarios such as component repair or bringing redundant systems online, which may then affect their rate of failure, cannot be accounted for in a standard fault tree. It is useful then to convert a fault tree into PN form and adapt it to include these other real-life factors.

Trivedi and Malhotra [74] outline the simplest way of converting a fault tree to a PN, and provide an algorithm for this purpose. Starting from the top event, each non-basic event is represented by a place, and the logic gates are formed by immediate transitions linking the places representing events further down the tree.

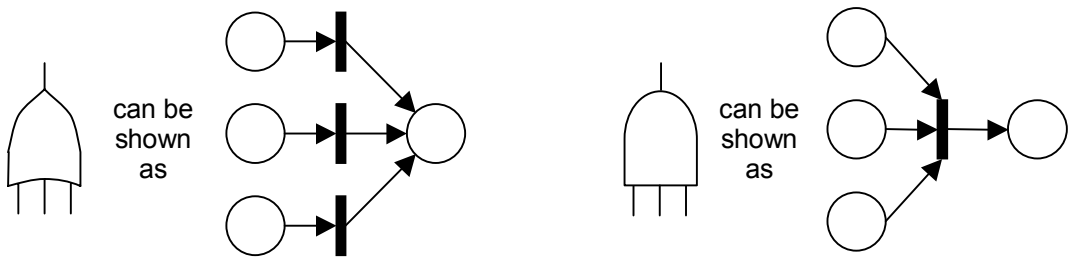


Figure 4.21 – OR and AND Gate PN Representation

An OR gate produces its output event if any number of the input events occur. In PN form, as shown in Figure 4.21, each input to the OR gate is a place, connected to an immediate transition, and each transition inputs to the single output place. If any gate-input place receives a token, that token will move to the output place. An AND gate produces its output event only when all of the input events occur. As the Figure shows, the PN equivalent is a series of input places all providing an input to the same immediate transition, as transition switching inherently includes an AND type logic rule. This transition then outputs to the higher-level event place. Only once all the input places are marked will the transition switch and deposit a token into the output place.

A problem with Trivedi and Malhotra's PN representation of OR gates is that if more than one input event occurs, more than one token will be put in the output place. Although this is a minor problem, this could cause effects in other parts of the net. It may then be necessary, once an input event has occurred, to prevent the other gate transitions from switching.

Malhotra and Trivedi also present the PN version of basic events. These are very simple, containing only two places and two transitions each, as shown in Figure 4.22. The two places represent the states "Component Up", where the relevant basic event is in a working state, and "Component Down", where it has failed. They are linked by a timed transition, whose time to switch may be sampled from a distribution of failure times which matches that of the relevant component. Once this time has passed, the basic event will fail, the transition will switch and the token will move into the "Component Down" place. Added to Malhotra and Trivedi's PN

representation of a basic event is a repair transition, which allows a failed component to return to the working state after a given time period.

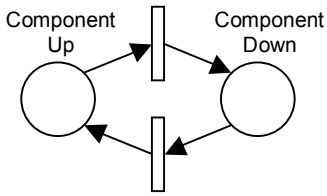


Figure 4.22 – Repairable Basic Event in PN Representation

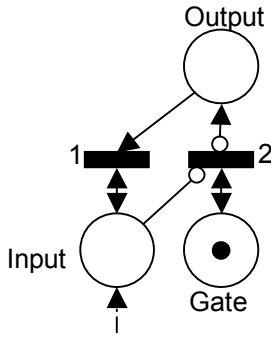


Figure 4.23 – NOT Gate PN Representation

A PN for the NOT gate is shown in Figure 4.23. This is adapted from Liu and Chiou [84]. When no token is present in the input place, the gate place enables transition 2, which switches and places tokens in both the output place and back in the gate place. The output place then inhibits transition 2 to prevent an infinite number of tokens being output to it (this was omitted in Liu and Chiou's paper). If a token appears in the input place and the output place is marked, transition 1 (also omitted from Liu and Chiou's paper) is enabled and switches, removing the token from the output place and placing a token back in the input place. Transition 2 thus becomes inhibited.

As logic gates in PN representation use only immediate transitions, then, as with Boolean algebra, Petri nets can be cut down in size [60]. For instance, in the situation that an OR gate has several AND gates as inputs, the PN may be constructed as in Figure 4.24(a). If there is a single place followed by a single transition, then these are superfluous elements and may be removed, as shown in Figure 4.24(b). Liu and Chiou [84] call this *absorption*.

Liu and Chiou [84] developed a method to get minimal cut sets from PNs which is identical to MOCUS in Fault Tree structures.

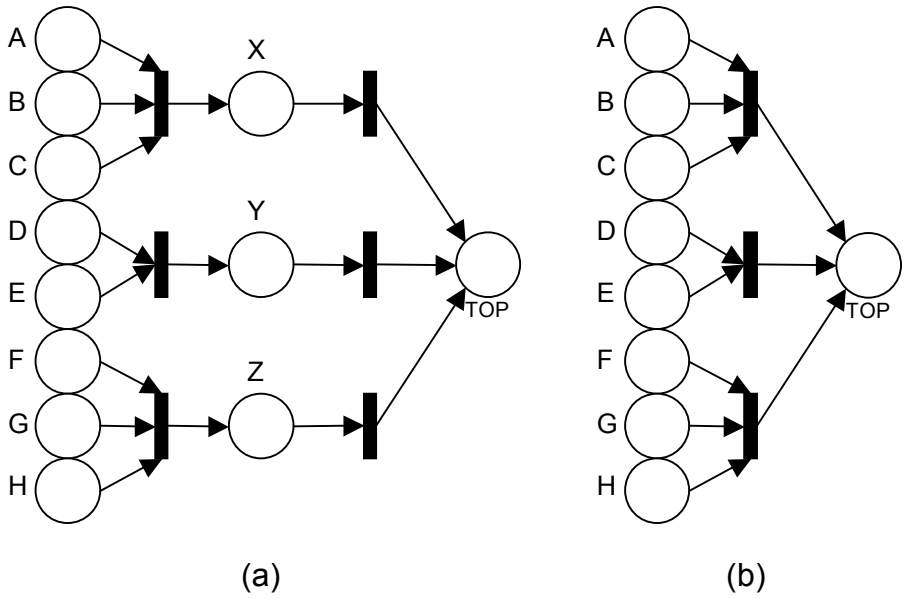


Figure 4.24 – Contraction or absorption of a PN.

4.3.4.3 Multi-Phased Missions

Phased missions were described earlier in this chapter. There have been a small number of papers that have attempted to model multi-phased missions using Petri nets and their work shall be highlighted here.

Mura and Bondavalli, in conjunction with others, have previously published papers [55, 85] concerning Phased Missions and PNs. These culminate in the most recent method, contained in their 2001 paper [76], to model more complex areas of concern of Phased Missions. Volovoi also discusses the modelling (not the computation) of phased mission reliability [75]. The 2001 Mura & Bondavalli paper [76] outlines a PN method of modelling four situations which gradually become more complex:

- The simple case, hitherto handled by all previous phased missions literature, of constant-duration phases in a fixed order. Intra-phase behaviour is modelled by a time-homogeneous Markov chain.

- Missions of different goals, either primary (a vital mission goal that must be completed unless system failures prevent this) or secondary (a less important, often optional, mission goal which can be performed based on conditions such as failures of systems, fuel levels and environment). The next phase can be dynamically decided depending on the events which occur during the mission. If, for example, a system failure prevents a primary goal being carried out, the next phases will change to include performance of a series of secondary tasks. Alternatively, if a mission is very successful, secondary goals may be scrapped, bypassing some possible intermediate phases. This then yields a tree of choices for the mission to follow.
- Phase changes are triggered by random events, creating phases of random start time and length. Even if the start times of phases are modelled by the negative exponential distribution, this is still a very complex problem.
- Intra-phase behaviour not satisfactorily modelled by time-homogeneous Markov chains. The papers published previous to Mura & Bondavalli's 2001 paper concerning Petri nets and phased missions used PNs where the system behaviour does not change with time. In this situation, transition switching delays are modelled using the negative exponential distribution, so the PN is time-homogeneous. The accuracy of modelling the reliability of systems within phases can be affected if no consideration of system aging is made. Two other papers [56, 57] have considered non-homogeneous Markov models with respect to Phased Mission reliability.

The main method suggested by Mura & Bondavalli is the use of at least two different Petri nets to model systems performing phased missions. The first, known as the Phase Net, models the tree of phase choices and the progression from the first phase to the last. The transitions model the timings of phase changes. An example phase net is shown in Figure 4.25.

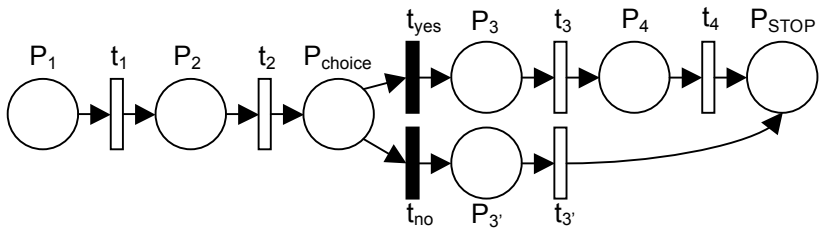


Figure 4.25 – Example Phase Net

In Figure 4.25 each phase is represented by a place. Only one of these phase places can be marked at any time. If phase place P_1 is marked, transition t_1 will switch after a time delay which is either deterministic or sampled from a distribution of phase 1 lengths. After phase 2 has completed, there is a choice of the next phase: the mission can go to phase 3 or phase 3', which may correspond to a choice between a primary or secondary objective, and may be influenced by variables in the model, such as system failures or current mission time. Regardless of the phase chosen, the mission ends once the token finishes in the P_{STOP} place.

The second PN, called the System Net, models the system failures in each phase (the authors combined this into a single PN for their example). A system failure may be modelled by the System Net and affect the choice of the specific phases performed. Similarly, component failure rates may be affected by a switch from one phase to another. The Mura and Bondavalli (2001) paper does provide an example system net, but it is rather obscure and disorganised. Instead, a good example of a system net may well be a fault tree in Petri net form, such as that in Figure 4.24(a). Success of each phase in the mission would depend upon the operation of several different systems, which themselves rely on the working of their constituent components. A component failure could then cascade through the fault tree Petri net (FTPN) and cause system, phase or mission failure.

Converting the PN to a Markov chain allows analytical solutions to be used. When this is not possible because of the system features, or the size of the PN prevents an efficient and accurate Markov solution being reached, Petri nets can be used in conjunction with simulation. This allows greater use of the flexibility of PNs, for example letting various distributions control the transition switching times, and

allows the PN modelling technique to be tailored more directly to a particular problem.

Volovoi [75] uses aging coloured tokens to attempt to simplify the modelling of systems performing phased missions. Despite this, the article is focused more on the aging of components within the phased mission system and not on providing a more accurate model of this scenario. One section involves the failure of phases, as shown in Figure 4.26.

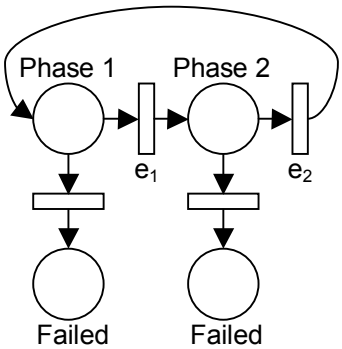


Figure 4.26 – Phase Failure Model [75]

Figure 4.26 shows two places (corresponding to phases of a mission) which run cyclically. Each phase place *i* has a transition associated with it with delay e_i. However, in Volovoi's model, the token currently in the "Phase 1" place is capable of aging when enabling either of the phase failure transitions. These transitions can age the token at different rates, simulating the effects that different phase environments might have on a common system. If the age of the token becomes too high, one of the failure transitions will switch and the system fails. As shown in Chapter 5, this model can be adapted for use with non-aging tokens in simulating the reliability of phased mission systems.

4.3.5 Summary

Petri nets are an extremely versatile and useful tool when applied to systems modelling. They have a significant capability when modelling the situations encountered in predicting system reliability. These features include component failures and repairs, logical combinations of component failures producing higher-level system failures, universal (perfect) repair, queued repair and multiple phases

within missions. If those missions have probabilistic phase choices, this can also be modelled.

While there have been many extensions of Petri nets since their conception, such as the inclusion of coloured or aging tokens, the components mentioned above form the basis of modern PNs. A Petri net with inhibitor arcs and arc multiplicities is regarded as “high-level”.

Reachability graphs (RG) detail all the possible markings of a PN from an initial state, and the transitions that must switch to achieve these. If a PN is time-homogeneous, it is known as a Stochastic or Generalised Stochastic PN (SPN or GSPN). The RGs produced for SPNs can be analysed mathematically using Markov techniques, to find the probability of reaching a particular PN marking. If a PN model is very large or aspects of it prevent this approach from being adopted due to incompatibility or inefficiency, then a technique such as discrete event simulation can be used to find these marking probabilities.

If the Petri net is used with a computational simulation tool, then the potential scope for modelling widens dramatically. A simulation PN model can include non-exponential switching time distributions, various repair schedules can be modelled, and so on. This can then use Monte-Carlo simulation to analyse any size of problem.

Perhaps one of the reasons why Petri nets are not as widely used as other methods for system reliability assessment is due to there being no unified standard definition of a Petri net. However, the basic “modern” Petri net (with timing, arc multiplicities and inhibitor arcs) can be extremely useful for modelling. Further adaptations may provide extra power, but should only be considered if the benefits over the basic PN are apparent.

Chapter 5 Development of Initial Model

5.1 Introduction

The limitations of the MFOP modelling methods presented and the suitability of Petri nets for modelling system reliability afforded an opportunity to develop a robust and flexible MFOP model. A preliminary model was created first, however, in order to prove the concept and provide a basis from which more complex aspects of MFOPs and Phased Missions can be analysed.

An MFOP can be considered as a sequence of missions which are performed back to back, without maintenance taking place between them. The simplest type of MFOP will be one where one mission is repeated a finite number of times before the MRP takes place. This scenario is the one considered in this chapter.

Each of the parts of the Petri net as used in the modelling of Phased Missions and MFOPs has a particular purpose. Places are used mainly to represent a particular state, event or a given factor or variable. For example, a place may represent the failure or operation of a component, the level of damage a system has incurred, the activity of a phase of a mission, the failure of an MFOP, etc. The presence of tokens within these places indicates that the event that the place represents has occurred, or can show what the value of a factor is. If a "component failure" event has an associated place, and this place contains a token, then the component is considered failed. Transitions move tokens between places, and thus change the status of the system or mission. For instance, the repair of a component involves the change of state from "component down" to "component up". This requires a transition to remove a token from the place representing the former state and create one in the place representing the latter. Similarly, a component failure can cascade through several levels of sub-system and system failure to cause platform failure for a phase. This cascading requires transitions to pass a token from the 'cause' place to the 'effect' place.

This chapter describes the development of a simple Phased MFOP model using ideas contained within the noted literature. The model developed can analyse a simple repetitive mission and MFOP profile containing a defined sequence of phases. The development of the PN model took place at the same time as its implementation in a simulation program. As a result, some of the decisions taken while growing the capabilities of the model came as a result of feedback from unsuccessful attempts to implement previous ideas.

5.2 Development of Model

Section 4.3.4.3 details some of the work undertaken by others in representing aspects of phased missions using Petri Nets. Figure 4.25 and Figure 4.26 show the two ideas that provided a base from which to develop further ideas. The System/Phase net idea from Mura & Bondavalli was developed to provide a more structured and rigorous modelling platform, the Phase Net being renamed to the Master PN (see Section 5.2.2), while the System Net also underwent many changes as explained in the next section. The method of modelling phase failure proposed by Volovoi [75] was also considered.

5.2.1 System Net

The purpose of the System Net (SN) is to model the failure of the system in the different phases and propagate any failures to the Phase or Master PN, so it can establish whether a phase failure (and therefore a mission and MFOP failure) has occurred. As this research models the multi-phased mission scenario, it was necessary to consider the failure logic for each individual phase in the SN. One of the simplest ways to represent the possible component-level causes of system failure in a particular phase is by using Fault Trees. As mentioned in Section 4.3.4.2, it is a very simple task to convert a FT into a PN, and so it was decided that the System Net would contain PN versions of the FTs for each phase. Each fault tree's top event (system failure in that phase), would then feed into the Master PN to propagate system failure into phase and mission failure.

A problem that became apparent in the programming of this part of the model was the usage of computer memory: the way that place and transition data were stored meant that a single System Net would be less memory efficient than having a different System Net for each phase failure FT. However, taking this step to reduce memory efficiency presented a further problem. It is likely for most systems that there are components whose failure will affect the system in more than one phase. In this instance, it was not immediately apparent where the component information would be stored and how each System Net would use it. There were several options available for this:

- It could be stored in the first phase failure PN to use that particular component. Subsequent phases affected by the failure of that component would take their inputs from it. A problem with this approach is that keeping track of all the components would be very complicated, and the phase failure PNs would very quickly lose the simplistic graphical nature of the FTs.
- Each instance of the component failure in different PNs would simply have its PN representation repeated throughout them all. This would, however, unnecessarily increase the memory taken up by the fault trees, and it would be difficult to ensure that the failure of each component was instantaneously repeated for all its instances.
- All the component failures would be modelled in a separate PN, the Component PN, which would then feed the occurrence of each failure into the correct phase failure PN as required. This would not increase the memory requirements, but would complicate its appearance further.

The third option was chosen as the most efficient way of representing the components in the system. This means that the System Net section can be broken down further into two subsections – the phase PNs and the Component PN. The different PNs communicate in this model by using *linking arcs*. These are standard arcs, shown as dotted lines to indicate their passing from one PN to another. The different PNs, therefore, could theoretically be combined into one whole – the separation is merely for ease of representation.

The three resulting Petri net types – component, phase and master – connect to each other for various reasons, as shown in Figure 5.1. The component PN links the failures of the components to the Phase PNs. The phase PNs, in turn, output their phase failures to the master PN. The master PN and the component PN interact in terms of the ending of one MFOP, the component repair process taking place, and the next MFOP beginning.

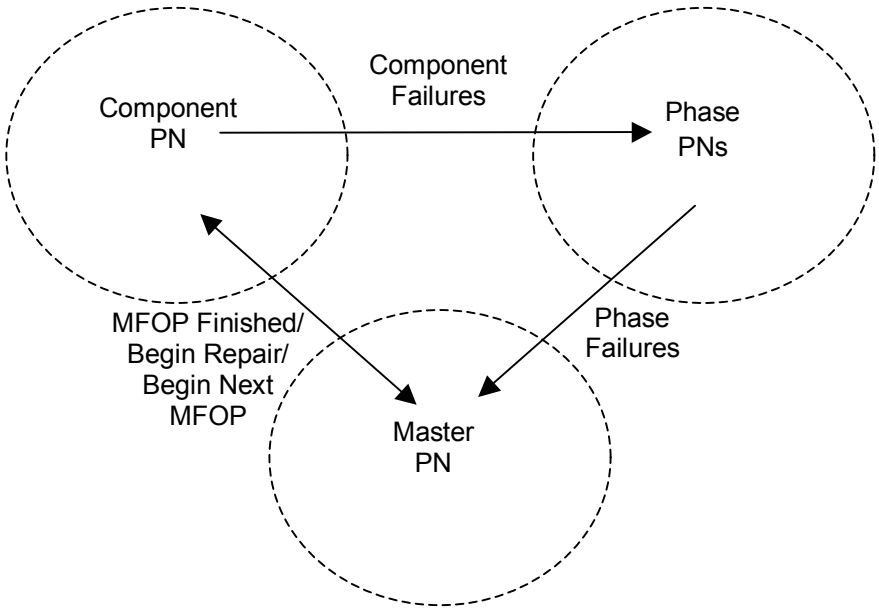


Figure 5.1 – Interactions Between Petri Nets

5.2.1.1 Phase Petri Nets

The phase Petri nets (PPNs) are the PN representation of the phase fault trees that are inputted to the model, and show the occurrence of the top event (system failure in the relevant phase) in terms of the basic events contained in the Component PN. The requirement here, therefore, is to convert and model as accurately as possible from the information that a FT provides.

Each top or intermediate event in the phase fault tree is represented in the PPN by a place. If that place has a token, the event can be considered to have occurred. The logic gates are reproduced in PN form by using immediate transitions in different ways. An example PPN is shown in Figure 5.1.

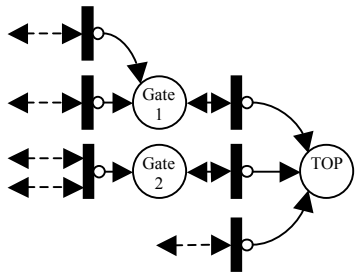


Figure 5.1– Example Phase Petri Net

When a component is simulated to have failed, a token will be deposited in a "component *n* failed" place in the Component PN. This will have a linking arc to one or more of the PPNs, and so will cascade the token down to a certain point in the PPN based on the failures of other components and the failure logic. The parts of the net facilitating this propagation are the immediate "logic gate" transitions. There are several different gates used in fault tree analysis, and these can be represented by Petri nets. Examples of OR and AND gate representations used in earlier work are shown in Figure 4.21. In this work, however, it is desirable to highlight each failed gate in a phase fault tree, regardless of whether a higher gate has also failed, due to the data that the simulation software must collect. Therefore, inhibitor arcs have been added to the AND and OR gates developed previously to enable this – see Figure 5.2 and Figure 5.3).

When there are sufficient tokens in the input places to a gate to mark the output place, a token is passed to the output place by the gate transition, and also back to the input place (see the double-edged arcs in Figure 5.2 and Figure 5.3). The inhibitor arcs from the output place to the gate transition then prevent an infinite number of transition switches, and each gate which is failed stays as marked.

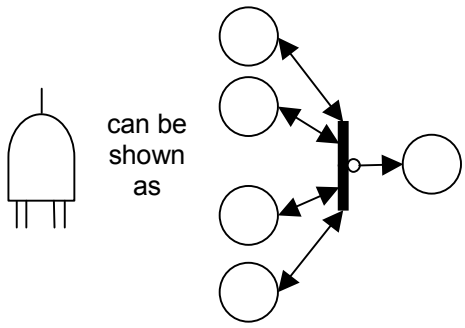


Figure 5.2 – Actual PN AND Gate Used in Model

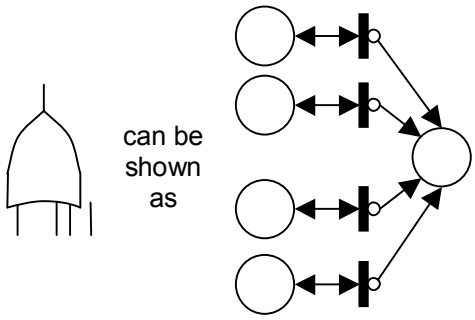


Figure 5.3 – Actual PN OR Gate Used in Model

5.2.1.2 Component Petri Net

The component Petri Net (CPN) was formed, as mentioned in Section 5.2.1, from the necessity of organising and modelling the basic event failures effectively and efficiently. A PN representation of a repairable basic event, adapted from a non-repairable version by Malhotra and Trivedi, has been shown in Figure 4.22. If a linking arc is created between the "component down" place and an input transition in one or more PPNs, this will allow the marking of the component as failed to be propagated to items higher in the system architecture.

The switching times associated with the failure transitions are randomly sampled from any one of a number of statistical distributions (negative exponential, normal, Weibull, etc); the behaviour of the repair transition depends on the maintenance policy being used in the modelling.

For instance, if each component was to be repaired individually a certain time after it failed, then each repair transition would (similar to the failure transition) have a unique time to switch. However, if, as in an MRP, the repairs will only commence at given times, the behaviour of the basic events is modelled differently. The maintenance policy used to simply model an MRP is that each failed component is instantaneously repaired at the same time as all the others. An example of a model of an MRP is shown in Figure 5.4.

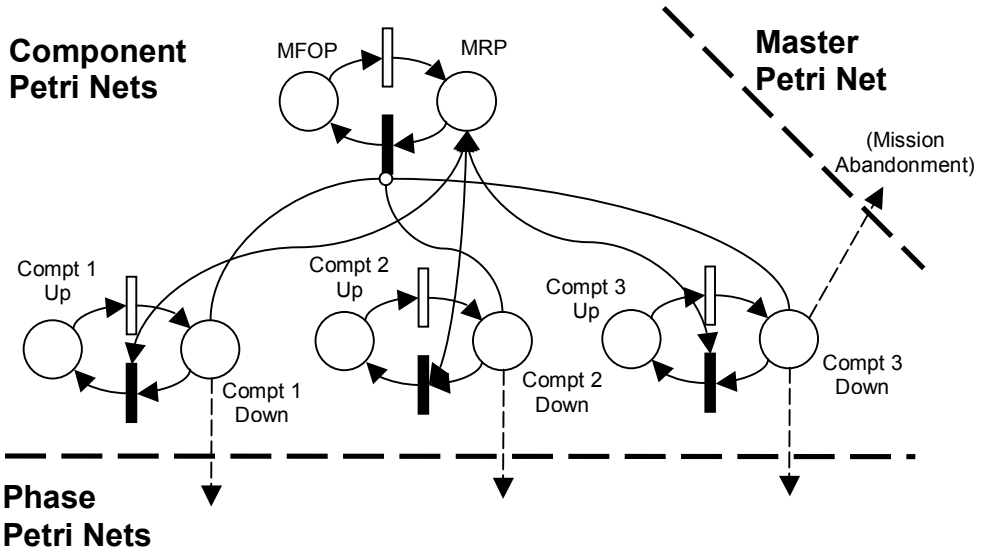


Figure 5.4 – Example Component PN

The Figure shows a set of three basic events, each with the standard "Compt *n* Up" and "Compt *n* Down" places, a timed failure transition but an immediate repair transition. Each repair transition is controlled by a "Repairing" place, shown at the top centre of Figure 5.4. During missions, a token resides in the "Not repairing" place. When an MFOP is ended, the timed transition (simulating the length of an MRP) will switch, beginning the repair process. Note that if only flying hours are considered to age components, this time should be zero. When a token is in the "Repairing" place, and components are failed, the immediate transition ending the repair period cannot switch due to the inhibitor arcs leading from the "Compt *n* Down" places. Instead, the repair place enables the repair transition for any failed basic events until each component's token is in the "Compt *n* Up" place. Only then will the "Repairing" immediate transition switch, ending the repair process and allowing a new MFOP to commence.

Also of note in Figure 5.4 is that a component failure may cause mission and MFOP abandonment, as shown by the arc leading from the "Compt 3 Down" place to the Master PN. This is explained further in Section 5.2.2.

5.2.1.2.1 Dependency Modelling

It is conceivable that dependencies can exist in the system and failures do not occur independently. In this circumstance a component may fail faster depending upon the functionality or failure of a different component or system. A simple example of this is the processor and fan in a computer – if the fan fails, the processor overheats and fails much quicker. This behaviour is dealt with in the model by allowing a component to have more than one failure transition. Only one of these can be enabled at any one time, depending on the components or systems which cause the acceleration of failure, as shown in Figure 5.5. In the Figure, the dependent component has four failure transitions, due to the dependency between two different components. Figure 5.5 shows the dependent and

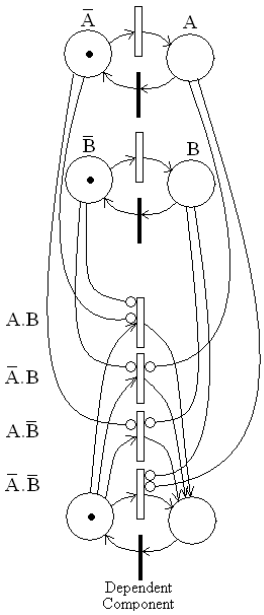


Figure 5.5 – Component Dependencies

independent components to be presently operational. This state means that transitions A.B, $\overline{A}$.B and A.$\overline{B}$ are inhibited, while transition $\overline{A}.\overline{B}$ is operational. If A fails but B remains operational, then transition $A\overline{B}$ would be newly enabled, while all the others, including $\overline{A}.\overline{B}$, would be inhibited. A new time to failure can now be sampled from an alternative distribution. This simple dependency modelling is capable only of describing the effect of other components on a component's failure rate, and not other factors such as phase change or redundancy configuration.

5.2.2 Phase Net (Master Petri Net)

The Master Petri Net (MPN) has been developed from the Phase Net idea discussed in Mura and Bondavalli. The point of this net is to show the completion of the phases in sequence, after which the mission is considered complete. Where MFOPs are considered, one mission will lead to the next without any maintenance.

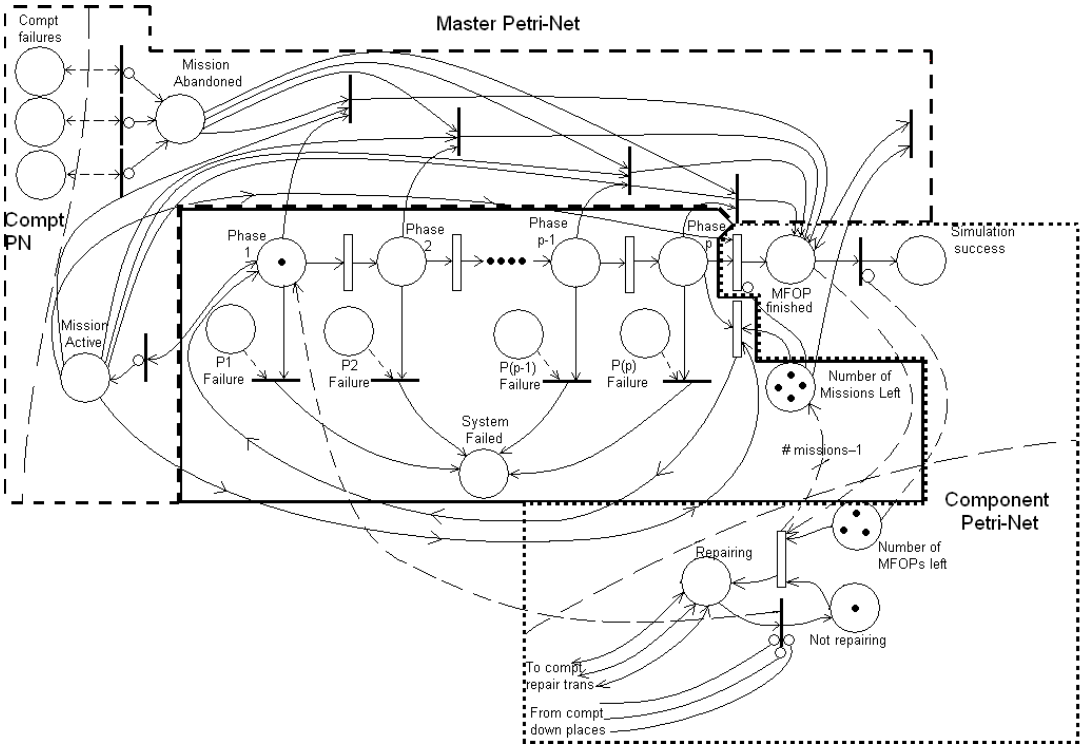


Figure 5.6 – Master Petri Net

The MPN is a complex net and as such can be considered in three interdependent sections, as shown in Figure 5.6. These sections are:

- Control of the sequence of phases, and failure or success of each mission (solid line border)
- Ending each mission or MFOP and performing repairs (dotted line border)
- Abandoning the mission due to specific component or system failures (dashed line border).

The section surrounded by a solid line controls the changing of the phases. Each phase has a place which, if marked, indicates that that phase is currently in operation. This outputs to a timed transition whose switching time is the length of the phase. Whilst these phase durations are often considered to be deterministic, it is possible that randomly sampled phase lengths may be desirable, which are as easily created. This timed transition inputs to the next phase place, whose timed transition is then enabled.

If the phase *n* top event occurs as a result of sufficient component failures, then a token will be present in the top place of the phase *n* PN such as that in Figure 5.1. This is connected to an immediate transition by a linking arc and, if phase *n* is currently in operation, allows the failure of the system in the mission. Both conditions (phase *n* execution and phase *n* failure) are required to be simultaneously true for mission failure to occur. This means that if a phase has completed before the top event occurs, then the mission can still succeed, whereas if a phase failure occurs before that phase has started, the mission will fail once the phase begins. This style of modelling phase sequences and failures was adapted from Volovoi [75, 73], see section 4.3.4.3.

The final phase place has two timed transition outputs modelling its duration, only one of which can be enabled at any time. If the mission completes successfully, it is possible either that another mission will take place straight away, before any maintenance, or that the system will enter an MRP. If the former is true, there will be a positive number of tokens in the "Number of Missions Left" place. This inhibits the top of the output transitions shown in Figure 5.6 and enables the bottom one. After this switches, it redeposits a token in the phase 1 place and restarts the mission. If the "Number of Missions Left" place is empty, the top transition is

uninhibited and enabled. This transition, upon switching, ends the current MFOP, after which the "repair" section of the MPN (the dotted part of Figure 5.6) becomes important.

The phases token will enter the "MFOP Finished" place, and, as mentioned in section 5.2.1.2, enable the "Commence Repair" transition in the component PN to begin system repair. This transition can only switch if there are a positive number of tokens in the "Number of MFOPs Left" place, and is reduced by one with each MRP that takes place. Once it switches, a number of tokens equal to one less than the number of missions to perform in each MFOP is deposited in the "Number of Missions Left" place. Once repair is complete, the token goes back to the "Not Repairing" place, and another is put in the "Phase 1" place for the next MFOP to begin. When the number of MFOPs left reaches zero, the immediate transition output from the "MFOP Finished" place is no longer inhibited. Completion of the final mission of an MFOP will thus cause a token to be placed by the transition into the "Simulation Success" place.

The final section has a dashed border in the Figure and allows for a mission to be abandoned. This abandonment occurs when critical safety-related failures occur. For example, an aircraft is returned to base to undergo repairs, prior to loss of the platform. This means that the current set of missions will cease, the MFOP is considered failed and an MRP will begin.

The modelling of abandonment is quite simple – once the "Phase 1" place is marked, it also marks a "Mission Active" place. If a particular component or sub-system fails, the relevant place representing its failure will be marked, as shown in Figure 5.4 and at the top-left of Figure 5.6. A linking arc from that place will cause an immediate transition in the MPN to become enabled, which in turn deposits a token in the "Mission Abandoned" place. Each phase place has a corresponding immediate transition which, if the mission is active and that particular phase is operational, becomes enabled upon mission abandonment and places the phase token in the "MFOP Finished" place, to begin the MRP. In this instance, there could be tokens left in the "Number of Missions Left" place, so an immediate transition removes these.

5.2.3 Simulation Model

Software simulating the Petri net to produce the overall system reliability has been developed. It takes in the phase fault trees and component failure data in text-file format, and automatically produces the different nets mentioned above. The component failure data and phase lengths (which can both be sampled from a number of distribution types such as normal, exponential, etc.) are used to give the relevant timed transitions distributions of switching times. A time for each transition to switch can be randomly sampled from these distributions, by using various methods such as those described in [2]. The simulation algorithm runs for n_s simulations as follows:

1. Randomly sample switching times for each newly enabled timed transition in each net from the switching time distribution assigned to it
2. Find the transition with the earliest switching time and switch it
3. Search through each of the immediate transitions and if any are enabled, switch them
4. Repeat step 3 until no more immediate transitions are enabled
5. Test for any of the following conditions and log them:
    a. If system has failed, begin next simulation
    b. If system has been abandoned, begin next MFOP
    c. If mission has completed, begin next mission
    d. If MFOP has completed, begin MRP
    e. If MRP has completed, begin next MFOP
    f. If all MFOPs and MRPs completed, simulation is complete – begin next simulation
6. If simulations completed < n_s, go to step 1, else end.

The results of the n_s simulations can then be outputted to a text-file for analysis.

5.3 Model Validation

To ensure that both the PN modelling method and the programmed version of are giving a satisfactory output, it is necessary to compare the results given against theoretical values. This section takes an example small system and compares the theory with the PN simulation model results for both non-repairable and repairable versions of the same system. If the results are similar, it places confidence in the ability of the model to accurately portray the mission reliability of systems undergoing MFOP. A third, larger, example has also been modelled.

5.3.1 Non-repairable Small System

In order to validate the model, results have been obtained for a simple non-repairable system and compared with the theoretical results. This simple phased-mission system was modelled with the Petri net software developed using the models described above.

The phase fault trees for the example considered are shown in Figure 5.7, with the corresponding PPNs shown in Figure 5.8. The system undergoes four phases, and is dependent upon four different components for operation. Each MFOP is assumed to consist of three missions of the four phases. The lengths of the phases and component failure characteristics are given in Table 5.1. The full Petri net for this example is given in Figure 5.9. The software described above was then executed and results obtained for the probability of mission and phase failure.

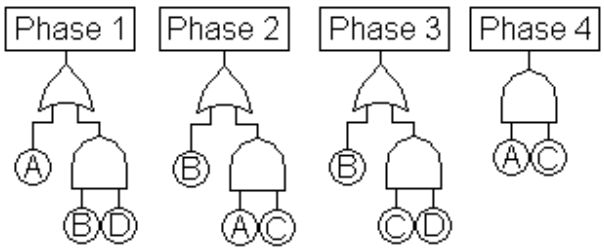


Figure 5.7 – System Phase Fault Trees

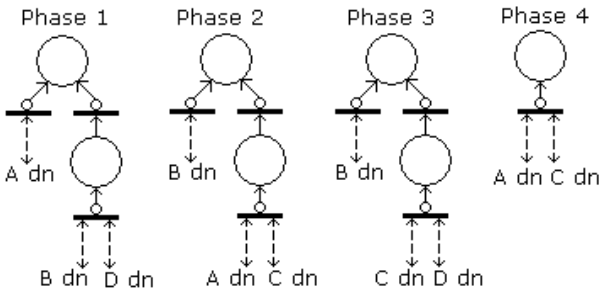


Figure 5.8 – System PPNs

Table 5.1 – Phase Lengths and Component Unreliabilities

Phase	Phase Length (Hrs)	Component	Failure Rate - λ (h^{-1})
1	0.5	A	0.0045
2	2.5	B	0.0130
3	4.0	C	0.0081
4	1.25	D	0.0011

5.3.1.1 Theoretical Analysis

In order to obtain an accurate measure of reliability for each phase in each mission of the example, it is first necessary to adapt the problem. Firstly, a three-mission period of four phases per mission without maintenance can be considered as a 12-phase non-repairable mission, with each phase fault tree occurring three times. The method of Esary and Ziehms [6] combines these phase fault trees into a single model for the mission. This method could not, however, calculate the individual probabilities of phase failure. Instead, the method of La Band & Andrews [42], as outlined in Section 4.2.1.6, can determine these phase unreliabilities, thus allowing comparison between the theory and the model. Mission unreliability is then the sum of the phase unreliabilities. This method has been used to establish values for the probability of system failure in each phase, mission and the overall MFOP.

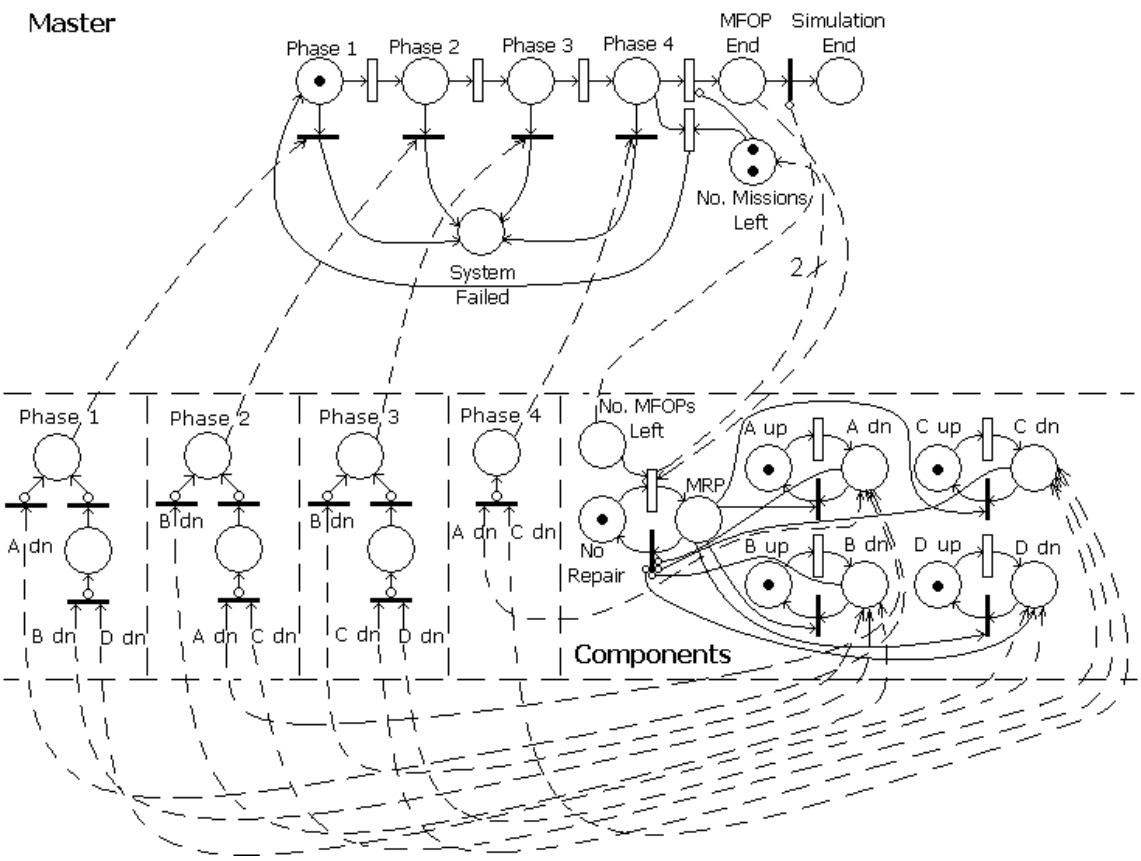


Figure 5.9 – Full PN for Example in Section 5.3.1

As an example, consider the occurrence of a system failure during phase 2 of the phased mission presented. This would be described by the prime implicants (minimal combinations of working and failed component states in non-coherent fault trees that cause top event failure) obtained from the fault tree shown in Figure 5.10. From Figure 5.10, the prime implicants of a system failure within Phase 2, represented by T_2, can be expressed in the following way:

$$\begin{aligned} T_2 &= (\text{Phase 1 Success}) \cdot (\text{Phase 2 Failure}) \\ &= \overline{(A_1 + B_1 D_1)} \cdot ((B_1 + B_2) + (A_1 + A_2) \cdot (C_1 + C_2)) \end{aligned} \tag{5.1}$$

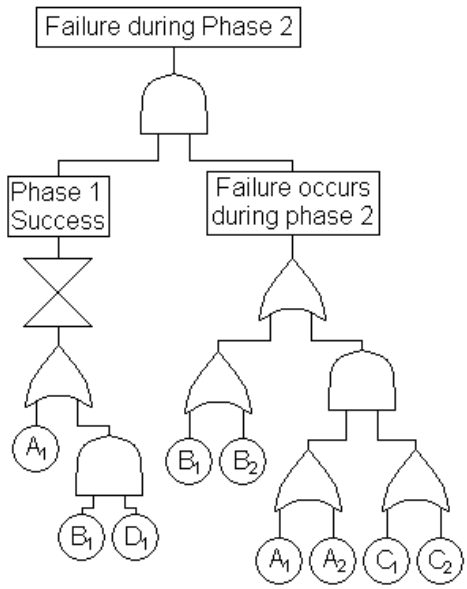


Figure 5.10 – Phase 2 Failure Fault Tree

Using the rules expressed in equations 4.25, equation 5.1 becomes:

$$\begin{aligned} T_2 &= \overline{(A_1 + B_1 D_1)} \cdot ((B_1 + B_2) + (A_1 + A_2) \cdot (C_1 + C_2)) \\ &= \left(\overline{A_1}\,\overline{D_1} + \overline{A_1}\,\overline{B_1}\right) \cdot \left(B_{1,2} + A_{1,2} C_{1,2}\right) \\ &= \overline{A_1} B_1 \overline{D_1} + A_2 C_{1,2} \overline{D_1} + \overline{A_1} B_2 + A_2 \overline{B_1} C_{1,2} \end{aligned} \qquad \textbf{5.2}$$

Once the prime implicants have been established, finding the probability of failure in phase 2 requires the application of the inclusion-exclusion expansion, as expressed in equation 2.3. As the number of prime implicants increases, it takes more terms in the inclusion-exclusion expansion to reach an exact value for the top event probability. If the amount of terms becomes large, it is necessary to approximate the overall failure probability, for example using either the rare event approximation or minimal cut-set bound methods as explained in Section 4.2.1.2. However, these approximations to the true reliability figure may not be accurate enough.

Since the simplification for sequential component phase failures uses OR logic, as shown in equations 4.25, it is more likely that using this will lead to higher numbers of prime implicants when finding overall phase failure probability ($A_{i+1,j}$ expands to give $j–i$ separate terms which would produce prime implicants). The simplification for component success in subsequent phases, however, inherently considers them

with AND logic ($\overline{A_{i+1,j}}$ has just one term in the prime implicant). Therefore, the expression for success up to the end of a phase is generally simpler than the expression for its failure. This can be exploited to give a more efficient way of finding the exact conditional phase failure probability, using equation 5.3.

$$\begin{aligned}\Pr(\text{Phase } p \text{ Failure}) &= 1 - \Pr(\text{Phase } p \text{ Success} \mid \text{Phase } p-1 \text{ success}) \\ &= 1 - \frac{\Pr(\text{Success up to end of Phase } p)}{\Pr(\text{Success up to end of Phase } p-1)}\end{aligned}$$

5.3

For instance, the success up to the end of phase 2 has the path sets (combinations of component failure and success events which cause the system to function) shown in equation 5.4.

$$\begin{aligned}\overline{T_2} &= (\text{Success in P}_2).(\text{Success up to P}_2) \\ &= \overline{(B_{1,2} + A_{1,2}C_{1,2})}.\overline{(A_1 + B_1D_1)} \\ &= (\overline{B_{1,2}}\,\overline{A_{1,2}} + \overline{B_{1,2}}\,\overline{C_{1,2}}).(\overline{A_1}\,\overline{B_1} + \overline{A_1}\,\overline{D_1}) \\ &= \overline{A_{1,2}}\,\overline{B_{1,2}} + \overline{A_1}\,\overline{B_{1,2}}\,\overline{C_{1,2}}\end{aligned}$$

5.4

Using the inclusion-exclusion expression given in equation 2.3, the mission success up to the end of phase 2, $R_{1,2}$, can be determined from equation 5.5:

$$\begin{aligned}R_{1,2} &= \Pr(\overline{A_{1,2}}\,\overline{B_{1,2}}) + \Pr(\overline{A_1}\,\overline{B_{1,2}}\,\overline{C_{1,2}}) - \Pr(\overline{A_{1,2}}\,\overline{B_{1,2}}\,\overline{C_{1,2}}) \\ &= r_{A_{1,2}} r_{B_{1,2}} + r_{A_1} r_{B_{1,2}} r_{C_{1,2}} - r_{A_{1,2}} r_{B_{1,2}} r_{C_{1,2}}\end{aligned}$$

5.5

The evaluation of the failure or success probabilities of each basic event can be determined using equation 5.6.

$$\begin{aligned}r_{A_{i,j}} &= 1 - q_{A_{i,j}} \\ q_{A_{i,j}} &= q_A(t_{i-1}, t_j) = \int_{t_{i-1}}^{t_j} f_A(t)dt = F_A(t_j) - F_A(t_{i-1})\end{aligned}$$

5.6

For constant failure rates these become:

$$q_{A_{i,j}} = (1 - e^{-\lambda_A t_j}) - (1 - e^{-\lambda_A t_{i-1}}) = e^{-\lambda_A t_{i-1}} - e^{-\lambda_A t_j}$$
$$r_{A_{i,j}} = 1 - q_{A_{i,j}} = 1 + e^{-\lambda_A t_j} - e^{-\lambda_A t_{i-1}}$$

5.7

Applying this to the expression for phase 2 success gives:

$$r_{A_1} = 1 + e^{-0.0045\times0.5} - e^{-0.0045\times0} = 1 + 0.9978 - 1 = 0.9978$$
$$r_{A_{1,2}} = 1 + e^{-0.0045\times3} - e^{-0.0045\times0} = 1 + 0.9866 - 1 = 0.9866$$
$$r_{B_{1,2}} = 1 + e^{-0.013\times3} - e^{-0.013\times0} = 1 + 0.9618 - 1 = 0.9618$$
$$r_{C_{1,2}} = 1 + e^{-0.0081\times3} - e^{-0.0081\times0} = 1 + 0.9760 - 1 = 0.9760$$
$$r_{A_{1,2}} r_{B_{1,2}} = 0.9489$$
$$r_{A_1} r_{B_{1,2}} r_{C_{1,2}} = 0.9366$$
$$r_{A_{1,2}} r_{B_{1,2}} r_{C_{1,2}} = 0.9261$$

Hence, from equation 5.5,

$$\begin{aligned} R_{1,2} &= 0.9489 + 0.9366 - 0.9261 \\ &= 0.9593 \end{aligned}$$

5.8

All values given are to four decimal places. A similar application of the previous steps to the path sets of success of phase 1, $\overline{A_1}\,\overline{B_1}$ and $\overline{A_1}\,\overline{D_1}$, yields the phase reliability to be $R_1 = 0.9977$. Using this and equations 5.3 and 5.8, the unreliability for phase 2 is therefore:

$$Q_2 = 1 - R_2 = 1 - \frac{R_{1,2}}{R_1} = 1 - \frac{0.9593}{0.9977} = 1 - 0.9615 = 0.0385$$

The alternative approach to calculating this value is to use the four prime implicants for phase 2 given in equation 5.2, and use the inclusion-exclusion given in equation 2.3. For later phases, the calculations saved by using the method described above

increases. This method was applied to obtain the unreliability in each of the twelve mission phases. The results are shown in Table 5.2.

Table 5.2 – Prime Implicants and Unreliabilities of Mission Phases

Phase	Prime Implicants (Phase Success)	Mission Reliability at Phase end	Phase Unreliability
1	$\overline{A_1}\,\overline{D_1}, \overline{A_1}\,\overline{B_1}$	0.9977	0.00225
2	$\overline{A_{1,2}}\,\overline{B_{1,2}}, \overline{A_1}\,\overline{B_{1,2}}\,\overline{C_{1,2}}$	0.9593	0.03850
3	$\overline{A_1}\,\overline{B_{1,3}}\,\overline{C_{1,3}}, \overline{A_{1,2}}\,\overline{B_{1,3}}\,\overline{D_{1,3}}, \overline{A_1}\,\overline{B_{1,3}}\,\overline{C_{1,2}}\,\overline{D_{1,3}}$	0.9103	0.05107
4	$\overline{A_{1,4}}\,\overline{B_{1,3}}\,\overline{C_{1,3}}, \overline{A_{1,4}}\,\overline{B_{1,3}}\,\overline{D_{1,3}}, \overline{A_1}\,\overline{B_{1,3}}\,\overline{C_{1,4}}$	0.9086	0.00194
5	$\overline{A_{1,5}}\,\overline{B_{1,5}}\,\overline{C_{1,3}}, \overline{A_{1,5}}\,\overline{B_{1,5}}\,\overline{D_{1,3}}, \overline{A_{1,5}}\,\overline{B_{1,3}}\,\overline{D_{1,5}}$	0.8772	0.03452
6	$\overline{A_{1,6}}\,\overline{B_{1,6}}\,\overline{C_{1,3}}, \overline{A_{1,6}}\,\overline{B_{1,6}}\,\overline{D_{1,3}}, \overline{A_{1,5}}\,\overline{B_{1,6}}\,\overline{C_{1,6}}$	0.8294	0.05448
7	$\overline{A_{1,5}}\,\overline{B_{1,7}}\,\overline{C_{1,7}}, \overline{A_{1,6}}\,\overline{B_{1,7}}\,\overline{D_{1,7}}, \overline{A_{1,5}}\,\overline{B_{1,7}}\,\overline{C_{1,6}}\,\overline{D_{1,7}}$	0.7862	0.05210
8	$\overline{A_{1,8}}\,\overline{B_{1,7}}\,\overline{C_{1,7}}, \overline{A_{1,8}}\,\overline{B_{1,7}}\,\overline{D_{1,7}}, \overline{A_{1,5}}\,\overline{B_{1,7}}\,\overline{C_{1,8}}$	0.7836	0.00327
9	$\overline{A_{1,9}}\,\overline{B_{1,9}}\,\overline{C_{1,7}}, \overline{A_{1,9}}\,\overline{B_{1,9}}\,\overline{D_{1,7}}, \overline{A_{1,9}}\,\overline{B_{1,7}}\,\overline{D_{1,9}}$	0.7580	0.03271
10	$\overline{A_{1,10}}\,\overline{B_{1,10}}\,\overline{C_{1,7}}, \overline{A_{1,10}}\,\overline{B_{1,10}}\,\overline{D_{1,7}}, \overline{A_{1,9}}\,\overline{B_{1,10}}\,\overline{C_{1,10}}$	0.7164	0.05493
11	$\overline{A_{1,9}}\,\overline{B_{1,11}}\,\overline{C_{1,11}}, \overline{A_{1,10}}\,\overline{B_{1,11}}\,\overline{D_{1,11}}, \overline{A_{1,9}}\,\overline{B_{1,11}}\,\overline{C_{1,10}}\,\overline{D_{1,11}}$	0.6784	0.05302
12	$\overline{A_{1,12}}\,\overline{B_{1,11}}\,\overline{C_{1,11}}, \overline{A_{1,12}}\,\overline{B_{1,11}}\,\overline{D_{1,11}}, \overline{A_{1,9}}\,\overline{B_{1,11}}\,\overline{C_{1,12}}$	0.6753	0.00451

5.3.1.2 Petri Net Model Results

The Petri net model was used in the simulation program to randomly sample from the appropriate distributions to produce timed transitions' switching times. To ensure a good convergence of the model to the reliability, 10,000,000 simulations were performed, with three four-phase missions in each maintenance-free operating period, and one MFOP per simulation. The system is, in effect, non-repairable. Table 5.3 shows the results obtained.

Table 5.3 – Results of Simulated Model

Phase	1	2	3	4	5	6
Phase Failures	22258	384042	490352	17581	312836	478172
Phases Started	10000000	9977742	9593700	9103348	9085767	8772931
Unreliability	0.00223	0.03849	0.05111	0.00193	0.03443	0.05451

Phase	7	8	9	10	11	12
Phase Failures	432422	25978	257002	416899	379478	30325
Phases Started	8294759	7862337	7836359	7579357	7162458	6782980
Unreliability	0.05213	0.00330	0.03280	0.05500	0.05298	0.00447

The software is capable of producing tables for each phase which show the instantaneous unreliability at regularly spaced points, for investigation of the convergence of the simulation results. The convergence of the first phase of the MFOP over the 10,000,000 simulations performed is shown in Figure 5.11. The graph shows the number of simulations against reliability level, measured every 200,000 simulations.

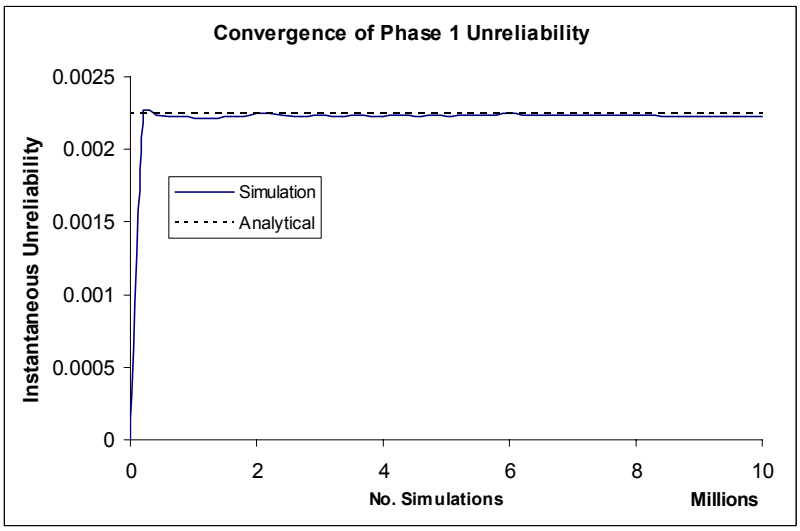


Figure 5.11 – Convergence of Phase 1 Unreliability

As can be seen from Figure 5.11, the convergence to the approximate phase unreliability value is very swift – the first point recorded, at 200,000 simulations, is just above the actual unreliability. Executing further simulations does not lead to a

much more accurate estimate for the system reliability, and for most of the rest of the set of simulations, this estimated value stays below the real figure.

5.3.1.3 Comparison of Results

Table 5.4 shows a comparison between the conditional phase failure probabilities produced by the Petri net software and the analytical solution. All, except the first phase, show a percentage error of less than 1%. The convergence of the model over the ten million simulations is very swift as Figure 5.11 shows. A good estimate to the unreliability can be established after around 600,000 simulations (within 2% of the analytical solution). Extending the analytical approach to a much larger system, as is typically found within industry, would require extensive calculations. However, extending the simulation model does not require a much greater increase in the length of time to find phase reliability.

Table 5.4 – Comparison of Analytical and Simulation Unreliability Results

Phase	1	2	3	4	5	6
Analysis	0.00225	0.03850	0.05107	0.00194	0.03452	0.05447
Simulation	0.00223	0.03849	0.05111	0.00193	0.03443	0.05451
Per cent Error	1.118%	0.037%	0.081%	0.246%	0.247%	0.057%

Phase	7	8	9	10	11	12
Analysis	0.05210	0.00327	0.03271	0.05493	0.05302	0.00451
Simulation	0.05213	0.00330	0.03280	0.05500	0.05298	0.00447
Per cent Error	0.069%	0.947%	0.253%	0.141%	0.081%	0.840%

5.3.2 Repairable Small System

Further demonstration of the accuracy of the modelling method is provided in the following example. This considers the same system, MFOP, mission and phase profile as that described in Section 5.3.1, but in this case the components are repairable. Components A and B, which have revealed failures, are repaired after each MFOP if failed; while components C and D, which have unrevealed failures, are inspected during every third MRP and repaired if failed at that point.

In order to provide analytical reliability figures for the repairable situation, a Markov model was produced which models the deterioration of the system from having all components working at time t = 0, over 12 MFOPs. The approach is taken from Clarotti *et al* [48], and details are provided in the next section.

5.3.2.1 Markov Analysis

The Markov model for the component failures is shown in Figure 5.12.

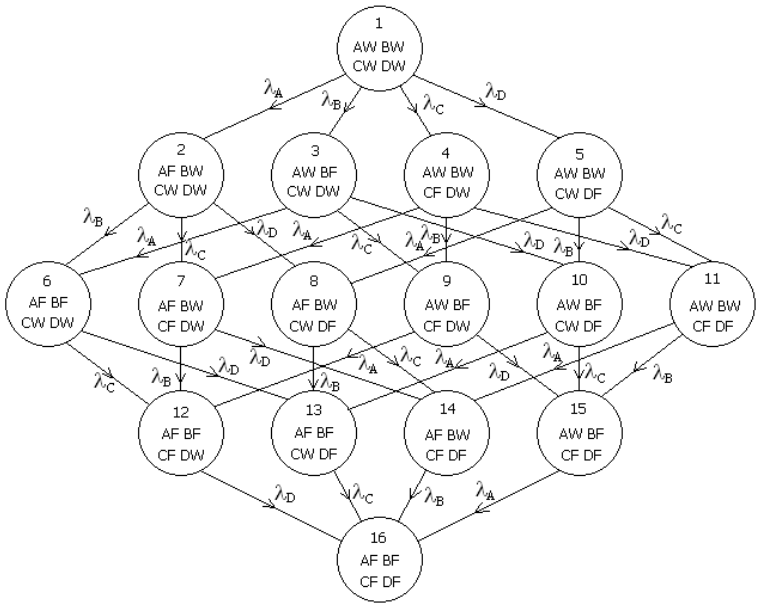


Figure 5.12 – Markov Model of Component Failures

Table 5.5 shows the system failure states for each phase, according to the state numbers in Figure 5.12. Since states listed in the second column of Table 5.5 lead to system failure, they are made to be absorbing states for the relevant phase and all transitions leading from them are disabled. For instance, the fault tree for phase 1 has cut-sets {A} and {B.D} (see Figure 5.7). Hence, state 2 (A failed, all other components operational) cannot lead to any other states, as it causes system and phase failure. The three transitions leading from this state are therefore disabled. States marked with an asterisk are unreachable states in that phase.

Table 5.5 – Failure and Acceptable States for Phases

Phase i	Failure States in i	Acceptable States at t_i
1,5,9	2,6,7,8,10,12,13*,14,15,16*	1,4,5,11
2,6,10	3,6,7,9,10,12*,13,14,15,16*	1,2,4,5,8
3,7,11	3,6,9,10,11,12,13,14,15*,16*	1,2,4,5,8
4,8	7,12,14,16	1,3,4,5,9,11
12	7,12,14,16	1,2,3,4,5,6,8,9,10,11,13,15

At time t_i (the end of phase i), in order for transition between phases, the system needs to occupy a state which causes failure neither in phase i nor in phase i+1. The third column in Table 5.5 shows the states that satisfy these requirements for each phase. The repetition of the mission profile within the MFOP leads to repeated sets of failure states for given phases. However, while phases 4 and 8 lead back to the profile of phase 1, phase 12 leads to a repair state, so its acceptable states at the phase end time differ.

The probability of the system being in each state at phase change is calculated, given the failure rates and phase lengths shown in
Table 5.1. This yields the probability of system success or failure at t_i. The probabilities of being in each of the acceptable states at t_i form the initial conditions for the next phase. The process of calculating the initial conditions for the next phase continues until the repair phase. When repair takes place, if components C and D are not inspected, the system is returned to any of the states 1, 4, 5 or 11. If they are inspected, the next MFOP will begin from state 1.

The Markov analysis tool within the modelling software FaultTree+ was used to find the MFOP success and failure rates. This uses numerical integration methods to find the state probabilities after set times. These figures were also predicted using the Petri net modelling software after performing 10,000,000 simulations. Both sets of results are given in Table 5.6 and results from the two models are compared in the next section.

Table 5.6 – Results from Markov Analysis and Petri Net Model

MFOP	Platform unreliability up to end of MFOP (Markov)	MFOP Failure Prob. (Markov)	MFOP Failure Prob. (Petri Net Model)	Percentage Error between results
1	0.32473	0.324730	0.324872	0.0437%
2	0.5498	0.333304	0.333222	0.0246%
3	0.7026	0.339405	0.339228	0.0522%
4	0.7992	0.324815	0.324865	0.0154%
5	0.8661	0.333167	0.332750	0.1252%
6	0.91157	0.339582	0.339362	0.0648%
7	0.94029	0.324777	0.323946	0.2559%
8	0.96018	0.333110	0.333117	0.0021%
9	0.97370	0.339528	0.339884	0.1049%
10	0.98224	0.324715	0.324741	0.0080%
11	0.98816	0.333333	0.333041	0.0876%
12	0.992179	0.339443	0.339061	0.1125%

5.3.2.2 Comparison of Results

A comparison of the MFOP failure rates produced by the Markov model and that produced by the Petri net model software (as shown in Table 5.6) shows a high degree of correlation. Due to the possible repair of C and D only after the third MFOP, the results follow a cycle where the MFOP failure probability increases slowly, and then returns to the initial level after the third MRP. There is a maximum of just over a quarter of a per cent error between any two values, and a mean error over all twelve MFOPs of 0.0747%. The correlation between the two sets of results gives confidence that the Petri net modelling software can accurately predict the reliability of repairable phased systems.

The convergence of the failure rate of the first MFOP over the run of simulations is shown in Figure 5.13.

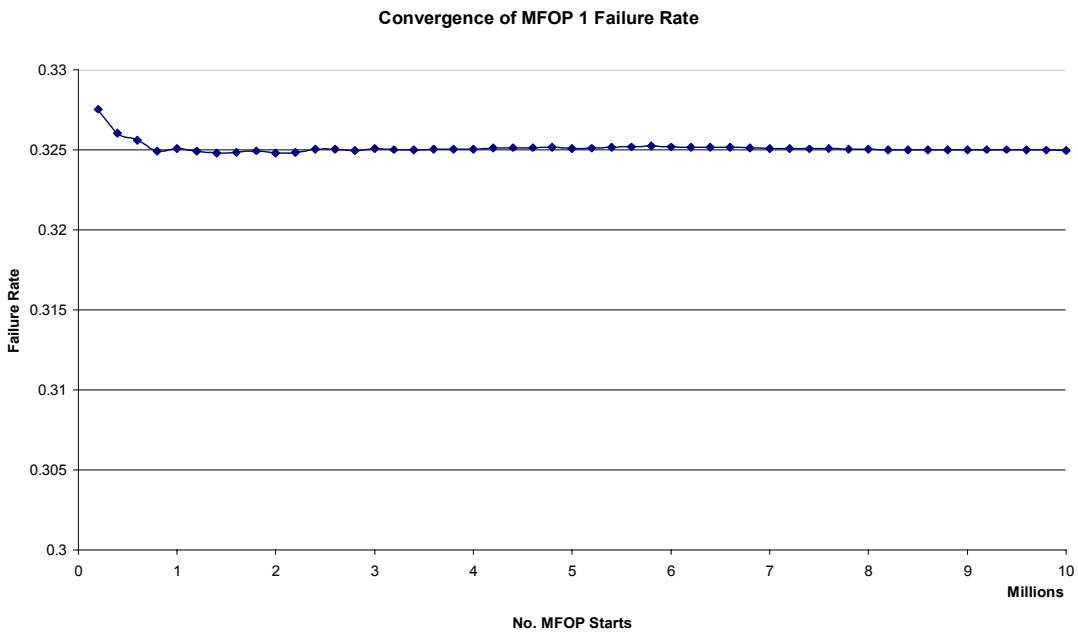


Figure 5.13 – Convergence of MFOP 1 Failure Rate

5.3.3 Larger Repairable System

This section gives the details and results of using the modelling method to analyse a larger, 10-phase, 10-component system, including MFOP complexities mentioned in Section 5.2.2. Table 5.7 shows the component failure rates and dependencies. Table 5.8 gives, for each phase, the durations and components which can cause abandonment. The dependencies column shows the new failure rate of the main component if the component listed in the column fails. Figure 5.14 shows the fault trees for the 10 phases.

Three missions were performed in each MFOP and three MFOPs were carried out in each simulation. Every component failure is repaired in each MRP.

All this detail was entered into the modelling software and 1,000,000 simulations were performed to reach an estimate of the likelihoods of failure of the MFOPs, missions and phases.

Table 5.7 – Component Failure Data

Component	Failure Rate	Dependencies
A	Exponential, λ=0.0025	D – Exp, λ=0.0015 B – Normal, μ=150, σ=20 Both - Exp, λ=0.000158
B	Weibull, β=3.2, η=130	
C	Exp, λ=0.0041	J – Exp, λ=0.0012
D	Lognormal, μ=5, σ=0.7	
E	Normal, μ=140, σ=23	
F	Exp, λ=0.0016	
G	Norm, μ=160, σ=19	A – Exp, λ=0.056 H – Lnorm, μ=5.2, σ=1.2 Both – Weib, β=1.3, η=340
H	Exp, λ=0.00036	
I	Weib, β=2.1, η=240	
J	Exponential, λ=0.0024	

Table 5.8 – Phase Lengths & Abandonments

Phase	Duration (Hours)	Abandonments
1	0.5	J
2	2.5	D, G
3	4.0	
4	1.25	D
5	Normal (μ=3.4, σ=0.2)	A, B, I, J
6	0.6	G
7	Exponential (λ=0.14)	G
8	3.2	E, H
9	2.1	
10	Normal (μ=5.6, σ=0.4)	C, D, E

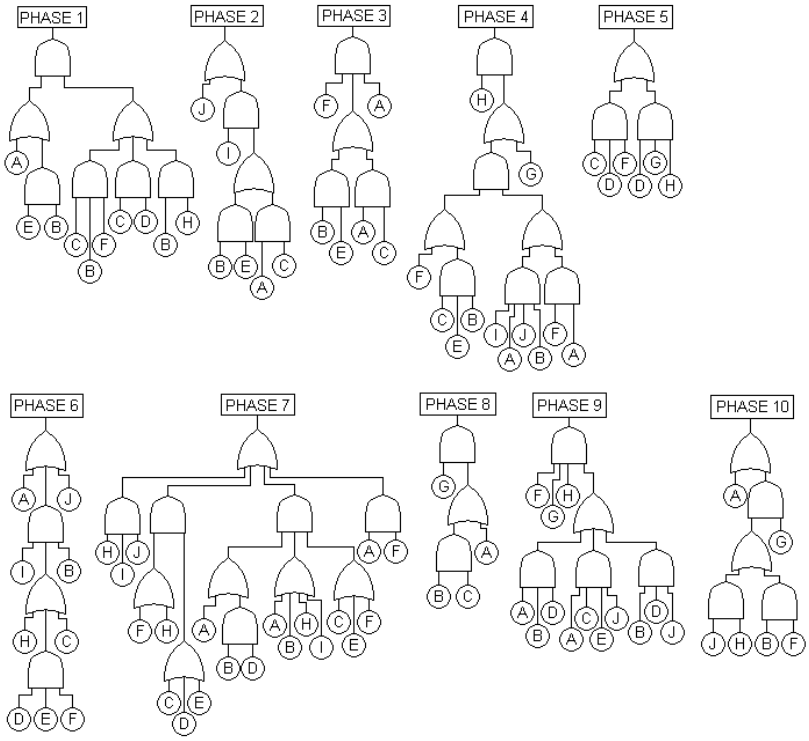


Figure 5.14 – Larger System Phase Fault Trees

5.3.3.1 Results

Table 5.9 shows the number of simulations, failures, abandonments and conditional failure probabilities for each of the MFOPs and missions, as outputted from the program. Table 5.10 then shows the number of failures of each phase in each mission, and their total failure probability.

Table 5.9 – MFOP and Mission Failure Results

MFOP/ Mission	Starts	Failures	Abandon-ments	Failure Prob.
MFOP 1	1000000	113270	618047	0.731317
MFOP 2	886730	60814	766730	0.933253
MFOP 3	825916	34780	778346	0.984514
Mission 1	2712646	105400	928724	0.381223
Mission 2	1678522	67725	805461	0.520211
Mission 3	805336	35739	428938	0.576998

Table 5.10 – Phase Failure Results

Phase	Starts	Failures in			Total Failures	Failure Prob.
		Mission 1	Mission 2	Mission 3		
1	5196504	3300	72030	34800	110130	0.021193
2	5086374	45587	60528	33264	139379	0.027402
3	4946995	9	27	24	60	0.000012
4	4946935	37868	30475	20633	88976	0.017986
5	4857959	274334	310171	187647	772152	0.158946
6	4085807	32128	22453	9263	63844	0.015626
7	4021963	51337	41698	22158	115193	0.028641
8	3906770	151011	88526	36544	276081	0.070667
9	3630689	0	0	0	0	0
10	3630689	438550	247278	120344	806172	0.222044

The convergence of the MFOP failure rates (without abandonments) over the run of 1,000,000 simulations is shown in Figure 5.15.

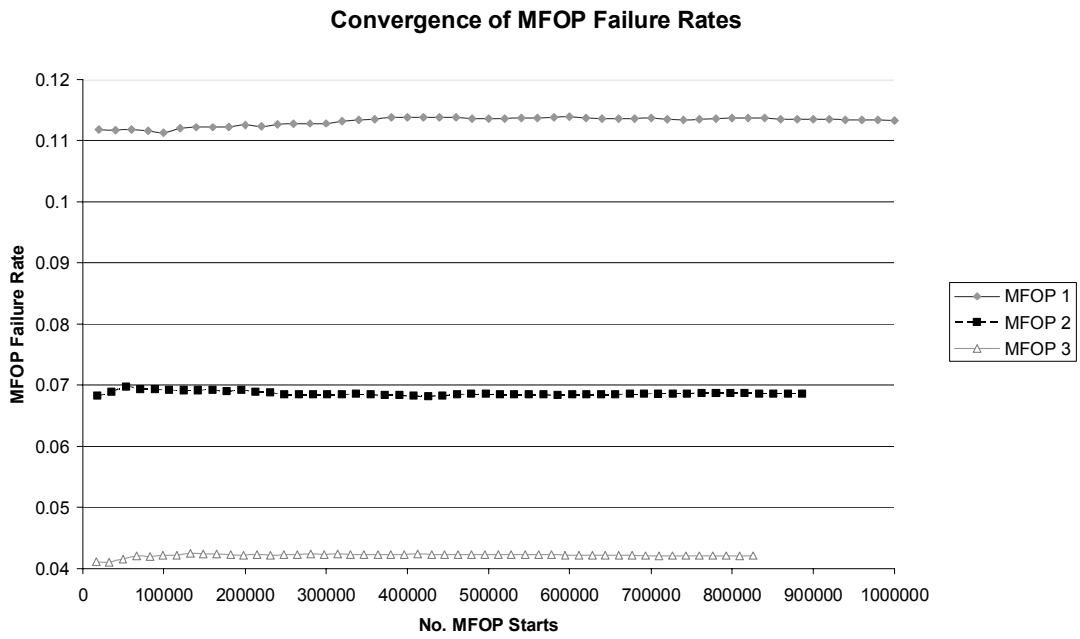


Figure 5.15 – Convergence of MFOP Failure Rates for Third Example

5.4 Summary

Petri nets provide an effective, easily understood and very powerful way of predicting the reliability of a system or platform. The Petri net technique extends to the area of Phased Missions, where complexities of modelling such as component failure rate dependencies, varying distributions and repairable systems are easily included. The technique can also be used to model a basic Maintenance-Free Operating Period.

The model outlined in this chapter can account for various reliability considerations such as component, system, phase, mission and MFOP failure, mission abandonment, the MRP and component failures affecting the failure rate of another component.

Comparing the results from the Petri net model with an analytical solution for an example non-repairable system, the correlation is good. System performance prediction from the model can also be reached without the complexities of finding a qualitative solution. It can produce the probability of failure in a particular phase as well as the overall system reliability. The solution speed is fast and the approach more efficient than that for the analytical solution, even for a small system. The results when applied to a repairable system are also accurate compared to a Markov model, providing confidence in the capabilities of the Petri net modelling method. Its capabilities for simulating larger systems have also been demonstrated.

Because the performance of the preliminary Petri net model has been demonstrated to be good, it is worthwhile to develop it further, including more complex aspects of phased missions and MFOPs. The development of this more complex model is covered in the next chapter.

Chapter 6 Development of Complex Model

6.1 Introduction

The successful development of a simple Petri net modelling technique for MFOPs and Phased Missions creates an opportunity to expand its scope. In order to provide a method of modelling the performance of MFOPs and phased missions as they would be applied in the real-world, many more considerations need to be made. Chapter 3 mentions the technologies and ideas identified thus far which will help to bring about a successful and lengthy MFOP. This includes redundancies, prognostics and so on. Section 4.2 discusses the development of various ways of modelling phased missions, and some of the latest research which suggests more complex aspects of it, such as phase insertions and phase choice. It is worthwhile to produce a method of modelling these additions to the main concepts of MFOPs and Phased Missions.

This chapter covers the development of the simple Phased Mission and MFOP Petri net model into one which considers important, relevant complexities of both Phased Missions and MFOPs. These are discussed and then the ways in which they are simulated using Petri nets is explained.

6.2 Phased Missions

6.2.1 Phase Lengths

The application of the modelling method proposed here as a simulation tool allows for a wide variety of distribution types to be considered (exponential, Weibull, normal, etc.). These distributions can be applied not only to component failure rates, but also to phase lengths. If one phase's length is randomly sampled, it is conceivable that a second phase may need to have a length based on that of the first. For instance, in a typical aircraft mission, the "return to base" phase length may

need to mirror that of the "fly to mission" phase. As such, two further options have been made available as phase lengths:

- *Reverse* a phase length: the time applied to the reverse phase mirrors that of the original phase. The length of the reversing phase is equal to the total time the original phase was active for. If the original phase was never carried out, the time to complete for the reversing phase will be zero.

- *Match* a phase length: the time for the matching phase to complete is equal to the *time left* for the original phase to complete. If the original phase has completed, the time to complete for the matching phase will be zero.

6.2.2 Probability Transitions

In order to model a situation where only one of several outcomes can happen, with a given probability, an addition to the Petri Net modelling method has been created. The previous way of modelling this situation, shown in Figure 6.1, would have been to have two transitions (immediate or not) with the same time to switch, both taking an input from the same initial place. These would then output to separate places, one for each transition. Only one transition can take the token from the initial place, and the system would apply a probability check to find which one. However, it is difficult to implement this system in a computer model, and so the Petri net method has been extended to include *probability transitions* and *probability arcs*.

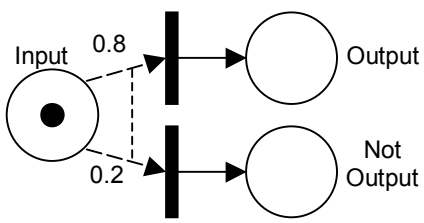


Figure 6.1 – Previous Method

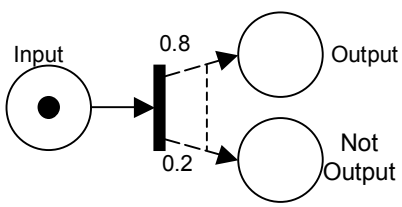


Figure 6.2 – New Probability Method

As shown in Figure 6.2, the adaptation of the PN method involves a single transition which has arcs which input to the output places with given probabilities. Otherwise, a probability transition retains the same functionality as normal transitions, such as

the ability to have a time delay, inhibition and output arcs which do not have a probability attached.

A probability transition must have enough probability arcs such that their sum equals 1, as the probabilities on these arcs are interdependent: only one of them will be selected for an output. Thus, in Figure 6.2, the place "Output" will receive a token in 80% of switches, while "!Output" receives it for the rest of the time.

This type of transition has found its use in applications such as probability-driven phase choices (section 6.2.12.2) and discrete events (section 6.3.2).

6.2.3 Fleet of Platforms

A typical real-life situation when considering platform reliability in action is that a fleet of platforms may be available to carry out many different combinations of missions. Each platform is identical to the others, in that they all have the same components, systems, prognostic and diagnostic capabilities, and so on. (Note that a "system" refers to any intermediate or top event of a phase fault tree – each of these events can very loosely be considered to correspond to the failure of a system in some way). Each set will, however, act independently from the others. One way of modelling this situation would be to have a different set of Petri Nets (Master, Component and Phases) for each platform. As the fleet is of identical platforms, this would create an unnecessary and inefficient increase in the memory required to simulate the scenario. Instead, coloured tokens, as described in Section 4.3.3.6, have been used to represent each of the platforms.

Each platform in the fleet is assigned a unique "colour". Only one set of component, phase and master PNs is then needed to model the entire fleet. Transitions treat each colour separately: for a transition to be enabled with a colour, each of the input places to the transition must have the correct number of tokens *of that colour*. The sampled switch times that the transition generates will differ for each different colour the transition is enabled in. Once the transition switches, only tokens of the switched colour are created in the output places.

For example, Figure 6.3 shows a Component PN with coloured tokens, for a three-plane fleet. Each plane has the same components as the others, and using the coloured PN rules stated above, each transition for each component will have a different time to switch, representing the varying time to fail for each component in the different planes. Component 1 in the "white" plane, for instance, may have a time to fail of 431.8 flight hours, while the same component in the "black" plane could fail after 5089.6 flight hours and that in the "grey" plane has a life of 3491.6 flight hours. The same transition will switch the different token at different times.

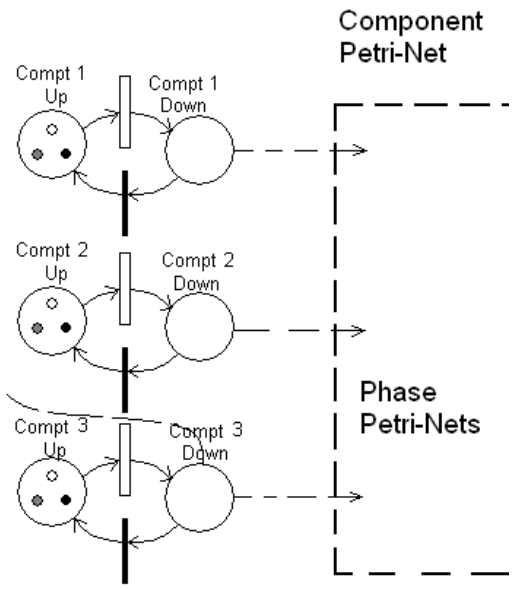


Figure 6.3 – Component PN with Coloured Tokens

Similarly, each plane will carry out a different set of MFOPs and missions, and will fail at different times and in different ways. The modelling of this is covered in the next section.

6.2.4 Platform Operation

In a typical scenario considering MFOPs, a platform might be expected to carry out any number and type of missions in a given interval, before maintenance takes place. The different planes in the fleet will be expected to perform different missions at the same time, and will therefore undergo different stresses and failures concurrently.

In simulating this situation, a "platform" has been considered to be an entity which carries out a distinct sequence of MFOPs, independent of other platforms in the fleet, as shown in Figure 6.4. The PNs for modelling the fleet are contained within the Master PN, as are those for modelling MFOPs and Missions.

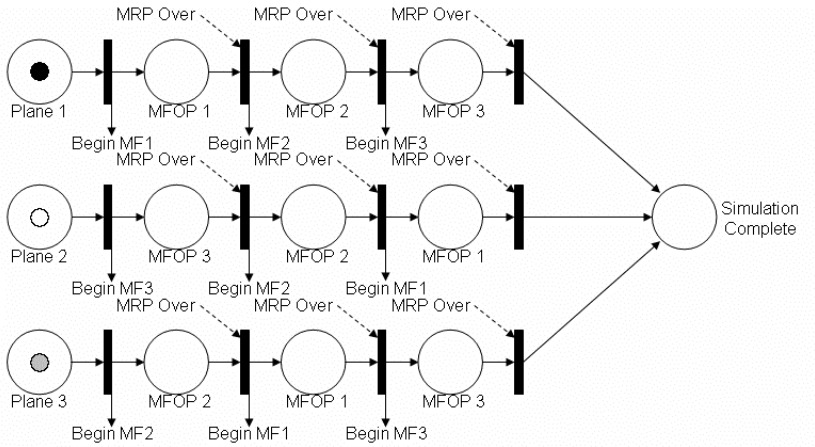


Figure 6.4 – Fleet PN Model

Figure 6.4 shows three "Planes", each with a differently coloured token in the "Plane p" place, where $p = 1,\ldots,n_{PL}$, the total number of platforms. This token enables an immediate transition which switches, and commences the first MFOP for the plane, putting a token in the "MFOP m_1" place, where m_1 is the number of the first MFOP for that plane. Once this MFOP is complete, an MRP will take place, repairing the plane to a state where it is capable of carrying out the next MFOP in the sequence. A place represents the "MRP Complete" state (shown in the Figure as "MRP Over"), and once this is marked the next MFOP begins immediately. Once each platform has completed its sequence of MFOPs, the simulation is considered complete.

Note that each platform can carry out any number of MFOPs in its sequence, and this number need not match those of other platforms. Similarly, a platform may carry out the exact same sequence of MFOPs as any other platform. However, there is no "system of systems" approach taken with this model: two platforms performing the same mission or MFOP concurrently are assumed to be independent of each other.

6.2.5 MFOP Operation

The example in Figure 6.4 shows three different types of MFOP. In this research, an MFOP is defined as a specific sequence of missions carried out without repair. If an MFOP differs from another by only one mission, it is a different MFOP and requires a different PN. PNs for example MFOPs are shown in Figure 6.5.

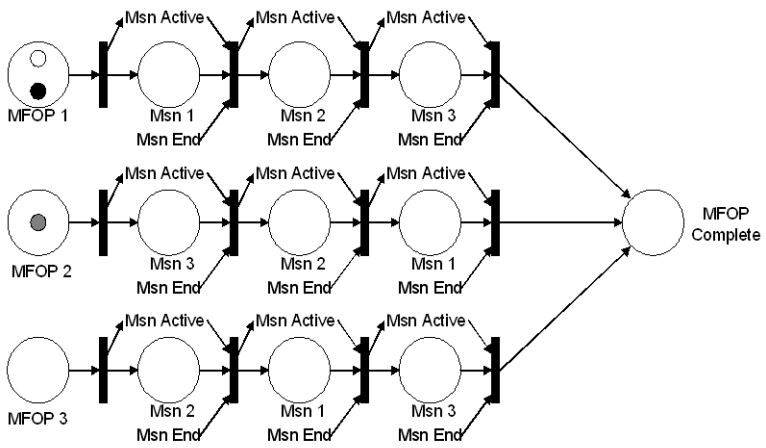


Figure 6.5 – MFOP PN Model

In Figure 6.5, a place, "MFOP m", where $m = 1,\ldots,n_{MF}$, the total number of MFOPs, represents the start of that MFOP. Considering the coloured tokens seen in Figure 6.3, Planes 1 and 2 are performing "MFOP 1", while Plane 3 is carrying out "MFOP 2". These tokens are placed here by the "Begin MF m" arcs from the immediate transitions shown in Figure 6.4. Once an MFOP commences, the coloured tokens are immediate passed to the "Msn u_1" place of the MFOP, where u_1 is the first mission of that MFOP, while another token is sent to the "Msn Active" place for that mission.

If a mission ends, its "Msn Active" and "Msn End" places will both be marked, and so an immediate transition switches in the PN shown below to start the next mission. After all missions have been completed, the MFOP is finished and a token will end up in the "MFOP Complete" place.

As mentioned in the previous section, completion of an MFOP leads to an MRP, after which the following MFOP will begin.

6.2.6 Mission Operation and Abandonment

Missions are modelled in much the same way as in the simple model. Shown in Figure 6.6, a phase to be carried out in the mission is represented by a place. Three other places represent the mission being active, abandoned and ended. Part of the platform and MFOP PNs, as explained in the two previous sections, are also shown.

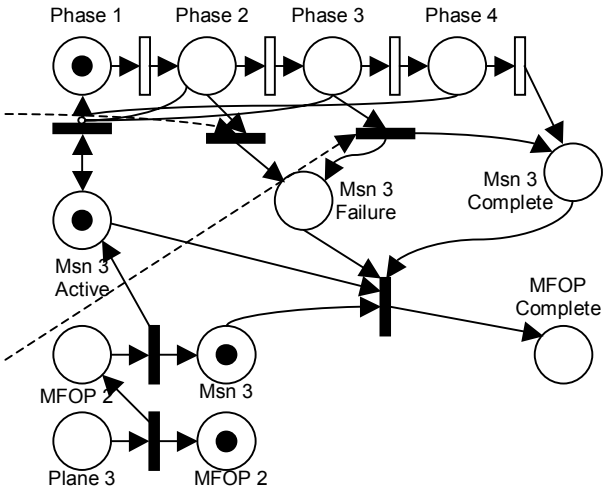


Figure 6.6 – Mission Petri Net

In Figure 6.6, Plane 3 begins its first MFOP, number 2, which then commences mission 3. The activation of mission 3 marks the place for its first phase, and after the phase length its phase transition switches to phase 2. This sequence continues until a token marks "Msn 3 Complete", and the next mission in the PN for the parent MFOP can begin, as shown in Figure 6.5. Note that it is possible for a phase to be repeated within the same mission.

If, during any phase, that phase's failure place is marked, this represents mission failure. The combination of the phase operation and phase failure places being marked will result in the mission failure and mission complete places being marked. This, in turn, removes the token from the MFOP PN and places it directly in the "MFOP Complete" place, so a Maintenance Recovery Period can begin immediately.

Because the failure of a mission in this way leads to MFOP completion and performance of an MRP, this failure reflects the abandonment of a mission. This "soft" MFOP failure allows the platform to perform future missions and stands in contrast to total loss, which is explained in the next section.

6.2.7 Total Loss of Platform

It is possible that a platform may suffer a catastrophic failure, where it is totally lost. Examples of this situation include a flight into terrain, explosion, abandonment in enemy territory, and so on. For the purposes of this model, any catastrophic failure will cause all activity for that platform to stop: component and system failures will no longer be possible, and no further missions, MFOPs or MRPs can be carried out. Figure 6.7 shows how this is modelled with Petri Nets.

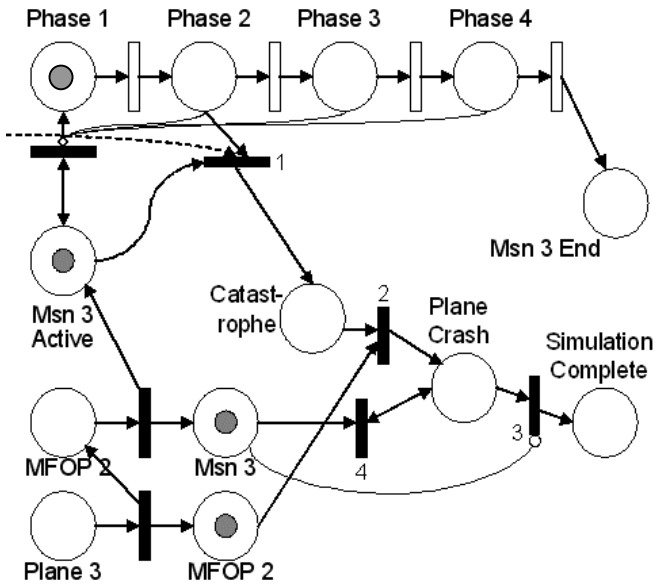


Figure 6.7 – Platform Failure Model

In Figure 6.7, part of the PN for an example platform is shown. Plane 3 performs MFOP 2 first, and so tokens are passed to both "MFOP 2" places from the initial "Plane 3" place. Similarly, Mission 3 is the first task in MFOP 2, so tokens are put into the "Msn 3" and the "Msn 3 Active" places. This commences Phase 1, which finishes after a time and begins phase 2.

If, during phase 2, a catastrophic failure arises and the top place of the Phase 2 PN becomes marked, immediate transition 1 removes the token from the Phase 2 and Msn 3 Active places. A token is placed into the "Catastrophe" place, taking the token from the "MFOP 2" place through immediate transition 2, and thus marking the "Plane Crash" place. This place removes any mission tokens from the formerly active MFOP through immediate transition 4, after which immediate transition 3 switches, completing the simulation for that platform.

There are two different ways, therefore, that a mission and an MFOP can be considered to have failed: abandonment, or total loss. Hence, this requires there to be two sets of failure logic for each phase.

6.2.8 MFOP Abandonment

Upon the abandonment of a mission as explained in section 6.2.6, the platform would return to base for repair. The phase, mission and MFOP would be considered failed, and the normal sequence of missions as modelled in Figure 6.5 would not take place. Thus, a model of this failure process is required, as shown in Figure 6.8.

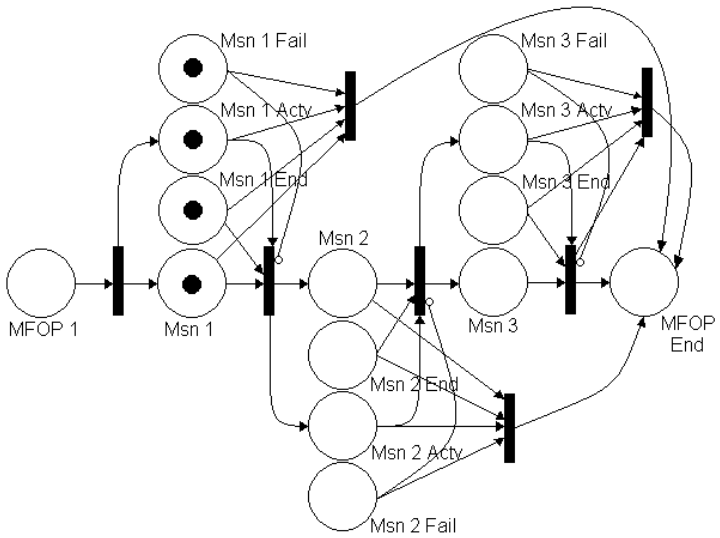


Figure 6.8 – MFOP Failure Model

In Figure 6.8, the mission profile of MFOP 1 is shown, as in Figure 6.5, running along the centre. Also shown here are Mission Failure, Mission Active and Mission

End places for each mission. If a mission is failed such that the platform is still capable of returning to base, a token will be in the "Msn *u* Fail" place. In Figure 6.8, the scenario is that the abandoned mission (Mission 1) has ended. In this case, the "Msn 1 Fail" place inhibits the transition which would typically commence the next mission. Instead, an immediate failure transition deactivates the mission, and places the coloured token in the "MFOP End" place. This allows repair to commence, after which the next MFOP will take place.

6.2.9 Mission Abandonment to Phase

Section 6.2.6 describes the modelling of the abandonment of missions. The modelling method requires the mission to immediately end, cancelling the MFOP and immediately beginning an MRP. Particularly for the case where a platform is performing a mission far from base, it is unrealistic to expect the mission to end immediately once the command to abandon it has been given. In this case, then, it is of use to allow a mission to abandon to a sequence of phases, after which the mission ends and an MRP can begin. Figure 6.9 demonstrates this idea.

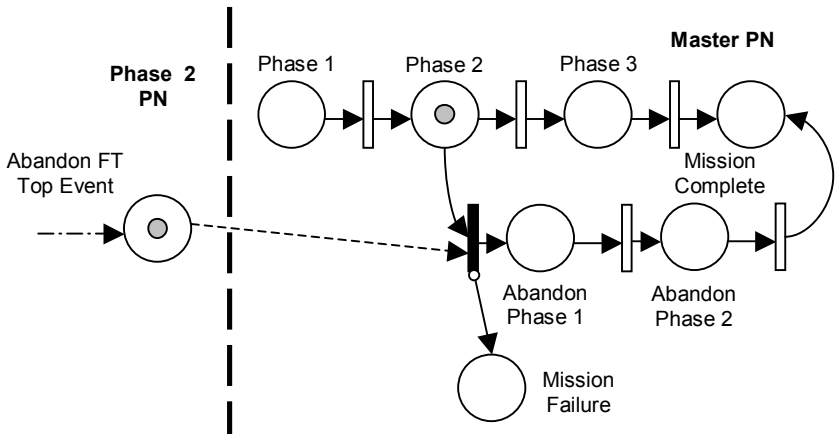


Figure 6.9 – Mission Abandonment to Phase

In the Figure, the top event place for the abandonment fault tree PN in phase 2 is marked. As the mission is currently in phase 2, the mission is abandoned and, instead of the token going directly to the "Mission Complete" place, as it does in Figure 6.6, a token is placed in the "Abandon Phase 1" place. This phase will take

time to complete, after which "Abandon Phase 2" takes place. Only after these two phases have finished will the mission have finished and an MRP can begin.

Of note is a small complication that arises from this provision of abandoning to phases: it would theoretically be possible for an abandonment phase to itself abandon to another phase, *ad infinitum*. Because of this, while it is possible for catastrophic failures to still occur during an abandonment phase, it has been made a condition of abandonment that the "Mission failure" place, when marked, inhibits all abandonment transitions. Thus, when a mission has been abandoned in a certain way, it cannot then be abandoned to a different phase: the sequence must complete as given.

6.2.10 MRP Modelling

If a platform is set to undergo more than one MFOP, there must be a Maintenance Recovery Period between the two, where the platform is brought back up to the point where it can perform the next MFOP. One way of modelling the progression from Ending MFOP → Repair → Next MFOP is shown in Figure 6.10.

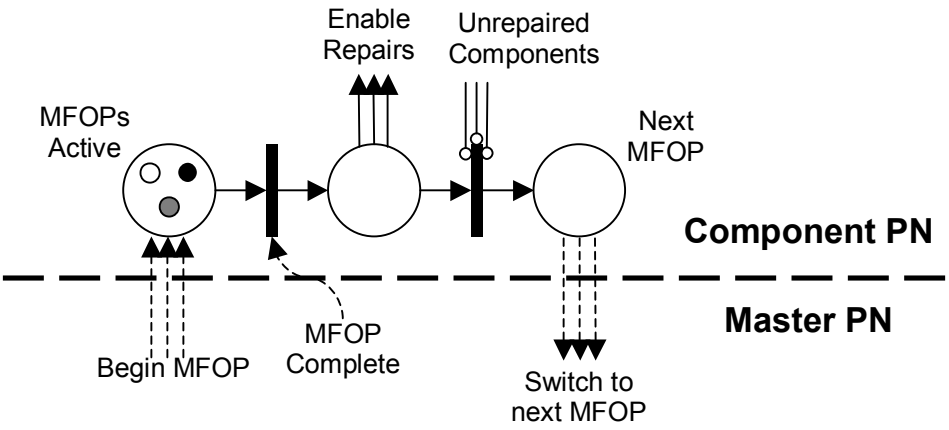


Figure 6.10 – MFOP-MRP Sequence Model

In the Figure, whenever an MFOP commences in the Fleet PN (Figure 6.4), a token of the relevant colour is placed in the "MFOPs Active" place in the Component PN. Nothing will then happen in this part of the overall PN until the MFOP is

completed, and a coloured token is present in the "MFOP Complete" place, as shown in Figure 6.5. This enables an immediate transition which switches and marks the "MRP Active" place. When in this state, the platform is capable of undergoing whatever repairs are required, and will only switch out of the state when the necessary components for the next MFOP have been repaired. These components' failure places will, if marked, inhibit the exit transition from the MRP Active state. When this repair is completed, the immediate transition to which the "MRP Active" place inputs deposits a token in the "Next MFOP" place, which corresponds to the "MRP Over" place shown in Figure 6.4, and allows the immediate transitions in that Figure to switch, commencing the next MFOP.

The "MRP Active" place is one of the most important in the whole net: it is through this one place that many of the processes explained below, such as inspections and replacements, are possible.

6.2.11 General Failure

If it is possible for a platform to be failed in the same way during any of its phases, it would be inefficient to have each phase fault tree repeat the same modes and events of failure. Instead, therefore, it is possible to define a *general failure fault tree* which, if the top event for either catastrophic failure or mission abandonment occurs, the mission and MFOP will be abandoned.

The fault tree PNs for general abandonment and total loss are contained in their own PPN, structured in the same way as the other PPNs. The main difference between the other PPNs and the general PPN is that the occurrence of either of the top events in the latter net can cause a mission failure regardless of the current phase. Figure 6.11 and Figure 6.12 show this.

In Figure 6.11, the General PPN failure top event place is marked with a token, indicating the platform's total loss. This place acts as an input to four different failure transitions in the Master PN shown on the right side of the diagram. Whichever phase place is marked in the active mission and MFOP ("Phase 1" and

"Msn 3" in this instance) will have its token removed by the corresponding failure transition which puts a token in the "catastrophe" place.

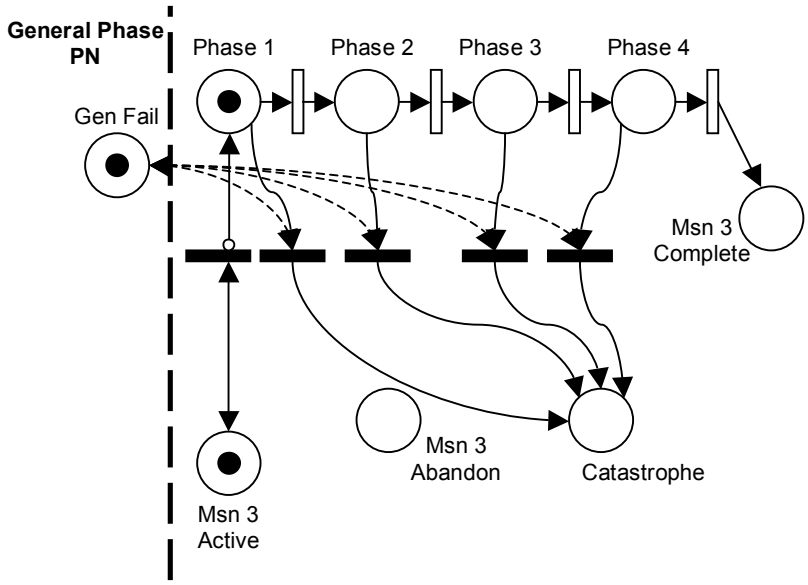


Figure 6.11 – General Platform Total Loss

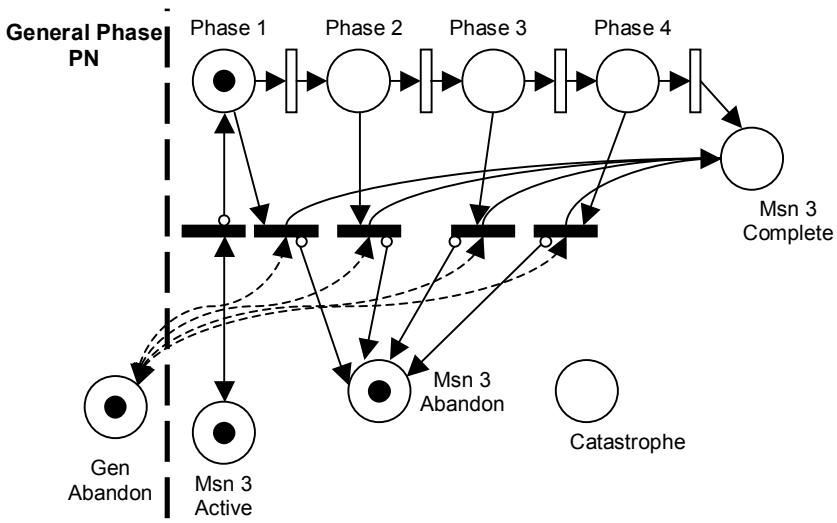


Figure 6.12 – General Mission Abandonment

The procedure for abandoning a mission works in a similar way, as shown in Figure 6.12. The marking of the General PPN Abandonment top event place will cause the relevant phase place to automatically lose its token via a failure transition, with the mission abandonment place being marked instead. If the phase is set to abandon to another phase, as explained in section 6.2.9, then this phase will begin (note that any abandonment transition for this new phase will be inhibited). Once the sequence of

abandonment phases has completed, or if there is no sequence, the "mission complete" place will be marked, and an MRP will begin.

Due to the possibility of any or all of general phase failure/abandonment and normal phase failure/abandonment occurring at the same time, a hierarchy has been created to prevent more than one of these affecting a mission. This works as follows:

1. General phase failure
2. Individual phase failure
3. General and individual phase abandonment

As such, the general failure place is set to inhibit the transitions relating to general abandonment and individual fail & abandonment. Individual phase failure will inhibit the abandonment transitions.

6.2.12 Phase Selection

A phased mission does not necessarily consist of one linear sequence of phases. It is possible that different events, such as the failure of one or more of the objectives of a mission (for example, a missile missing a target), or component or system failures, may lead to one phase from an array of choices being selected to be performed next. The nature of the drivers behind the selection can be broken down into two areas: those prompted by events considered within the model, and those which are outside its scope. The former use these events to enable or inhibit progression to a different phase, while the latter use probability transitions, explained in section 6.2.2, to randomly select (with a given weighting towards each option) which phase is carried out next.

6.2.12.1 Event-driven Phase Selection

Component or system failures can affect the choice of the phase to begin next. As an example, consider a mission with an attack phase where a missile must be launched. The failure of either the launch system or the missile will prevent this

phase from being successful, and so an alternative phase may be performed instead, such as sending the platform to a secondary objective. This can be modelled using the PN system as shown in Figure 6.13.

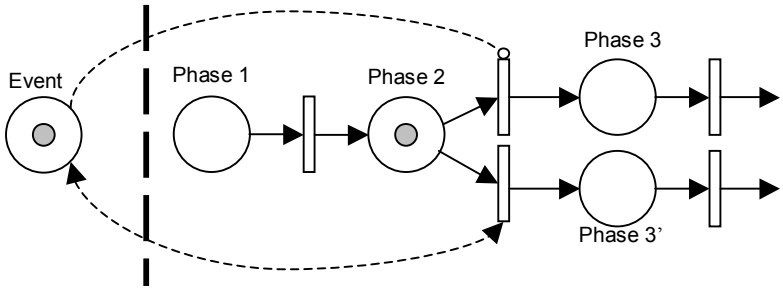


Figure 6.13 – Event-driven Phase Selection

In this example, phase 2 in the master PN has two output transitions. An event (be it basic, intermediate or top) controls which of these is enabled at any one time, depending upon whether it is marked or not. If the component or system is operational, it will not be marked, and so the mission will progress normally from phase 2 to phase 3. If a failure has occurred and the event place is marked, the normal phase transition is inhibited, and instead the one shown below it in Figure 6.13 is enabled instead. As a result, phase 2 will enter phase 3' once it is finished.

6.2.12.2 Probability-driven Phase Selection

Certain factors (such as the weather on a given day or the likelihood of a successful attack) are not considered directly within the scope of the model. These factors may still affect the behaviour of a mission, however, and as such a method to decide the following phase according to user-defined probabilities has been provided. This method uses the probability transitions introduced in Section 6.2.2, with the probabilities on the arcs corresponding to those of entering each phase.

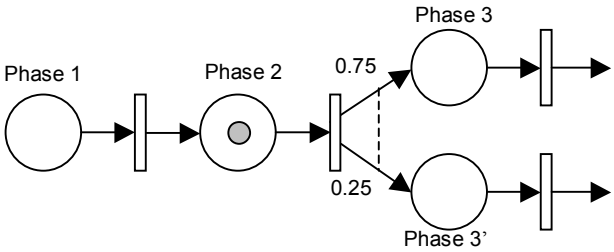


Figure 6.14 – Probability-driven Phase Selection

Figure 6.14 shows a probabilistic phase transition taking its input from the phase 2 place. In 75% of cases, the output selected will be phase 3, while in 25% of cases it will be phase 3'. This probability may represent anything uncontrollable by the operator, such as requiring a prevalent weather condition for phase 3, or the presence of civilians in 25% of cases requiring an alternate mission plan.

6.2.13 Phase Insertion

It is possible that, due to external conditions, an emergency phase may need to be performed immediately, after which the original phase resumes where it was interrupted. An example to consider is where a plane must refuel at an intermediate base or in-flight, due to spending longer than planned in the air.

Phase insertion is another complexity of phased missions modelling which can be simulated in Petri net form. A typical way to mimic the insertion is shown in Figure 6.15 below.

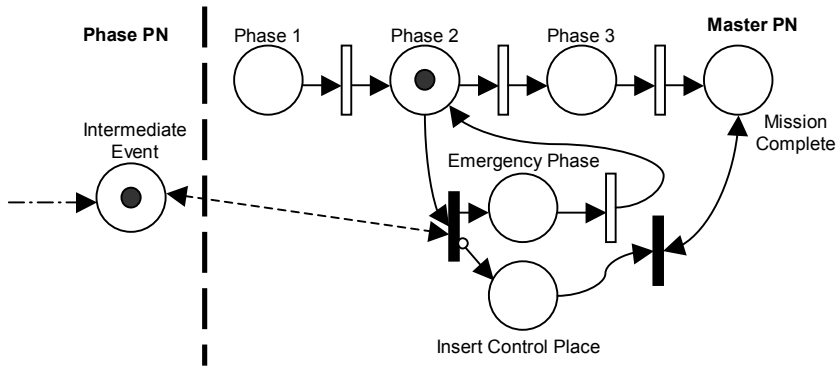


Figure 6.15 – Phase Insertion

In the Figure, an intermediate event has occurred during Phase 2. This intermediate event prompts an immediate transition to remove the token from the Phase 2 place and put one in the Emergency Phase place and one in the "Insert Control Place". The latter of these inhibits the Insert transition from switching more than once in a given mission. Once the Emergency Phase has completed, its phase transition switches and the token is placed back in the Phase 2 place. The timed transition for Phase 2 resumes its previous switch time (increased to account for the interruption)

and the mission can continue. Once the mission has ended, an immediate transition removes the token from the Insert Control place.

As the phase interruption can compete with phase failure and abandonment in removing the token from the phase place, it has also been given a place in the hierarchy mentioned in section 6.2.11, at the bottom. Therefore, the phase insertion transition will be inhibited by all phase failure and abandonment top event places. Also, in order to ensure that phase insertion cannot occur in abandonment phases, every abandonment transition will, in addition to marking the abandonment phase and the "Mission failed" places, put a token in all of the Insert Control places for that mission.

The improvements to the model shown in section 6.2 apply primarily to the nets in the Master PN, which control the active phase, mission and MFOP for each platform. The following section covers those changes which allow for more complex modelling of aspects regarding reliability and platform performance. These include the MFOP enablers mentioned in section 3.5.1.

6.3 Reliability & Maintainability and MFOP Enablers

The discussion of MFOPs so far in this chapter has focused primarily on their relation to the concept of phased missions. However, as outlined in Section 3.5, suggestions have been published as to how an MFOP of a useful duration is achievable. This section covers the inclusion of these aspects in the Petri net model.

6.3.1 Inherent Reliability

The consideration of inherent reliability in the MFOP philosophy provides for a greater understanding of the failure modes and distributions of the components of the platform. Improvements to the component reliabilities can be made and will thus affect the reliability of the platform. However, this improvement does not need to be

modelled using Petri nets – rather, more reliable and accurate failure time distributions can be provided for input to the model once this is known.

6.3.2 Discrete Events

It is possible to describe an event that does not have a "failure" like other basic events, but instead is the occurrence or non-occurrence of some outcome with a given probability at a given time, in response to a trigger. These types of events are here referred to as "discrete" events (DEs). An example of this is a simple human-error consideration, such as the maintenance crew of an aircraft not replacing the fuel cap before flight. This is not an event that can be described as a failure over time, as it cannot happen mid-flight. Instead, it is a DE that either does or does not occur after a prompting event (in this case, refuelling before a mission begins).

Discrete events use immediate probability transitions to provide an output to either an "event occurs" or an "event does not occur" place. The full net for a DE is shown in Figure 6.16.

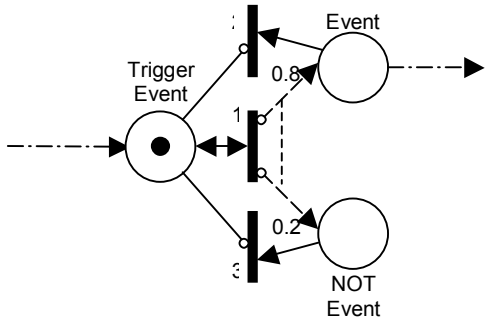


Figure 6.16 – Discrete Event Model

The trigger for a DE can be anything considered within the scope of the model: an MRP, the start of a phase, mission or MFOP, a component failure and so on. If the trigger event occurs and its associated place becomes marked, the place causes transition 1 in Figure 6.16 to switch. This will mark either of the "Event" or "NOT Event" places, depending on a randomly sampled number between 0 and 1, and the arc probabilities. Transition 1 is disabled by the marking of the output places, while transitions 2 and 3 will remain inhibited by the trigger event, until such time as the

trigger event place becomes unmarked. When the input event is no longer occurring, either transition 2 or 3 (depending on which output was originally selected) will switch, removing the token from the output place. Discrete event 'Yes' places are able to act as inputs to phase fault trees, reproducing responses to failures or event occurrences.

Phases and MFOPs have multiple places which represent their operation. If a discrete event takes an input from a phase or an MFOP, a "summary" place is created, which takes as inputs each instance of the occurrence of the relevant event in an OR logic. When the summary place is marked, the discrete event makes its decision then.

6.3.3 Component Replacement

The simple MFOP model considered that the replacement of a component could only occur if the item was failed. The repair process moved the component directly from the failed state to the working state. It is possible, however, that a component which is not failed needs to be replaced, for instance due to having a scheduled life after which it is considered unsafe to allow the component to continue operating.

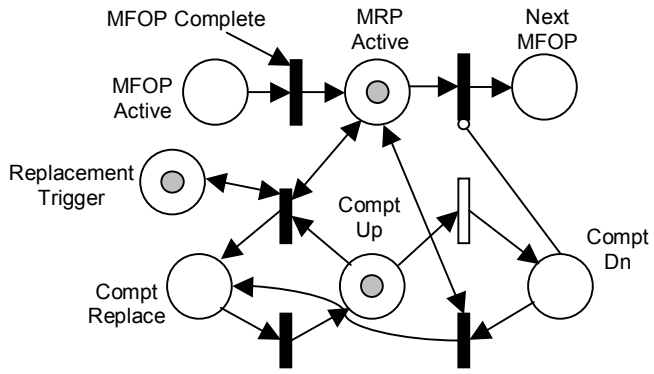


Figure 6.17 – Component Replacement

Figure 6.17 shows a component PN which is connected to the MFOP-MRP-MFOP places as described in Figure 6.10. If a component fails after a given time, the token shown in the "Compt Up" place will move to the "Compt Dn" place. During an MRP, previously the token would have returned directly to the "Compt Up" place.

In the model shown in Figure 6.17, however, the token instead moves to the "Compt Replace" place. A token is then replaced in the "Compt Up" place where another failure time can be sampled randomly.

The occurrence of an event may mean that the component must be replaced during the next MRP, regardless of whether it is operating or not. Figure 6.17 shows the "Compt Up" place providing an output to a transition on its top-left, which also takes inputs from the replacement trigger and MRP Active places. This transition then passes an output to the "Compt Replace" place. The presence of a token in the "Compt Replace" place resets the time of the component failure transition. The token is then replaced in the "Compt Up" place and a new time to failure generated.

This component replacement model is used extensively as a part of other modelling methods. One of these is the concept of Line Replaceable Items, which are explained in the next section.

6.3.4 Line Replaceable Items

Components are rarely removed as individual items. For all the typical inspections, repairs or replacements that need to be performed during maintenance, this would be a very costly procedure, in terms of time and money. Instead, to speed this process up, they are often located together in batches in physical items which can be removed or replaced as a whole. These groups are known as Line Replaceable Items or LRIs. A LRI may contain other LRIs, a full system or several independent components. A LRI containing other LRIs is typically referred to as the *parent*, while those it contains are *child* LRIs.

A LRI, as far as this model is concerned, is a simple, descriptive way of allowing multiple component replacements in response to a single stimulus, regardless of whether those components are currently in the working or failed states. The children for each LRI are specified, whether these are other LRIs, components or systems. Also given is the logic which dictates how many of the children must have failed in order for the parent to be considered worth replacing. This will either be all of the

children (in which case the input logic is essentially an AND gate), any one of the children (an OR gate), or a certain number (a x-out-of-y vote gate).

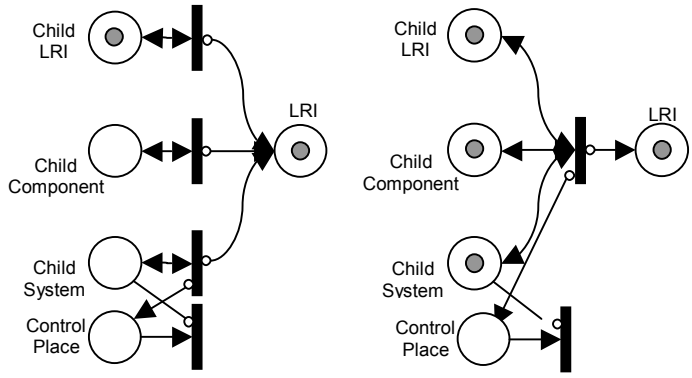


Figure 6.18 – OR and AND logic inputs to LRI place

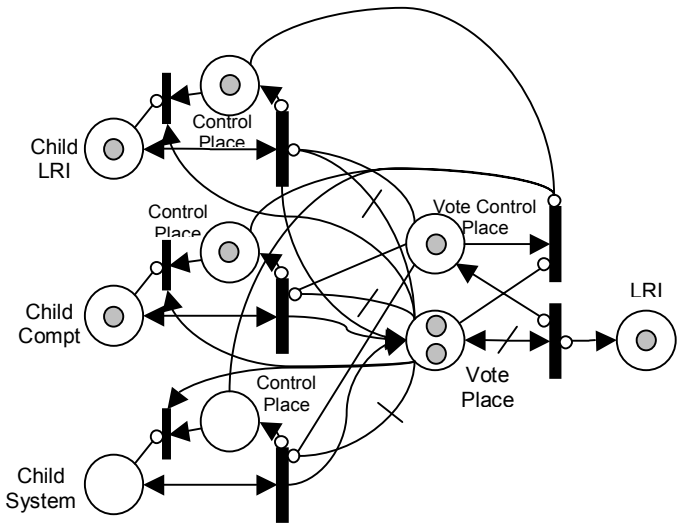


Figure 6.19 – 2-out-of-3 Voting gate input to LRI place

Figure 6.18 shows the first two cases mentioned, with the left-hand PN being that where any one input needs to have failed in order to replace the LRI. The right-hand PN is for the case where all of the inputs need to have failed. The three places which input to the LRI place are a child LRI replace place, a component failed place and a system failed place. If a system failed place provides input to the LRI, then the transition it inputs through must also output to a control place, which prevents the system failure place from causing multiple replacements of the LRI in an MRP. This control place is cleared whenever the system failed place is cleared.

Figure 6.19 shows a much more complicated PN using the same three places for inputs as in Figure 6.18. This is an example net where any two of the three inputs need to have failed for the LRI to be replaced. Each input place enables a transition

which places a token in a “vote place”, as well as in an input control place, which prevents the transition from switching more than once for a given failure. The vote place outputs to a transition which allows it to mark the LRI place. Due to the arc-multiplicity of two, the vote place can only enable this transition when there are two tokens within the vote place, which is when enough inputs are marked. When this condition is satisfied, the vote place inhibits all input transitions, and switches its output transition. This remarks itself, and marks the LRI place and a “vote control place”. The vote control place prevents the vote place from repeatedly enabling and switching its output transition.

If an input place becomes unmarked, its control place will output its token to a newly enabled transition. The vote place also outputs to this transition, and so will reduce its token count by one whenever the transition switches. When all input places are no longer marked (which will only happen during an MRP), the vote place will be empty, and the vote control place will output to a transition which removes its token as well. Thus, the net is cleared before a new MFOP begins.

The nets shown thus far merely explain how a LRI place can come to be marked. Once it is marked, it must play a role in replacing each component contained within it. Consider the PN shown in Figure 6.20. There are three components shown: A, B and C, each of which can be replaced as demonstrated in the previous section. Also shown are places representing six LRIs. LRI 1 is the parent of LRIs 2 and 3. LRI 2 is the parent of LRIs 4, 5 and 6. LRIs 3, 4 and 5 are not considered in this example other than to demonstrate the nature of the LRI parent-child hierarchy.

At some point during an MFOP, LRI 2 is marked for replacement during the following MRP, due either to failures of its child items or to having a scheduled replacement time. A LRI, when marked for replacement, will also mark all of its child LRIs for replacement. This therefore creates tokens in the places for LRIs 4, 5 and 6, the latter of which contains A, B and C. At the time of the MRP, A and C are operational, while B has failed. The “LRI 6” place acts as the replacement trigger for each of the components, as shown in Figure 6.17. Its being marked causes the tokens to be removed from A and C’s up places, and B’s down place, and mark the

respective replacement places. These then re-mark the component-up places, following which a new time to fail is generated for each component.

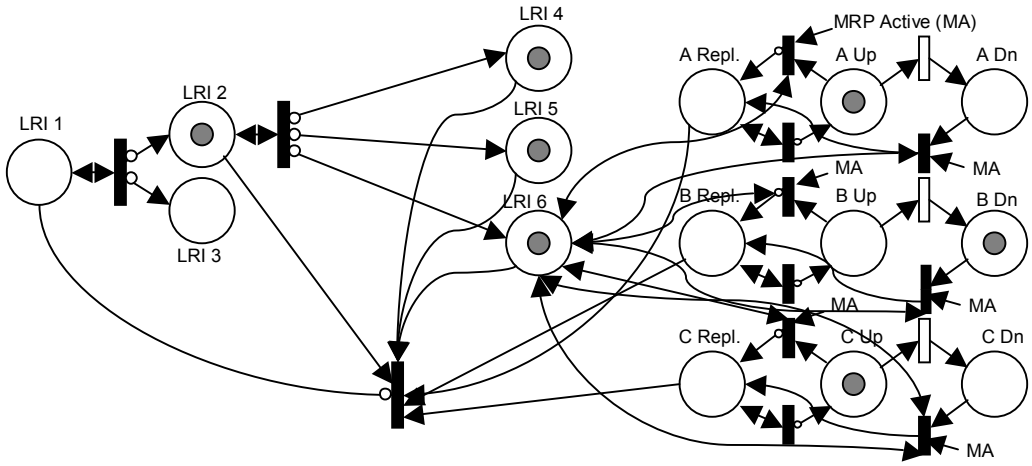


Figure 6.20 – LRI Replacement

Each LRI has a transition which acts to remove the tokens from the places representing that LRI, all its child LRIs and the replacement places of all the components contained in itself and its children. If a LRI has a parent, the parent LRI place will inhibit the clearing transition for the child LRI, as the parent will have its own clearing transition which must switch instead. The transition that performs this function for LRI 2 is shown at the bottom-centre of Figure 6.20. Once the LRI has been replaced and all the relevant "Compt Replace" places are marked, this transition switches, removing the tokens from each input.

Figure 6.10 shows each "component failed" place as inhibiting the transition which allows progression from an MRP to the next MFOP. If a component is fixed by a LRI, it will no longer directly inhibit this transition; the LRI place does this instead.

If a system is specified to be an input to a LRI, this means nothing other than all the components which it contains are all to be replaced. However, the system failed place becomes an input to the LRI, so any failure logic will focus only on the system and any other inputs, not on the individual components within the system.

Note that a component or system cannot be part of more than one LRI, excepting that LRI's set of parents. This is because a LRI is a physical location of the component, rather than an arbitrary grouping.

6.3.5 Scheduled Life Replacements

Certain items within the platform may be important, and yet cannot be supported within the platform architecture by redundancies or other means. If these items fail, they could put the platform into a critical state, and so they are subject to scheduled life replacements (SLRs). This means that, once a certain time has expired after the item was installed, the item will be replaced regardless of whether it has failed or not. While this is typically an expensive maintenance strategy in terms of throughput of components, it may prove rewarding in the improvement in overall platform reliability.

There are two types of item to which an SLR can be applied: LRIs and individual components. The left-hand-side of Figure 6.21 shows the PN for an SLR as used with a LRI. The SLR needs to mark the main LRI place after a given length of time. To achieve this, whenever any MFOP is defined as being "active", it will mark a SLR control place, unless the LRI is already set to be replaced. If it is not, the marked control place will enable a timed transition set to switch after the length of time defined by the user. When this switches, the LRI is marked to be replaced during the next MRP. A third transition forces the SLR control place to be emptied whenever the LRI place is marked in a different way.

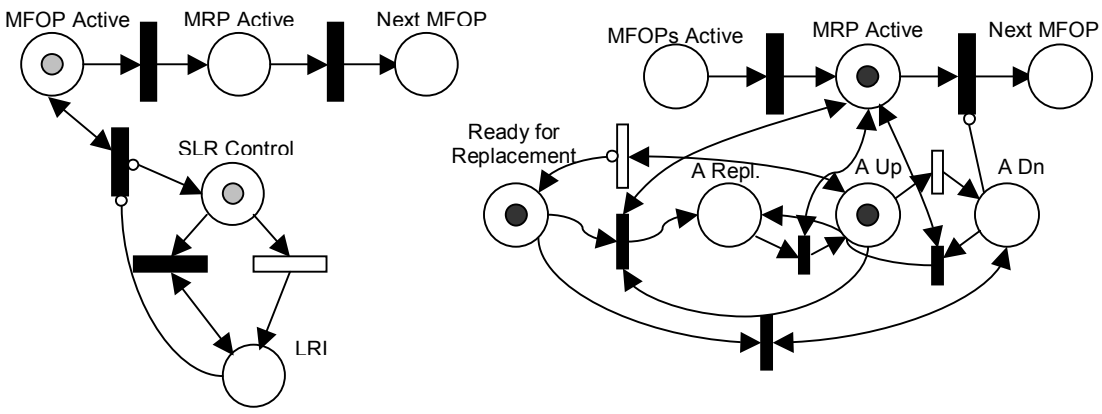


Figure 6.21 – SLRs for an LRI and a Component

Figure 6.21 also shows the net modelling an SLR for a component. For the LRI, an SLR control place inputted to the timed transition, but for the component A, its "up" place takes this role. The "Ready for Replacement" place shown in the Figure is similar in purpose to the main LRI place. While the "A Up" place is marked, a timed transition is enabled which, once the time has expired, will place a token in the "ready for replacement" place, without removing it from the up place. If the component were to fail after this time, the token in the "A Dn" place would cause the ready for replacement place to be emptied, through the transition at the bottom of the Figure, and the component would be repaired in the next MRP as normal.

If, however, the component remains in operation until the next MRP, then the marking of the MRP Active, Ready For Replacement and A Up places causes an immediate transition to switch which empties the latter two places and marks the "A Repl." place. This resets the time to fail for A, and when the up place is marked again, a new time to fail is generated.

If a component is specified to have a scheduled life replacement, and the component is a specific member of a LRI (that is, not as part of a parent system), then a few alterations must be made to the nets shown in Figure 6.20 and the right-hand-side of Figure 6.21 to allow the two to work together. These alterations are shown in Figure 6.22.

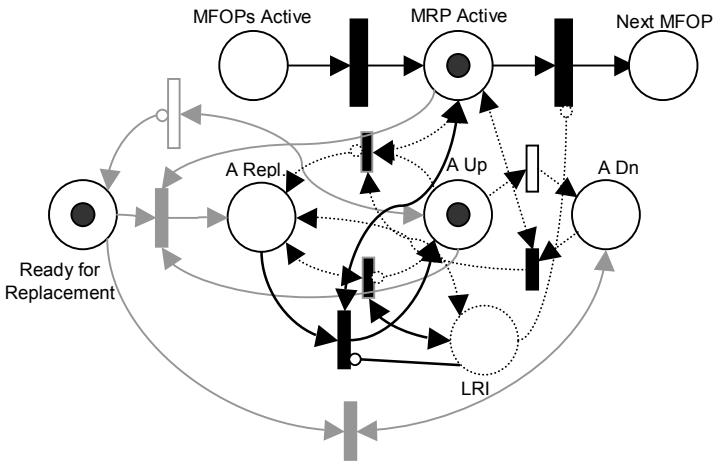


Figure 6.22 – Combining Component LRI and SLR

Figure 6.22 shows standard component and LRI arcs as dotted, and SLR arcs and transitions as grey. A new transition is created, inhibited by the LRI place and enabled by the MRP active place, which allows the replacement place to input back to the up place. Thus, there would be two transitions returning tokens performing this role. The one which was created explicitly for the LRI would be newly enabled by the LRI place, always allowing the replacement place to remark the up place regardless of whether it is the LRI or the SLR which has prompted the replacement.

6.3.6 Diagnostics

The purpose of a diagnostics system is to detect the occurrence and location of failures, faults or performance deterioration within the overall system. This can either be performed in real-time through the use of sensors, or during maintenance periods through inspections. These play a role in the ability to predict future failures, either of individual components, or of phases and missions, as well as affecting redundancies. The ways in which these two failure detection methods have been implemented in the PN model are outlined below.

6.3.6.1 Sensors

In real situations, sensors monitor a feature of a system, such as flow rate through a pipe, or potential difference between two points. For the purposes of the model, however, a sensor is a single component which, if working, will detect a failure of

another component or a system. The manner in which the sensor or sensor system decides that the failure has occurred is irrelevant here; only the ability to detect it is important.

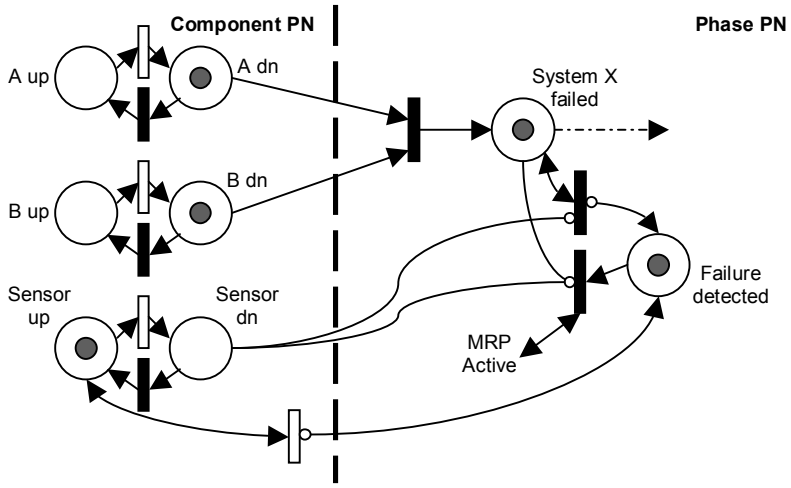


Figure 6.23 – Sensor Detecting Failure of a System

In Figure 6.23, system X consists of two components, A and B, such that they both need to fail for X to fail. A third component, the sensor, will detect the failure of X only if it itself is working. This is handled in PN form by having an immediate transition inputting from X alone (and so being enabled whenever X is marked), outputting to the "Failure detected" place and back to "System X Failed", and being inhibited by the "Sensor dn" place. In the marking in Figure 6.23, the currently operating sensor allows the marking of the detection place.

A common problem with failure detection systems is a *spurious trip* or *No Fault Found* (*NFF*). Occasionally, a sensor will consider the item it is monitoring to have failed when no actual failure has occurred. Where there are consequences of the reported failure, such as switching power to a backup system, or abandoning a mission to return to base, this can be costly. This has been modelled in Figure 6.23 by attaching the "Sensor up" place to a second timed transition and inputting to the "Failure detected" place. The timed transition will sample a time randomly from the NFF rate, which describes the distribution of times at which NFF errors occur. If the sensor remains operational and the component it is sensing does not have a detected failure before this time expires, the transition will switch, reproducing a spurious trip.

When an MRP takes place, any tokens in the "Failure detected" place, regardless of being caused by a real failure or spurious trip, will be removed as long as neither the sensor nor the sensed component or system is failed.

6.3.6.2 Inspection

If a component or system's operation is not monitored by a diagnostic sensor system, it may instead be inspected at regular intervals throughout its life to detect failure or signs of wear. Within the scope of the MFOP policy, these inspections must take place within MRPs after either regular time intervals (numbers of hours), numbers of MFOPs, or in response to the detected failure of a higher system. Inspections can take place on components or systems.

The formation of a PN for each item undergoing an inspection depends on the following factors:

- Does the inspection take place after a fixed period of flying hours, a fixed number of MFOPs, or on the discovered failure of a parent system?
- Is it a component or a system?
- Is it a member of a LRI, either directly or through a parent system?

The subsections below explain the PNs for each type of case, beginning with the implementation of the inspection on the various possible items.

Component with no parent LRI

If a component which is not a member of a LRI, either directly or through a parent system, is to be inspected, then this is relatively simple to model. Figure 6.24 shows the typical PN for this case.

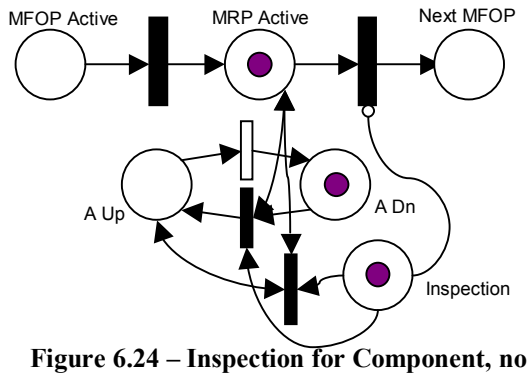


Figure 6.24 – Inspection for Component, no LRI

The Figure shows a standard component, without any replacement place. An inspection place has been also included, and when this is marked with a token, at the next MRP, one of two things will happen. If the component is in the working state, then the transition at the bottom of the PN shown will switch and clear the inspection place. If the component is failed, then the inspection will switch the repair transition and the token will return to the "A up" place.

Note that if the component has scheduled life replacements, then as explained in section 6.3.5, it will have had a replacement place created. However, the addition of the inspection place enabling the repair/replacement transition does not affect the SLR PN, or vice versa. There would usually be an inhibitor arc between "A Dn" and the MRP complete transition, as shown in Figure 6.10, but this is deleted and an arc leads instead from the inspection place. This is because a failure may be carried through an MRP if an inspection on a failed component is not yet due.

Component with parent LRI

A component which is part of a LRI will not typically be replaced individually, but instead when the overall LRI requires replacement. Section 6.3.4 explains how a net for a component LRI member is generated, and how they interact with each other. Requiring the component to undergo an inspection complicates the net further, depending on whether the component is a direct part of a LRI, or a parent system is. If the component is a direct member of a LRI, then the net as exemplified in Figure 6.25 applies.

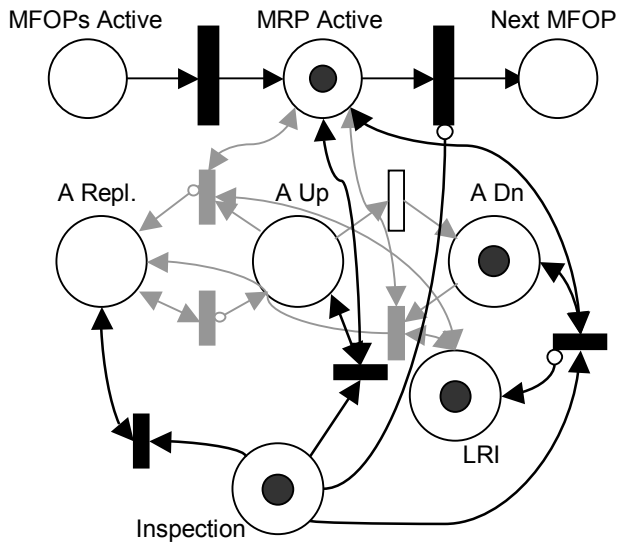


Figure 6.25 – Inspection for Component in LRI (Direct Member)

The Figure shows, in grey, the standard PN for a component LRI member. Its "down" place inputs to the LRI (whether this is via AND, OR or vote logic does not affect the inspection net), while the LRI place forces replacement when marked. The inspection place inhibits the "MRP Complete" transition as explained above, and it outputs to three different transitions. As the component is a direct member of the LRI, its input to it is based on the component having failed. However, its failure is not revealed until the inspection has taken place, and therefore an arc must lead from the inspection place to the LRI input transition used by the component.

The two other transitions to which the inspection place outputs remove its token based on the LRI having been replaced, but due to a component failure, and an inspection of the component revealing that it is still operational.

If the component has a parent system which is a direct member of a LRI, then the component will undergo replacements, but will not have an input to the LRI directly. It has been considered that if a component is specified to undergo inspections independently of a parent system, then the component must be able to be replaced by itself, in addition to being part of a LRI. The PN for this is shown in Figure 6.26.

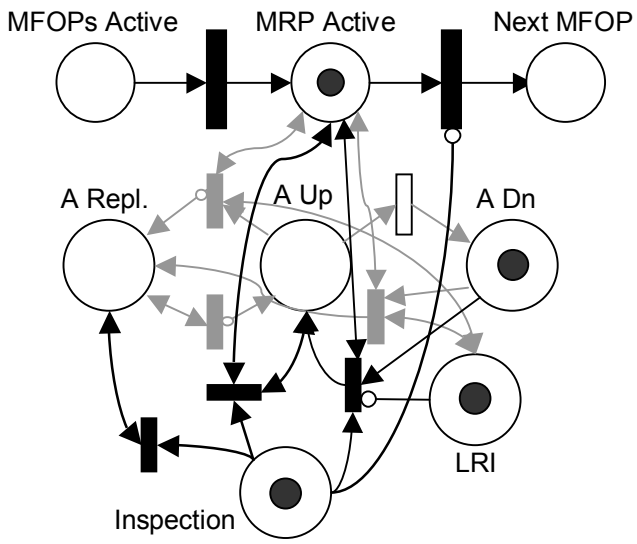


Figure 6.26 – Inspection of Component with LRI (Indirect Member)

The Figure is almost identical to Figure 6.25, but for the difference of one transition: the LRI input transition, at the right of Figure 6.25, is not shown, as the component does not directly input to the LRI. Instead, a transition which allows the token to return directly from "A Dn" to "A Up", without going through the replacement place, is shown. This is enabled by the MRP Active place, A Dn and the inspection place, and inhibited by the LRI place, so that if the LRI is due to be replaced, this is the superior condition. If an inspection during the MRP finds that the component is failed, it is returned immediately to the working state.

System with no parent LRI

Inspections of systems which are not part of a LRI do not have an immediate effect. If a failure of the system is detected, then the system as a whole cannot be simply removed and replaced, as it is not part of a LRI, and so the model assumes that the system's components are in different physical locations in the platform. Instead, the system's inspection must lead to further inspections of components within the system, as the system cannot be repaired until the components within it are repaired. The PN for this case takes this consideration into account, as shown in Figure 6.27.

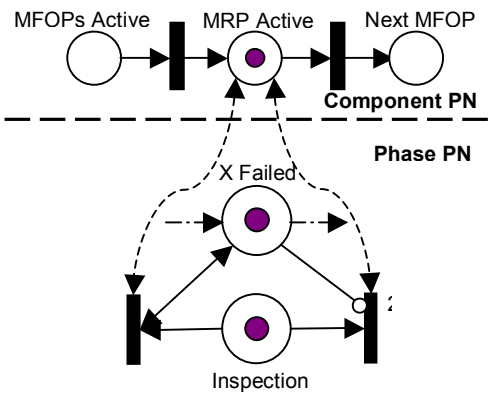


Figure 6.27 – Inspection of a System with No LRI

The Figure shows the failure place for System X, and an inspection place accompanying it. The transition labelled '1' in Figure 6.27 takes inputs from the "MRP Active" place, System X failure and the inspection place (returning tokens to the former two places). When components are inspected following the failure of a parent system, it is transition 1 which provides the token to the components' inspection places, as shown in Figure 6.32. Transition 2 removes the token from the inspection place during an MRP if the system is found to be working.

System with parent LRI

A system which is part of a LRI can be set to be inspected at regular intervals. The net which is created depends upon the logic used to decide whether the LRI should be replaced. If, as shown in Figure 6.18, the system failure place inputs to the LRI with an AND or OR logic, it will have a control place created for it. Upon being marked for inspections, the control place is converted to an inspection place. This is done as shown in Figure 6.28 below.

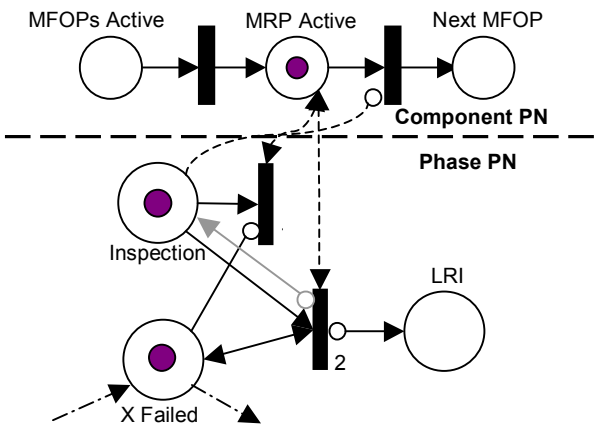


Figure 6.28 – Inspections on System with LRI (AND/OR Logic Input)

In the Figure, an input & inhibitor arc is shown greyed out between the inspection place and transition 2. This is the arc as created for the control place in Figure 6.18, which is now the inspection place. This arc is deleted from the net, and an arc leading from the inspection place to transition 2 replaces it. This new arc sets the inspection place as only allowing the "System X failed" place to input to the LRI place when the inspection is due. Transition 1, which removes the token from the inspection place when X is found not to have failed, is newly enabled by the MRP Active place. Finally, the inspection place is set to inhibit the "MRP Complete" transition in the Component PN.

If, instead, the logical input to the LRI is a *m*-out-of-*n* vote gate, then the input transition to the LRI Vote gate will already have had a control place created. This cannot be converted into an inspection place; a new place is created instead. As shown in Figure 6.29, this is set up in a very similar manner to Figure 6.28: the inspection place outputs to a transition (1) which removes the token in an MRP if the system X is found to be working. The transition (2) which inputs to the LRI vote place also now takes an input from the inspection place, and is enabled by the MRP Active place.

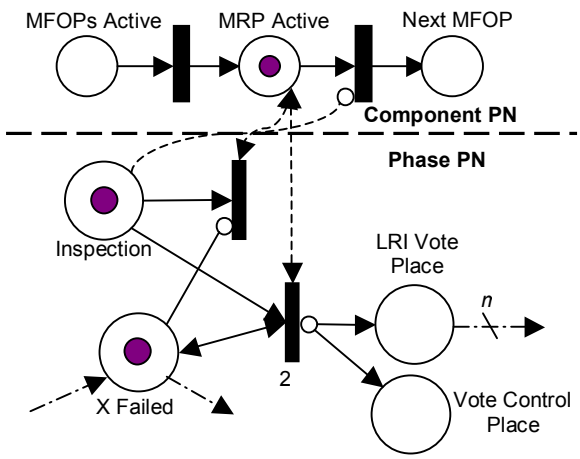


Figure 6.29 - Inspections on System with LRI (*m*-out-of-*n* Vote Logic Input)

However the inspection place is applied, there must be conditions set to allow the model to know when an inspection is required. For this, there are three inspection schedules provided:

- After a user-defined interval t_I
- After *f* MFOPs have been performed
- Once a parent system has been inspected and found to have failed

These three schedules are rendered in PN form as shown below.

Inspection after Time Interval

Figure 6.30 shows how the inspection place is marked when it is set to be performed after a given time interval. A timed transition is created, with its distribution set to be deterministic and the delay to that specified by the operator. This transition is enabled by the "MFOPs Active" place in the Component PN, so that whenever the platform is undergoing MFOPs, the transition will be enabled and waiting for the time delay to expire. The transition inputs to the Inspection place (which also inhibits it, so if an inspection is due because of another schedule, the transition is disabled). Whenever the time expires, an inspection will take place in the next MRP.

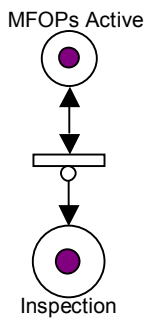


Figure 6.30 – Inspection after Time Interval

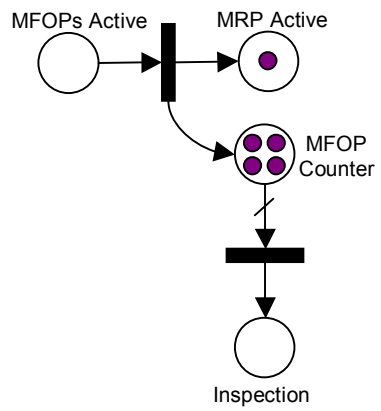


Figure 6.31 – Inspection after *f* MFOPs

Inspection after MFOPs

Figure 6.31 shows the PN method of modelling inspection after a given number, *f*, of MFOPs. Every time the "MFOP Complete" transition switches from a completed MFOP to an MRP, a token is also placed in the "MFOP Counter" place. When *f* tokens have been counted in that place, indicating that *f* MFOPs have completed, another immediate transition switches, removing those tokens and placing one in the inspection place.

Inspection after Parent System Failure

If a failure is detected or revealed in an important system, a typical procedure would be to investigate further to find the reasons for the fault. This revelation of failure could, therefore, cause a child system or component to be inspected. If an item is specified to have this type of inspection, it can be modelled in PN form. There are two cases within this scenario: where the parent system is itself inspected, and where it is not. If it is inspected, then the inspection place for the child item is connected to the transitions created for the parent system to pass on the noted failure – these are shown as transition 1 in Figure 6.27 and transition 2 in Figure 6.28 and Figure 6.29. Figure 6.32 below shows how the net is created to model this.

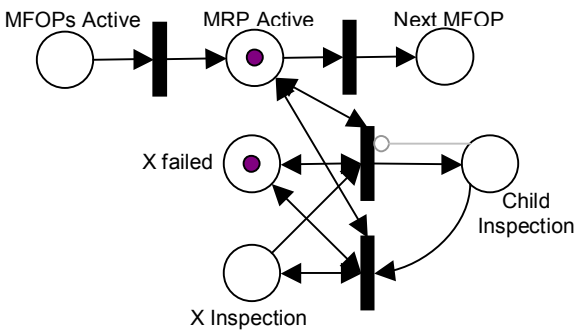


Figure 6.32 – Inspection after Parent System Failure Found

In Figure 6.32, the transition mentioned above inputs to the "Child Inspection" place when, in an MRP, an inspection of system X has found a failure. If the child system or component is a member of a LRI, it will not inhibit this transition, while it will do so if it is not a member of a LRI. As such, the inhibitor arc is shown in grey in Figure 6.32. In order to prevent the child inspection place being marked for inspection several times, another transition is created to remove any tokens in the child inspection place when a failure in system X is found during an MRP. This is shown at the bottom of the figure.

If the parent item is not inspected, then the inspection of the child depends directly upon the failure of the parent item. If the parent item has a sensor attached to it, the marking of the "failure sensed" place (as shown in Figure 6.23) is used to trigger inspection in the child; if not, the system failure place is used. The nets are created as shown in Figure 6.33.

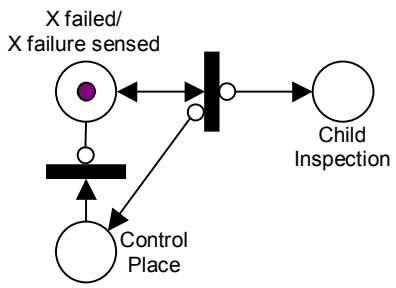


Figure 6.33 – Inspection after Parent System Failure

When the place representing either "System X failed" or "System X failure sensed" is marked, this will mark both the inspection place of the child item and a control place, provided that neither of these are already marked. When an MRP next takes

place, the child item will be inspected for failure. If the control place is marked and the failure or failure sensed place is newly unmarked, a second transition removes the token from the control place.

The inspections modelling methodology has been designed to be compatible with an item having multiple inspection schedules. If it has more than one way of establishing whether it should be inspected, there is still only one inspection place, but different ways of inputting to it are created.

Being able to detect item failures in the ways outlined above provides a great deal of control over when items can be replaced and how detected failures can impact on current and future missions. One of the major applications of these diagnostic tools is the ability to bring online redundant items. This is expanded in the following section.

6.3.7 Redundancies

Redundant items are often put in systems in order to provide an extra degree of reliability – if the primary item fails, the redundant item is brought online to continue its function. The nature of redundancies is such that a group of components within a system are considered to be either *online* or *offline*, whenever that system is in either of those states. Components which are offline are further specified as having either *cold, warm* or *hot* failures. These relate to the component's failure rate being zero, reduced, or that of the online component respectively.

Redundancies are simulated in the model in two ways. They can either be *manual*, where an operator has manual control over bringing the redundant item online; or *automatic*, where a sensor system detects the failure and acts accordingly. The former is subject to error in that the operator may fail to bring the redundant item online, while the latter will fail to occur if the sensor has failed. Failure to bring a redundant system online is the equivalent to that system having failed, and so this must also be handled by the PN model.

In the explanation below of how PNs are created to represent redundancies, a standard nomenclature is used. 'X' refers to the item which, upon failure, requires a redundant item 'Y' to become operational. Places are used in each redundancy net to represent:

- System X failure
- System X/Y operational (usually abbreviated in diagrams to "X op" or "Y op")
- Redundancy successfully/unsuccessfully switched (written in diagrams as "Yes" or "No")
- System Y not operational ("Y not op")

Each component in an item related to a redundancy will have its main failure transition (or transitions, if it has dependencies on other components, as outlined in section 5.2.1.2.1) enabled by the item's operational place. If the component has warm or hot failures when offline, another failure transition will represent this, and will be inhibited by the operational place. Figure 6.34 shows this.

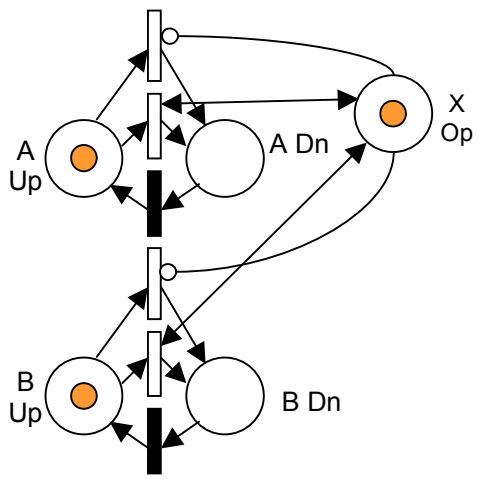


Figure 6.34 – Operational System Affecting Component Failure Transition

If the redundancy is manually switched over by an operator, then a level of human error will creep into the action. Any failure by the operator to correctly activate the redundant item will leave it unable to perform the function it is required to. Thus, the item can be considered to have failed. The process of manually switching between the primary item and the redundancy is modelled by the PN demonstrated in Figure 6.35.

The PN shows six places and nine transitions. The left-most place, “X failed” enables transition 1 when it is marked. If either of places “Yes” or “No” are not already marked, transition 1 will switch and, as it is a probability transition, will mark one of them depending on the value of P(Y) and a random number. If the token is placed in “Yes”, this means the switch to the redundant item has been successful, while “No” being marked indicates the opposite.

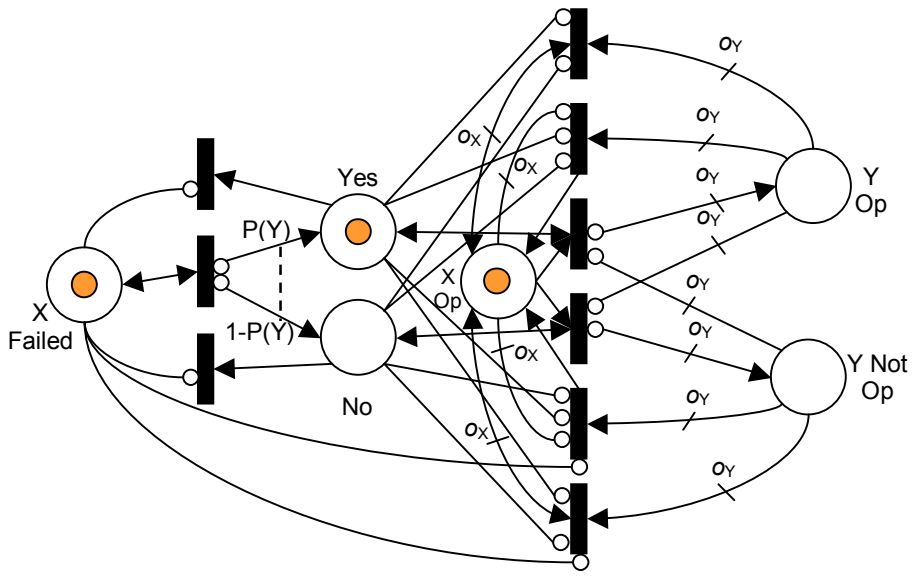


Figure 6.35 – Manual switchover from Item X to Item Y

If “Yes” is marked and system X is still operational, transition 4 will be enabled and “X Op” will pass its token to place “Y Op”, indicating that item X is now turned off and item Y is switched on. As mentioned above, this will inhibit the online failure transitions of the components in X and enable those in Y. Similarly, if “No” is marked, X Op passes a token to “Y Not Op”, and any online failure transitions in Y will not be enabled.

Of note are the arc multiplicities leading to and from places “X Op”, “Y Op” and “Y Not Op”. These are marked o_X or o_Y as appropriate, and represent the number of redundant outputs from X and Y respectively. That is, if X has one redundant output, $o_X = 1$. If Y were to have two further redundant items for its own failure, Z_1 and Z_2, $o_Y = 2$. If Y has no outputs, $o_Y = 1$. The reason for this is that each item X with o_X redundancies $Y_1, Y_2, \ldots, Y_{o_X}$ must output one token to enable each of these, and so must contain o_X tokens to switch each of them on upon its failure.

If X is repaired at any point, the "X failed" place will become unmarked and, depending on whether "Yes" or "No" was originally selected by transition 1, either of transitions 2 or 3 will be enabled, unmarking the relevant place. This, in turn, will enable either of transitions 6 or 8. If the redundancy switch was successful, transition 6 will switch, removing all tokens from Y Op and placing one in X Op. When X Op contains o_X tokens again, it inhibits these transitions. "X Failed" inhibits the transitions allowing return from "Y Not Op", so that X cannot become operational while it remains failed. Transitions 7 and 9 exist to allow Y Op to be switched off if X Op is still marked, for some reason, though this should only happen if item Y can be switched on by more than one primary item.

Similar to the PN in Figure 6.35 is that for a sensed switchover to a redundant item. In this case, however, the "sensor failed" place acts as the "No" condition, while "X Failure sensed" is the "Yes" condition. No probability transition is needed as the function of the redundancy is based purely on the sensor working properly. Figure 6.36 shows this.

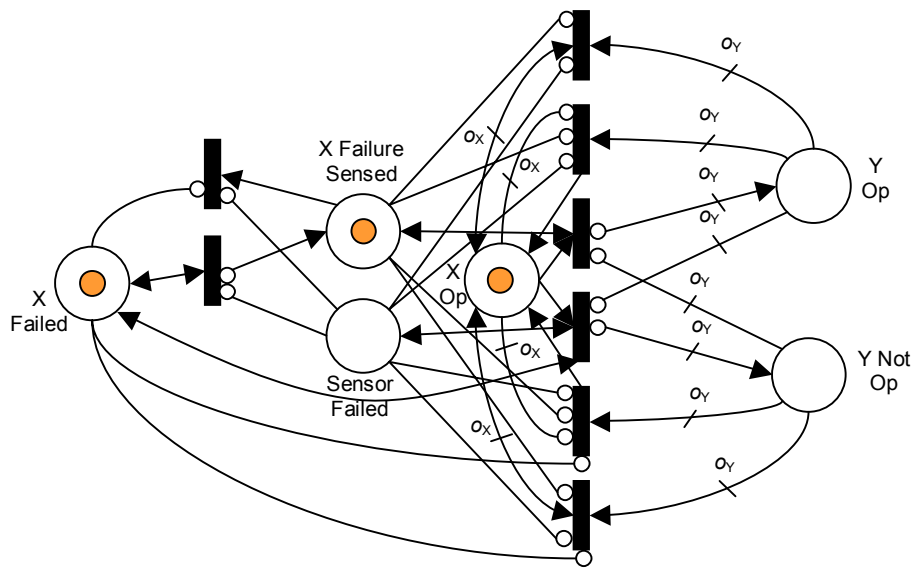


Figure 6.36 – Sensor Switchover from Item X to Item Y

In the Figure, transitions 1 and 2, and the arcs between "X failed", "X failure sensed" and "sensor failed" are exactly as shown in Figure 6.23. If the sensor fails, and then X fails, transition 4 will be enabled and X Op will be unmarked, while Y Not Op is marked with o_Y tokens. This will only be unmarked when "Sensor

failed", "X failed" and "X failure sensed" are all unmarked, with a token returning to X Op. If a failure in X is detected, accurately or spuriously, transition 3 marks Y Op and unmarks X Op.

The nets shown thus far model the ability of components to be online or offline, and the switching from an item X to an item Y, either manually or automatically. A complication of this, however, is that item Y may fail while it is operational. Similarly, Y may be repaired in an MRP, while X is not repaired. Another scenario already mentioned is that where Y is not activated, and thus is counted as having failed. In order to propagate the failure onto higher events, there must be interaction between "Y not op" and "Y failed" places.

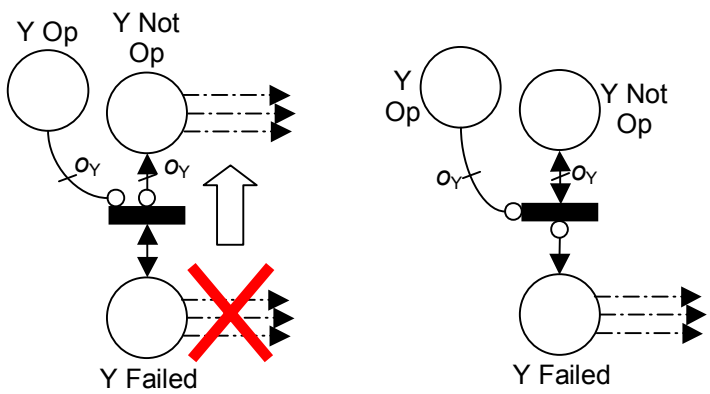


Figure 6.37 – Failure of Redundancy for Component and System

Figure 6.37 shows two similar nets which handle the issue of Y failing to be switched on. The left-hand PN is that for a component. If component Y is a redundant item, the place representing "Y failed" outputs to a transition which marks "Y not op" with o_Y tokens, as long as Y Op is not marked. The arcs from "Y failed" propagating its failure to higher events are transferred to the Y not op place instead. If Y is a system, as shown in the right-hand PN in Figure 6.37, then the marking is the other way around: Y not op marks Y failed instead. The reason for this difference is that if the component failure place were to be marked by the Y not op place (which is the simpler of the two options), there would be two tokens in the component PN (one in the "up" place, one in the "down" place), which must is a prohibited condition. This is not an issue for the system failure place, however.

The “Y op” place inhibits the transition in either net in Figure 6.37 because if it is marked and a failure occurs in Y, it must then switch to “Y not op”. Figure 6.38 shows how this simple switching is performed. If the opposite is true, and neither the “No” place nor the “Y failed” places are marked, and yet the “Y not op” place is, then it is set to switch back to Y op as shown in Figure 6.39.

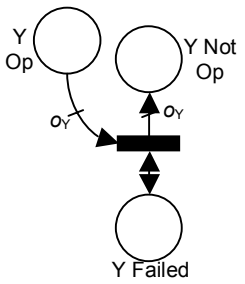


Figure 6.38 – Failure in Operational Item Y

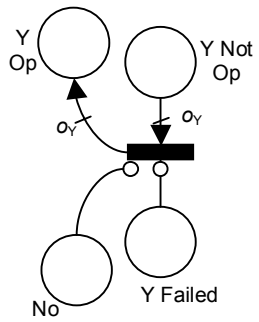


Figure 6.39 – Repair of Failed Item Y

Another issue which arises is when a system with a redundant backup system contains sub-systems which themselves have redundancies. The issue concerns the components within the systems. Typically, a primary or redundant system will enable or disable the online and offline transitions of all sub-components, even those that are contained within sub-systems, as they are all turned on or off at the same time. If, however, a sub-system has a redundancy, then its sub-components may be switched on or off either when the main system *or* the sub-system switches to the redundancy.

Consider the platform shown in Figure 6.40. It consists of two parallel systems, A1 and A2, of which A1 is the primary system and A2 is the secondary, redundant system. If A1 fails, all sub-systems and components are switched off and A2 becomes online, along with all its sub-items. A1 contains three subsystems: B1 and B2 are in parallel, with B2 acting as the redundant system for B1; C is independent. A2 contains two subsystems with no redundancies. Each subsystem contains two components in series.

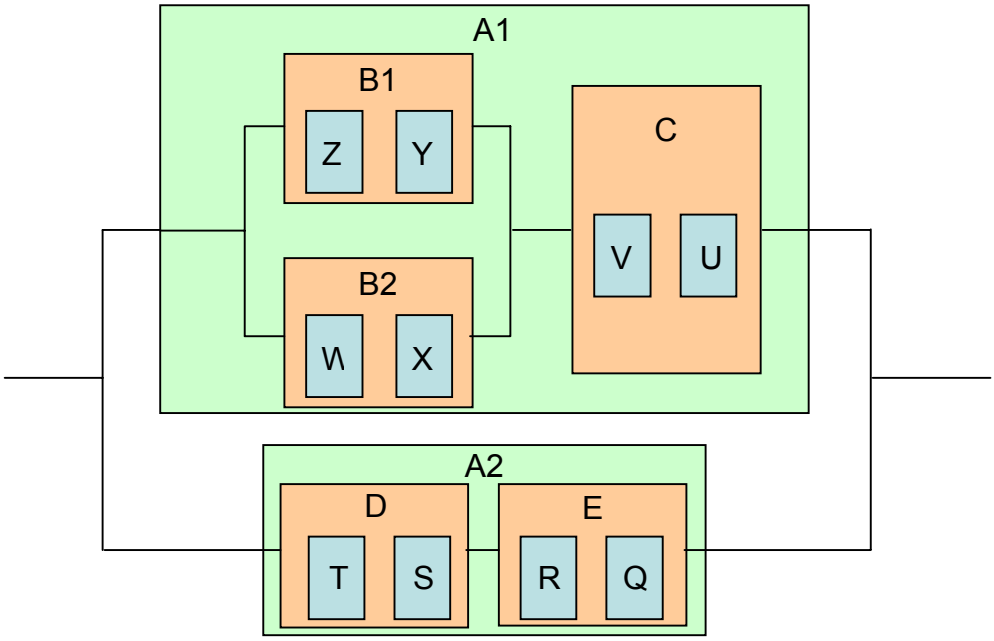


Figure 6.40 – Example Platform with Redundant Systems

At the start of a mission, A1 is online and A2 is offline, along with sub-systems D and E, and components Q, R, S and T. Within system A1, B1 and C are initially online, while B2 is offline. If B1 fails, B2 is turned on and continues its function, allowing A1 to continue operating as well. If C fails while B1 is operational, A1 fails and B1 is turned off. Thus, components Z and Y are dependent on the operation of both B1 and A1 for being online or offline.

It is undesirable to have more than one "System Operational" place enable or inhibit components' online and offline transitions, as shown in Figure 6.34. This is because, to continue the example used above, while the online transitions of Y and Z would rightly be disabled whenever either of A1 or B1 were disabled, the offline transitions would need both of them to be inactive to be enabled. So, if B1 failed and switched to B2, its components' offline transition would be inhibited as A1 is still operational. The solution to this problem is, firstly, to allow A1 to control the marking of the operational places of both B1 and B2, such that they can only be marked when A1 is marked. Secondly, if any component is controlled by two redundant systems in a hierarchy, as shown in Figure 6.40, only its most immediate parent system with redundancies is allowed to operate its online and offline transitions.

For instance, the platform in Figure 6.40 would allow the "System A1 Operational" place to control only the components in System C, U and V. Components Y and Z would be controlled by "B1 Op", and W and X by "B2 Op". The marking of child operational places by the parent operational place is performed as shown in Figure 6.41. If the parent system operational place ("P Op") becomes marked, an immediate transition will copy this to the child item operational place ("C Op") as long as it is not already marked with o_C tokens and is not failed. Once again, the arcs have multiplicities related to the number of outputs of the parent and the child. A second transition removes the tokens from C Op when P Op is marked with less than o_P tokens.

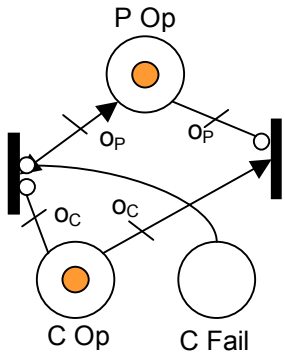


Figure 6.41 – Marking of Child Op places by Parent Op place

A point which has already been mentioned briefly is that an item can have more than one redundancy. Similarly, an item can be the redundancy for more than one input. In the former case, the places representing "X op", "X failed", "Yes", "No", and the transitions which affect just these places (numbers 1, 2 and 3 in Figure 6.35; numbers 1 and 2 in Figure 6.36), are not duplicated. Places for "Y op", "Y not op", and transitions inputting and outputting from these are created for each redundant item, if they do not exist already. These transitions are numbers 4-9 in Figure 6.35 and 3-8 in Figure 6.36. If an item is being used as a redundancy for multiple primary items, its "Y op" and "Y not op" places will be reused, as will the transitions in Figure 6.37 and Figure 6.39. Transitions 4-9 in Figure 6.35 and 3-8 in Figure 6.36 will be duplicated for each primary input to the redundant item.

In summary, a method has been shown which allows creation of Petri nets to model the situation where a system or component has a redundant item which can perform

its function in the event of its failure. There is the capability to model any combination of system and component, with multiple inputs or outputs possible. Redundancies can be switched either manually, with a penalty relating to the probability of failure to switch on the redundant item; or automatically through sensors, with a penalty relating to the potential for failure of the sensors.

6.3.8 Prognostics

Prognostics have been identified as one of the key areas to help bring about a useful duration for an MFOP. Prognostic technologies attempt to accurately predict the future failure of components or systems. This can be through a health monitoring system, which uses sensors to keep track of the stresses on and lifetimes of items, or to measure the deterioration in output of a system. Another option is inspection, where maintenance engineers visually examine an item to verify its level of wear. A third part of prognostics is Scheduled Life Replacement, where an item is automatically replaced after a certain lifespan regardless of how worn it is. This was explained in section 6.3.5.

Modelling prognostic systems introduces some complexities – in real life, components have masked future failure times, and the exact time of failure is impossible to predict. However, during their life, components may give signs which indicate how they cope with the stresses they are placed under, and thus how they may last into the future. Warrington *et al* discuss one method of modelling prognostics, as included in their URAM simulation model [23]. Components which can be monitored have a *prognostic indicator* which allows an estimate of their future failure time to be made. However, the accuracy of this prediction is based on the skill and experience of the technician inspecting the item. The prognostic model used in URAM takes account of this, assigning a probability value to the event of the prognostic indicator being successfully noted or not. This value increases over time, as the indicator becomes more difficult to miss. Also accounted for is a minimum time before which the indicator cannot be noticed, known as a *prognostic horizon*.

It is difficult, however, to reconcile this concept with randomly sampled failure times within a computer simulation model. As stated above, it is expected that the prediction of a future failure time on the simulated time to failure. In a real situation, however, the future failure time is not known – the only clues to this figure are the levels of wear which occur. In order to accommodate this fact, therefore, a different model to Warrington *et al* has been proposed.

The prognostic model can be broken down into several parts:

- Creating 'wear' in components – this wear can be either physical, particularly in terms of mechanical items, or it can be simply a way of expressing deterioration in an item's performance. The term 'wear' is used to denote any performance measure or physical characteristic which would allow prediction of a component's future life.
- Establishing what the wear level on the component is, through inspection or diagnostic systems.
- Using the amount of wear on a component to predict what life it is likely to have left.
- Considering the missions within the MFOP and the revealed system and component failures together with the predicted failure times for prognostic-monitored components to establish whether a mission is likely to fail catastrophically.
- If a catastrophic failure is found to be imminent mid-mission, abandoning (and therefore failing) the current MFOP to preserve the platform. Alternatively, when the platform is analysed during an MRP, looking at possible future failures and replacing them before they can cause failures in the next MFOP.

6.3.8.1 Modelling Component Wear

The proposed method of modelling prognostics considers each item which is monitored to have at least one level of wear specified for it, with each level corresponding to a defined portion of the lifetime of the component. For instance, a given component may have the following levels of wear:

- Level 1 (Early wear/bedding-in) – 0-10% of item lifetime
- Level 2 (Entering late life) – 70-80%
- Level 3 (Failure imminent) – 90-99%

Each level of wear, as seen, corresponds to a percentage of the lifetime of the item in which it tends to occur. These levels of wear, when they occur, can then be used to make pessimistic estimates of when the future failure will occur. The model considers the probability of the wear level occurring at any time to be uniform within the percentage bounds stated, and zero outside those levels.

The percentage values pertaining to each level of wear are inputted to the model. This can then use a randomly sampled number between 0 and 1 to generate a percentage point of the component's life at which it will enter that level of wear, as Figure 6.42 shows.

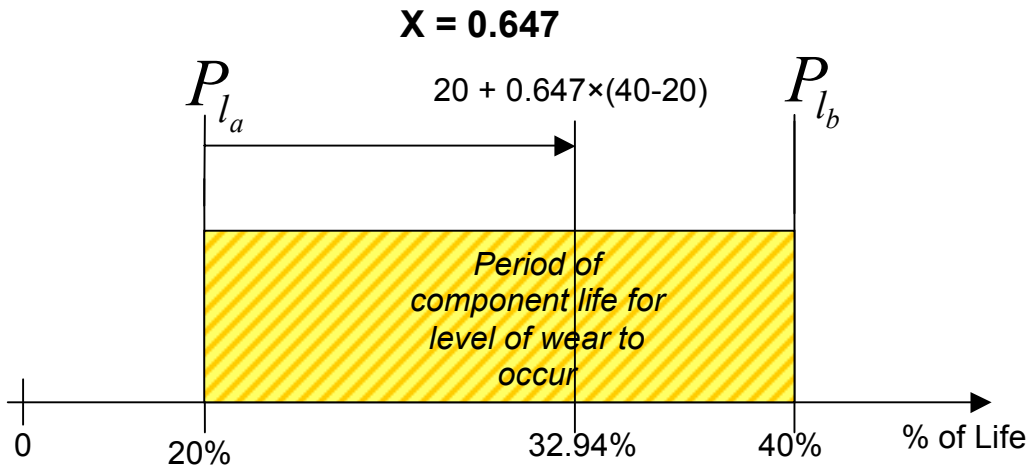


Figure 6.42 – Sampling a Random Point of Component Life for Wear Level

In Figure 6.42, a number, X, has been randomly generated and is set to 0.647. If the range between P_{l_a} and P_{l_b} is considered as a unit, then the random number defines a proportion of that unit. The time at which the wear level is sampled to occur is therefore as shown in equation 6.1.

$$t_l = \left(X \cdot (P_{l_b} - P_{l_a}) + P_{l_a}\right) \cdot \left(t_{F_c} - t_{R_c}\right) + t_{R_c} \qquad \textbf{6.1}$$

Using this formula gives a progression through wear which has a degree of randomness, and is linked to the component's life. In the example shown in Figure 6.42, P_{l_a} = 20% and P_{l_b} = 40%. The percentage life at which the component will switch to the wear level is therefore 0.647 × (40-20) + 20 = 12.94 + 20 = 32.94%. When multiplied by the component's lifespan and added to the last repair time, the time to enter the wear level is found. This time is used in a PN which allows the component, when operational, to build up wear.

Figure 6.43 shows the standard "up" place for a component as being connected to a timed transition. Whenever the component is operational, this transition will be enabled, and the time generated for it to switch will be equal to that calculated by equation 6.1. Whenever it does switch, an additional token will be deposited in the "Wear Level" and "Control" places, indicating the current level of wear of the component. Once the maximum number of wear levels, n_l, is reached, an inhibitor arc with that multiplicity prevents the transition from switching.

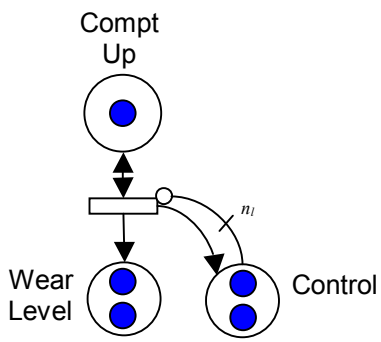


Figure 6.43 – Modelling Wear in Components

6.3.8.2 Diagnosing Wear Level

Once a method of modelling wear in components is established, a further issue is that of discovering what the wear level is. As mentioned in section 6.3.6, there are two main ways of diagnosing faults in a system: through sensors or inspection. This also applies to discovering wear in a component, and affects not just the net which is created, but also the way in which the prognosis can be dealt with.

If sensors monitor the wear in the item, then the wear level will be discovered immediately, during an MFOP, and action can be taken to mitigate it should an imminent component failure be important enough to threaten the mission. The only issue threatening this capability is the operation or failure of the sensor component. Figure 6.43 shows how this is applied in PN form. The "sensor failed" place for the sensor component inhibits the wear level timed transition, so that the detected level of wear, equal to the number of tokens in the corresponding place, stays equal to the last updated value. As the component's failure time gets closer, no more wear is detected, and so no mitigating actions can take place unless any predictions from wear that has been found are accurate enough to prevent mission failure.

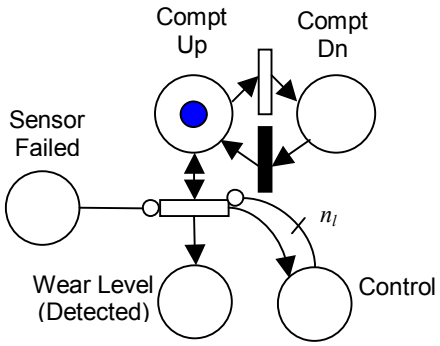


Figure 6.43 – Sensor Detecting Current Wear Level

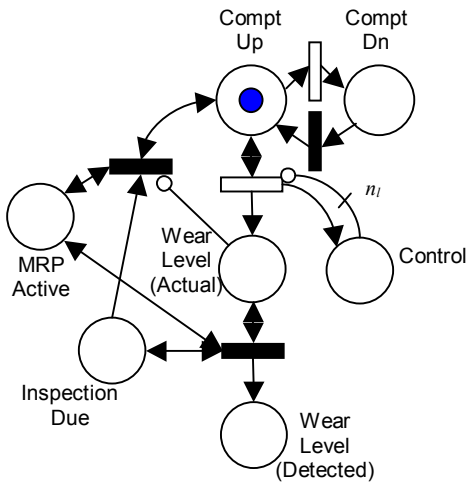


Figure 6.43 – Inspecting Component Wear Level through Inspection

If, instead, the wear levels are found through inspection, then as inspections can only take place during an MRP, there are two types of wear level – the state in which the component is in, in real-time, and the state of which the operator is aware, for which plans can be made. This difference between actual and detected wear levels is shown in Figure 6.43. In the Figure, one place representing both types of wear level is shown. The timed transition inputting to "Wear Level (Actual)" is left free to switch, as long as the control place is not inhibiting it. This place cannot cause any actions to take place, however. Only once an MRP is taking place and an inspection is due will the current wear level be discovered and component replacements are able to take place. The transition which removes the inspection

place's token, should the component be found to be operational, is inhibited by the Wear Level (Actual) place, to prevent the loss of the token occurring before the inspection place can move the tokens from Wear Level (Actual) to Wear Level (Detected).

6.3.8.3 Predicting Component Failure Times

The purpose of creating a model for component wear is to allow the level of wear to provide information about the probable life of the component. Calculations can be made regarding the level of wear and the time at which it was detected in order to create bounds for the component failure time. These calculations take into consideration a confidence interval in order to provide a value for component failure time.

The simplest situation is where a component's increase in wear level is detected by a sensor. Note that "wear level" refers to the category of wear of the component (a discrete level), rather than the continuous amount of wear on the component. If a component's wearing is sensed, then the time at which the present wear level category was entered is known very accurately. The formula which calculates the lower and upper bounds for the component failure time, t_{F_a} and t_{F_b} respectively, where the time of entering the wear level is known accurately is shown in equation 6.2 below.

$$t_{F_a} = 100 \times \max\left(t_{F_a}, \frac{t_{l=n_l} - t_R}{P_{n_{l_b}}} \right)$$
$$t_{F_b} = 100 \times \min\left(t_{F_b}, \frac{t_{l=n_l} - t_R}{P_{n_{l_a}}} \right)$$

6.2

Setting initial values for t_{F_a} and t_{F_b} as being zero and infinity respectively, equation 6.2 sets their latest values to the most accurate of either the current value, or a new value calculated with the latest information. This new value is the component's age at entering the wear level divided by one of the bounds to reach a minimum and a

maximum for the failure time. As an example, consider a component, operational since time t=0, whose wear levels are specified to be:

- Wear level 1 – 30-40%
- Wear level 2 – 50-65%
- Wear level 3 – 78-92%

The component is sensed as entering wear level 1 after 34.092 hours. This marks the bounds of its failure time as lying between t_{F_a} and t_{F_b} as calculated below.

$$t_{F_a} = 100 \times \max\left(0, \frac{34.092}{40}\right) = 85.23 \text{ hours}$$
$$t_{F_b} = 100 \times \min\left(\infty, \frac{34.092}{30}\right) = 113.64 \text{ hours}$$

These bounds for the failure time of [85.23,113.64] hours are fairly loose, mainly due to the early time and the bands for wear level 1. It is usual for the failure time to become more accurate and refined as the component ages. A confidence interval specifies the point at which we are sure to that proportion that the component will not have failed before. For instance, if the confidence interval is specified to be 95%, then this suggests that we can be 95% certain that the component will not fail before 86.65 hours in this example, as 85.23 + (113.64–85.23)×(1–0.95) = 86.65 hours.

If the component enters wear level 2 after 66.86 hours, the bounds for the failure time are refined as shown below:

$$t_{F_a} = \max\left(85.23, 100 \times \frac{66.86}{65}\right) = \max(85.23, 102.86) = 102.86 \text{ hours}$$
$$t_{F_b} = \min\left(113.64, 100 \times \frac{66.86}{50}\right) = \min(113.64, 133.72) = 113.64 \text{ hours}$$

Thus the earlier failure time bound has been refined substantially, but the later time has remained the same. The confidence interval of 95% suggests a new estimated

failure time of 103.40 hours. Calculating results for wear level 3 may lead to further refinement, and thus allow much more confidence over abandoning an MFOP at a particular time, or replacing the component in an MRP before it has had the chance to fail.

While sensed component monitoring can lead to good levels of accuracy in predicting component failure time, it is not so straightforward when component wear is only discovered through inspections. In this scenario, component failure times are predicted similarly, but with less accurate information. Consider the example used with a sensed component above, but instead of the wear level being discovered immediately, it is found through inspections at regular intervals of 40 hours.

As above, the component enters its wear level 1 at time 34.092 hours. At the inspection at time 40 hours, the component is found to have entered wear level 1. The earliest point at which it could have entered this is at the very beginning of the component's life, and the latest point is just before the current inspection. To find the bounds for the component failure time, equation 6.2 can be adapted to take these considerations into account:

$$t_{F_a} = 100 \times \max\left(t_{F_a}, \frac{t_{I_{z-1}} - t_R}{P_{l_b}} \right)$$
$$t_{F_b} = 100 \times \min\left(t_{F_b}, \frac{t_{I_z} - t_R}{P_{l_a}} \right)$$

6.3

From equations 6.3, the bounds can be calculated as accurately as possible:

$$t_{F_a} = 100 \times \max\left(0, \frac{0}{0.4} \right) = 0 \text{ hours}$$
$$t_{F_b} = 100 \times \min\left(\infty, \frac{40}{0.3} \right) = 133.33 \text{ hours}$$

A problem with the calculation above is that, obviously, if the component's failure time were 0 hours, then the component would have failed immediately, and not be at wear level 1. A limiting factor on the minimum value, therefore, is whether or not the suggested failure time would lead to entering wear level 2 before the current inspection. This limiting condition can be expressed as shown in equation 6.4 below.

$$t_{F_a} = \max\left(t_{F_a}, \frac{t_{I_{z-1}} - t_R}{P_{l_b}}, \frac{t_{I_z} - t_R}{P_{l+1_b}} \right)$$

6.4

Given that the earliest time that the component could enter wear level 2 is negligibly after the current inspection time, the earliest time that this suggests for the component failure time is $t_{F_a} = 100 \times \max\left(0, 0, \frac{40}{65}\right) = 61.54$ hours. The final term in equation 6.4 states this as the inspection time divided by the upper percentage bound for the next wear level. The bounds of (61.54,133.33) suggest a 95% confidence interval of failing after 65.13 hours.

At the next inspection, at 80 hours, the component is in wear level 2. The predictions for component failure time come out as

$$t_{F_a} = \max\left(61.54, 100 \times \frac{40}{65}, 100 \times \frac{80}{92}\right) = \max(61.54, 61.54, 86.96) = 86.96 \text{ hours}$$

$$t_{F_b} = \min\left(133.33, 100 \times \frac{80}{50}\right) = \min(133.33, 160) = 133.33 \text{ hours}$$

which suggests a 95% confidence interval failure time of 89.28 hours. Given that the next MFOP is likely to take longer than 9.28 hours to complete, if the failure of the component will cause mission failure, then the component would be replaced during the MRP in which the inspection is taking place. The assessment of whether components are critical for mission and MFOP operation is covered in the next section.

6.3.8.4 Assessing MFOP Criticality

Once predictions have been made for all components whose wear levels have updated, it is important to check whether the new information suggests that the current mission or MFOP is under threat. There is a hierarchy of desired outcomes from them, which affects how the prognostic systems are applied. The best result from an MFOP is the successful completion of all missions within it. If, however, the MFOP is destined for failure, then abandonment of a mission before it has begun (for aircraft, this will be during the safest point of a mission, before take-off) is best. During a mission, it is most important to prevent catastrophic failure, and loss of the platform, so that it can carry out more missions in the future. If the current mission must be abandoned to do this, then this is the best option.

In order to best protect an entire MFOP, an assessment of component wear, taking into account revealed and sensed failures which have not been repaired, for each individual mission is performed during the MRP preceding it. This forces the replacement of any component which is considered critical. During an active MFOP, criticality is only tested for the current mission. If the assessment results in a possibility of completing the current mission without catastrophic failure, then the mission is not abandoned. It will only be aborted once it is considered that there is no way to avoid total loss. This criticality assessment is also performed at the start of a mission, in order to prevent it taking place if the platform is not in a healthy state.

The criticality assessment consists of three main parts:

1. Assigning predicted durations of each phase in the mission
2. Establishing what the most likely mission phase profiles are
3. Estimating whether each phase in the mission is likely to fail or not.

Appendix A.4.2.4 explains the methods by which the model assesses the criticality of the current mission and MFOP in more detail. Because phases can be inserted or chosen to follow the current phase, as explained in sections 6.2.12 and 6.2.13, this can lead to complicated or multiple possible mission profiles. The model considers

each phase insertion, choice and abandonment and predicts the mission profile based on how they currently stand.

If, however, there is a probability-based phase choice at some point within the mission, this leads to the creation of multiple mission profiles, each with a different probability attached. Mission profiles with a probability of less than 1×10^{-6} are discarded, while in the others, the phase durations are summed to find the estimated duration of the mission. The mission profiles are then sorted in order of length and probability of occurrence. The probabilities of each mission profile are added together with increasing duration and, once the summed probability is greater than the confidence interval, mission profiles with longer durations are discarded.

The model then takes each phase in each mission profile in turn. Taking into account revealed and sensed component failures as well as components predicted by the prognostics model to have failed by the time that phase has ended, decides if any phase in that profile will have failed, thus failing the mission profile. Depending on the predictions made, the model will either take immediate action to abandon the current mission, replace all critical components, or do nothing. The method by which the PN performs these actions is explained in the next section.

Usually, once a component gets to a certain wear level, it would be replaced at the next MRP – this is Condition Based Maintenance. The prognostic model used here makes predictions which consider the importance of a component to a mission. Hence, if the component's failure in the current mission or forthcoming MFOP is calculated to be critical, it will be repaired in the next or current MRP. If it is not, the platform is considered to be capable of carrying the potential failure, and it will be repaired once it has actually failed.

6.3.8.5 Modelling Actions Taken Based on Predictions

There are, as stated above, two main actions that can be taken if component failures are deemed to be imminent. If the component failures are predicted to cause mission and MFOP failure before they can be satisfactorily completed, then they will be abandoned. If this occurs, or if during an MRP components are found to be critical for any part of the next MFOP, they are replaced.

In order to perform any of these actions, the wear level place shown in Figure 6.43 is linked to a timed transition, which inputs to a place called "Action", as shown in Figure 6.43. Only one arc leads from the wear level place to the timed transition and back, in order to enable it, rather than n_l arcs. This is because the action place can be marked even if the wear level place only contains one token, if the model predicts the component failure to occur before the end of the current mission.

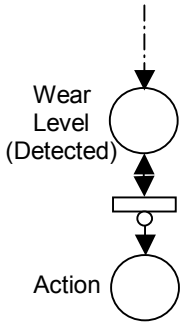


Figure 6.43 – Action Place

The action place sets off component replacement during either an MFOP or an MRP. The way it does this varies according to whether or not the component is a member of a LRI. If it is not, a replacement place and transitions are created as in Figure 6.17, with the Action place acting as the replacement trigger. Figure 6.44 shows this in more detail.

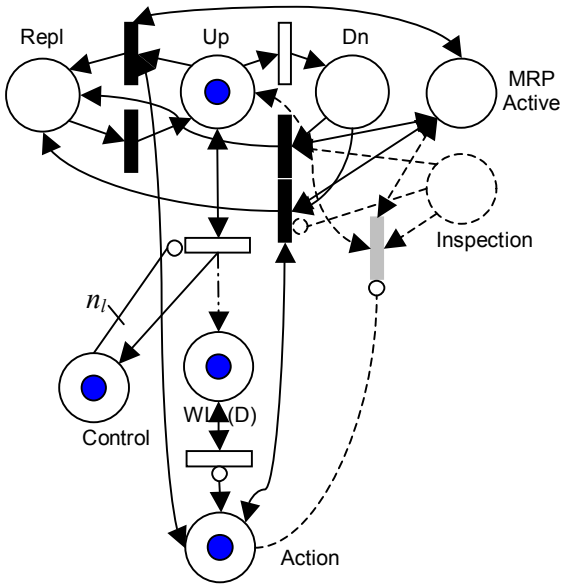


Figure 6.44 – Prognostics Prompting Component Replacement

In the Figure, the action place is connected to the immediate transition to the top-left of the "Up" place via an input arc and an output arc. Whenever the action place is marked and the component is operational, the component will be replaced during the next MRP. If the component has failed, then the action place is likely to have been marked if the component was critical for the MFOP. Either way, the failed component will be replaced at the next MRP. If the component undergoes

inspections, the inspection place will be an input to the repair transition shown to the bottom-left of the “Dn” place. The transition immediately below this, which the action place enables, is inhibited by the inspection place so that its token will be used up in the next MRP.

The method by which the failed component is replaced is unimportant, either by inspection or due to a prognosis. The transition which removes the inspection place if the component is operational, shown in grey in Figure 6.44, is inhibited by the action place (this is the same transition which is also inhibited by the “Wear Level (Actual)” place, as shown in Figure 6.43.

If the component is an input to an LRI, then the PN for replacement is set up slightly differently. The creation of an LRI will have already set the component up for replacement, and if the component is a direct input to the LRI, then the component failure place will have an input and an output arc to an LRI input transition, as shown in Figure 6.18 and Figure 6.19.

Figure 6.45 shows the component with replacement net, and the prognostic net as well. Similarly to the inspection place in Figure 6.44, the LRI place enables the component repair transition and inhibits the prognostic transition allowing component repair. In this case, however, the action place does not enable the component replacement transition (to the top-left of the “Up” place). Instead, whenever it is marked, it forces the LRI which the component is a part of to be replaced, regardless of the state of the other members of it. The LRI will cause the component to be replaced instead.

When the component replacement place is marked, immediate transitions are enabled which remove the tokens from the action, wear level (actual), wear level (detected) and control places. Each of these places inhibits the transition emptying the replacement place, so they can each be cleared before the net moves on.

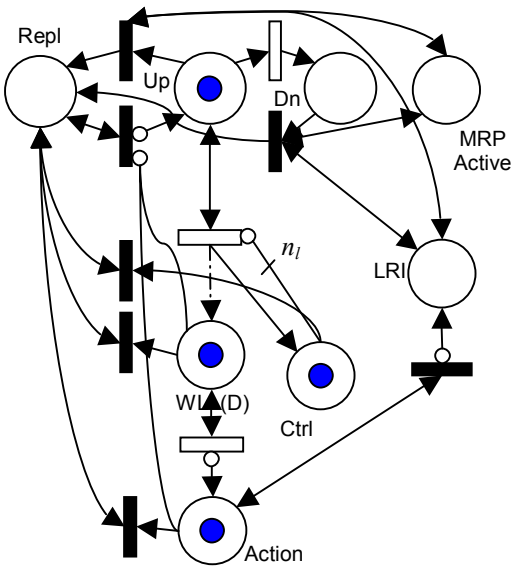


Figure 6.45 –Component Member of LRI with Prognostics

The other action that the prognostic system can take if it predicts catastrophic failure, is abandoning an active mission. Section 6.2.11 explains how, if specified by the operator, a method to fail or abandon any phase in any mission is provided. This general failure mechanism is used to allow the prognostic system to abandon the mission. Regardless of whether the operator has included consideration of general abandonment, a top event place for this eventuality, in addition to all of the transitions which remove the place from the phase places, as shown in Figure 6.12, is created anyway. When prognostics are specified for a component, a link is created between the "action" place and the "General abandonment top event" place via an immediate transition, as shown in Figure 6.46. This is inhibited by the "MRP Active" place, so that the transition will not switch during an MRP, when a mission cannot be abandoned.

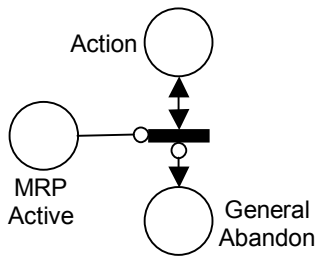


Figure 6.46 – Prognostic Abandoning of Mission

To summarise, a method, procedure and set of PNs have been explained which allow the model to consider prognostic systems, which predict the future failure of

components and their effect on the platform. This allows the abandonment of missions and the replacement of items before they are allowed to result in total loss.

6.4 Summary

This chapter has explained the creation of nets which simulate the behaviour of systems undergoing MFOPs and phased missions. The modelling method presented takes account of complex aspects of MFOPs and Phased Missions, including:

- The Petri nets are newly capable of modelling multiple platforms within a fleet, multiple MFOPs and multiple missions. Missions and MFOPs can be catastrophically failed or abandoned, with catastrophic failure leading to total loss of the platform.
- A "general phase" set of fault trees can be provided, in order to allow a mission or MFOP to be failed or abandoned in any phase.
- A mission need not consist of a simple linear progression from one phase to the next. Phases may be inserted immediately, interrupting the current phase, which resumes afterwards. Alternatively, the next phase may be chosen, either randomly or resulting from the failure or operation of certain systems or components, from an array of possible candidates.
- Discrete events are events which do not occur according to a probability-time distribution, but in response to a stimulus. The stimulus may be any event, such as the commencement of a particular phase, the failure of a system or starting an MRP. When the trigger event occurs, the discrete event either occurs or does not occur according to a set probability. This can then feed into phase or general fault trees.
- In real life, components may be replaced when they have not failed. This is modelled in the program by using a "component replacement" place. When this is marked, the component's time to fail is reset and a new one created when the component is operational again.
- The function of replacing items regardless of being operational or failed is used by a method of modelling Line Replaceable Items. These are batches of

components on the same physical item, which are easily removed and replaced, to provide a quick changeover during maintenance.

- Another complexity of MFOPs to be considered is Scheduled Life Replacements. These force a component or LRI to be replaced in the next MRP after a certain time.
- A method of modelling prognostic monitoring systems is presented. This simulates the build-up of wear in components, and then uses that wear to calculate predictions for their future failure. If this future failure is imminent and critical for the MFOP, the current mission can be abandoned and the component set to be replaced.
- Diagnostic modelling is provided in the form of "sensors", which discover failures and faults in real-time, during missions; and "inspections", where these are only found during the Maintenance Recovery Period. Finding these failures can affect the replacement of components, systems or LRIs, and can affect mission abandonment.
- Redundancies are items which take over the functionality of a specific component or system whenever it fails. The capability for modelling this is also included here.

All of these modelling methods have been implemented in a form which allows production of results which describe the performance of the platform or fleet. This is explained in Appendix A. The next chapter describes the application of the proposed model to a life-sized system, and the methods by which suggested improvements are evaluated for effectiveness.

Chapter 7 Application of Model to Large System

7.1 Introduction

A modelling method has been created, both in theory and in the form of a software tool, which allows the estimation of reliability parameters for a fleet of platforms performing maintenance-free operating periods and phased missions. It is important, however, to demonstrate the potential applications of this in the real-world industry. One of the potential uses for it could be in the design stage of a new military aircraft, to attempt to minimise the possibility of missions failing. Another is in the proposed installation of systems designed to protect mission reliability onto a currently operational platform.

In order to show some of the principles which may be applied when using the method developed in this thesis, an example is given in this chapter which takes a large, complex system architecture and attempts to reduce the MFOP, mission and phase unreliability figures through the use of MFOP enablers.

7.2 Application to Previous Systems

Before being applied to a large system, the complex program was first used with the three examples employed in Chapter 5. This was in order to establish that it was producing verifiable results, and is consistent with expectations, giving a degree of confidence that the results for more complex systems will also be accurate. The reason for the reuse of these systems was because the nets produced by the complex model do not exactly match those of the simple model explained in Chapter 5. For instance, a net for a platform will be created featuring the MFOP sequence, and the inspection procedure for the repairable system differs in the complex model. As a result of the difference in the PNs, a comparison between the simple model results and those of the complex model becomes necessary.

7.2.1 Non-repairable Small System

The details of the non-repairable small system are given in section 5.3.1. These details were inputted to the new model, and 10,000,000 simulations performed. The results of the simulations from the complex model are compared to those of the analysis and of the simple model, in Table 7.1. These results are the phase failure probabilities.

Table 7.1 – Comparison of Model and Analysis Phase Failure Probabilities for Non-Repairable System

Phase	1	2	3	4	5	6
Analysis	0.00225	0.03850	0.05107	0.00194	0.03452	0.05447
Simple Model	0.00223	0.03849	0.05111	0.00193	0.03443	0.05451
Complex Model	0.00224	0.03857	0.05096	0.00194	0.03453	0.05442
Per cent Error	0.258%	0.173%	0.216%	0.105%	0.036%	0.092%

Phase	7	8	9	10	11	12
Analysis	0.05210	0.00327	0.03271	0.05493	0.05302	0.00451
Simple Model	0.05213	0.00330	0.03280	0.05500	0.05298	0.00447
Complex Model	0.05209	0.00328	0.03270	0.05486	0.05302	0.00448
Per cent Error	0.027%	0.229%	0.031%	0.124%	0.008%	0.660%

The Table shows a high degree of correlation between the phase failure probabilities given by the complex model and those from the analytical results. The highest error is just under two thirds of a per cent, though this higher figure can be attributed to the very low failure probabilities. In reality, the figures are very accurate.

7.2.2 Repairable Small System

Section 5.3.2 discusses a repairable system for which a Markov analysis method was used to solve. These results were compared with a run of 10,000,000 simulations of the simple model. 10,000,000 more simulations were performed with

this system using the complex model. The results from this are compared with the previous sets of results as in Table 7.2.

Table 7.2 – Comparison of Model and Analysis MFOP Failure Probabilities for Repairable System

MFOP	MFOP Failure Prob. (Markov)	MFOP Failure Prob. (Simple Model)	MFOP Failure Prob. (Complex Model)	Percentage Error between results
1	0.324730	0.324872	0.324961	0.0710%
2	0.333304	0.333222	0.333218	0.0258%
3	0.339405	0.339228	0.339415	0.0029%
4	0.324815	0.324865	0.324308	0.1560%
5	0.333167	0.332750	0.333617	0.1351%
6	0.339582	0.339362	0.339962	0.1120%
7	0.324777	0.323946	0.324464	0.0962%
8	0.333110	0.333117	0.332244	0.2600%
9	0.339528	0.339884	0.339519	0.0027%
10	0.324715	0.324741	0.324702	0.0039%
11	0.333333	0.333041	0.332512	0.2463%
12	0.339443	0.339061	0.341150	0.5030%

The Table shows, once again, a good degree of correlation between the complex model results and those of the simple model and the Markov analysis. The highest error is just over half a per cent, while most other errors are less than a tenth of a percent. These figures are also very accurate, and lend yet more confidence to the complex model.

7.2.3 Repairable Larger System

Section 5.3.3 describes a larger repairable system, featuring ten phases and ten components. It has features such as phase abandonment and component dependencies. 1,000,000 simulations were performed using the complex model program, and these are compared to the results from the simple model in
Table 7.3 and Table 7.4.

The Tables show a good correlation between the two sets of results. The relative error between every data point for the complex and simple model is less than 1%.

There is one exception to this, the failure probabilities for phase 3, which have an exceptionally low value.

Table 7.3 – Comparison of MFOP and Mission Failure Probabilities for Large Repairable System

MFOP/ Mission	Simple Model Failure Prob.	Complex Model Failure Prob.	Failure Prob. error
MFOP 1	0.11327	0.11346	0.1651%
MFOP 2	0.06858	0.06903	0.6461%
MFOP 3	0.04211	0.04243	0.7621%
Mission 1	0.03886	0.03916	0.7747%
Mission 2	0.04035	0.04046	0.2671%
Mission 3	0.04438	0.04418	0.4546%

MFOP/ Mission	Simple Model Abandon Prob.	Complex Model Abandon Prob.	Abandon Prob. error
MFOP 1	0.61805	0.61734	0.1146%
MFOP 2	0.86467	0.86420	0.0546%
MFOP 3	0.94240	0.94226	0.0155%
Mission 1	0.34237	0.34162	0.2184%
Mission 2	0.47986	0.48034	0.0992%
Mission 3	0.53262	0.53202	0.1125%

Table 7.4 – Comparison of Phase Failure Probabilities for Large Repairable System

Phase	Simple Failure Prob.	Complex Failure Prob.	Error between results
1	0.02119	0.02130	0.4892%
2	0.02740	0.02743	0.0823%
3	1.21×10^{-5}	1.42×10^{-5}	14.3073%
4	0.01799	0.01803	0.2428%
5	0.15895	0.15887	0.0464%
6	0.01563	0.01573	0.6775%
7	0.02864	0.02879	0.5284%
8	0.07067	0.07054	0.1847%
9	0.00000	0.00000	-
10	0.22204	0.22164	0.1801%

Thus, for each of the models used by the simple program, the complex model has been able to reproduce the results very accurately. While this only confirms that the modelling of the simple aspects of the complex model is accurate, this is still an important basis to continue on to modelling a large system.

7.3 Application to a Large System

The close correlation between the results from the simple model and those from the complex model suggest that the complex model is capable of producing satisfactory results. A more complicated demonstration of its operation is given in this section.

In order to ensure that the model was capable of handling the type of large, complex systems used by operators such as the Royal Air Force, systems information on a model of aircraft were provided. This information comprised a single-phase fault tree establishing how the aircraft could experience total loss, and a set of documents explaining missions systems and giving component failure data. Also provided was information on three typical missions that the aircraft is typically required to perform. This data has been modified to produce only a demonstration of the sort of results which can be achieved.

The conversion of this data into something compatible with the model required the production of two sets of fault trees – catastrophic failure and abandonment for each phase. The single-phase total loss fault tree was slightly truncated, due to its size, so that intermediate events at a certain level of resolution were categorised as "undeveloped", with their failure rates set to that calculated by FaultTree+. After this, each branch of the tree was analysed for its relevance to each phase of each mission that the aircraft would perform. If it was deemed to be not applicable to a particular phase, the branch was deleted. Thus, a number of distinct total loss phase fault trees were produced. The compilation of the set of phase abandonment fault trees was based on the observation that, while engine failure and similar incidents may cause catastrophic failure, the failure of an aircraft to deploy a missile or sonobuoy can only cause the mission to fail and be abandoned. Thus, the first set of mission abandonment fault trees was based on the failure of mission-critical, but non-flight-critical systems. Each fault tree considers the failure of any of these phase-critical systems as prompting phase abandonment. The causes of failure for each of the systems were estimated from the systems information and component failure data provided.

An important point is that while the information used with the model is based on that available for the aircraft, it would be very difficult to reproduce the typical conditions of mission or platform failure without an expert knowledge of the system. Because of this, efforts have been made to produce a set of mission scenarios which are realistic in terms of size, complexity and in the failure rates used, but may not necessarily represent accurately the operation of an aircraft in an actual military environment. The aim of this chapter is to demonstrate the usefulness of the Petri net modelling method when applied to a large system; it is not to exactly replicate and improve the mission performance of a military aircraft.

7.3.1 The System

Much of the data regarding the systems which make up the aircraft is subject to security restrictions, and so cannot be published in full. This section, however, aims to give a brief overview.

7.3.1.1 Potential for Total Loss

The main ways in which Total Loss can occur are considered to be:

- Deviation from intended flight path:
    - Loss of aerodynamic control (LAC):
    - Loss of/incorrect thrust from engines: this could be due to lack of fuel, insufficient thrust from engines (for instance, due to an engine failure), or an erroneous supply of air data due to icy conditions.

- Loss of structural integrity:
    - Structural damage and physical hazards: either due to failure of systems such as powerplant systems or the landing gear; or due to physical hazards present in the aircraft's environment.
    - Fire or explosion: explosion could be due, for instance, to the fuel tank exploding (such as happened with TWA Flight 800 in 1996 [86]), while a fire could have stemmed from the engines, weapons bay or several other systems.

    - Failure of the landing gear to support the aircraft.

- Controlled Flight into Terrain:
    - Autopilot fails
    - Display errors
    - Erroneous navigational data
    - Pilot lack of consciousness
    - Human performance limits exceeded

7.3.1.2 Potential for Mission Abandonment

Missions are abandoned when any of the mission critical systems have failed. These have been identified as:

- Tactical Command System (TCS) – interacts with the aircraft's other systems to provide the crew with all the data they need to complete missions. Allows for mission, information and sensor management, and displays all information through various visual units. Interfaces with crew through the use of keyboards, mice, programmable entry panels, multi-function displays, and so on.
- Defensive Aids Sub-system (DASS) – this system allows the detection of threats, evaluates the size of the threat, warns the crew and puts in place measures to deal with the threat.
- Flight Management System (FMS) – provides data on aircraft position and other navigational information
- Magnetic Anomaly Detector (MAD) – detects fluctuations in the earth's magnetic field which may indicate the presence of a ferrous objects such as a submarine.
- Radio Communications System (RCS) – allows communication with other entities such as aircraft, mission control, air bases, and so on, either through voice or data transfer (such as tactics, mission data, etc.)
- Stores – items deployed during missions. May include weapons, defensive aids and sonobuoys amongst others.

- Radar/Identification of Friend or Foe (Radar/IFF) – allows detection of various items, with follow-up functions. For instance, surface vessels may be detected, classified and tracked; submarine masts may be spotted and tracked; warnings of adverse weather can be given; persons may be found during a search and rescue (SAR) operation.
- Electronic Support Measures (ESM) – detects the emission of electromagnetic data, and provides information about the emitter, such as bearing, range, position, and emitter type.
- Electro-Optical Surveillance/Detection System (EOSDS) – provides day or night imaging surveillance capability while airborne.

7.3.1.3 Missions and Phases

There are three missions used in the set of simulations: anti-submarine warfare (ASW), anti-surface warfare (ASUW) and search and rescue (SAR). Each of the missions has the following basic profile:

- Pre-Flight Check and Taxi (1 hour)
- Engine Start (15 minutes)
- Take-off (1 minute)
- Climb (15 minutes)
- Transit (3 hours)
- Descent into operational area (15 minutes)
- Operations (various durations)
- Ascent out of operational area (15 minutes)
- Return transit (3 hours)
- Descent towards runway (15 minutes)
- Approach (10 minutes)
- Landing (5 minutes)
- Taxi to hangar (30 minutes)

The order of the phases and their lengths are based upon the data provided. The individual missions have the following operational phases:

- ASUW and ASW:
  - On-Task (5 hours) – 10% of phases have an inserted "Attack" phase.
  - Attack (9 minutes) – reverts to On-Task once finished.
- SAR:
  - On-Task (2.5 hours)

Each mission requires the use of different systems to the others, and thus several of the phases mentioned above have several different total loss and abandonment fault trees depending on the mission they are used in. Each phase which can be abandoned is set to abandon to the logical return phase – for instance, "Transit" abandons to the phase "Return transit". As such, most of the return phases reverse their counterparts' durations, as mentioned in Section 6.2.1, rather than having an absolutely defined value. The phase fault trees used are given in Appendix B, in the format as inputted to the simulation program.

7.3.1.4 Platforms and MFOPs

A fleet of three platforms is used with a set of three possible MFOPs. Each platform performs each MFOP once, in a different order. These orders are:

Platform 1 – MFOP 1, MFOP 2, MFOP 3
Platform 2 – MFOP 3, MFOP 2, MFOP 1
Platform 3 – MFOP 2, MFOP 3, MFOP 1

Each of the MFOPs is a different combination of the three available missions. These combinations are:

MFOP 1 – ASUW, ASW, SAR
MFOP 2 – ASW, ASUW, SAR
MFOP 3 – SAR, ASW, ASUW

7.3.1.5 Component data

Most component data was taken directly from the information provided or best estimates obtained. However, some component data was missing, and so a universal estimate of a failure rate of one failure in a million was applied to these components, using a negative exponential distribution. Note, however, that this does not indicate that all components with a failure rate of 1×10^{-6} had missing data.

7.3.1.6 Enablers

No enablers were used in the first generation of the simulations. One of the aims of this chapter is to show the process by which platform weaknesses are identified and enablers, such as prognostics or redundancies, can be used to reduce their contribution to MFOP and mission failure.

7.3.2 Performance of Simulations and Results

Three sets of simulations were carried out, with improvements being made between each set and the next. The following sections describe the results of each set and the improvements made for the next set.

7.3.2.1 First Generation

The first set of simulations was performed and the results compiled and analysed. 70,100 simulations were performed, at which point the result was considered to have converged. Figure 7.1 shows the convergence of each of the platform catastrophic failure (CF) probabilities.

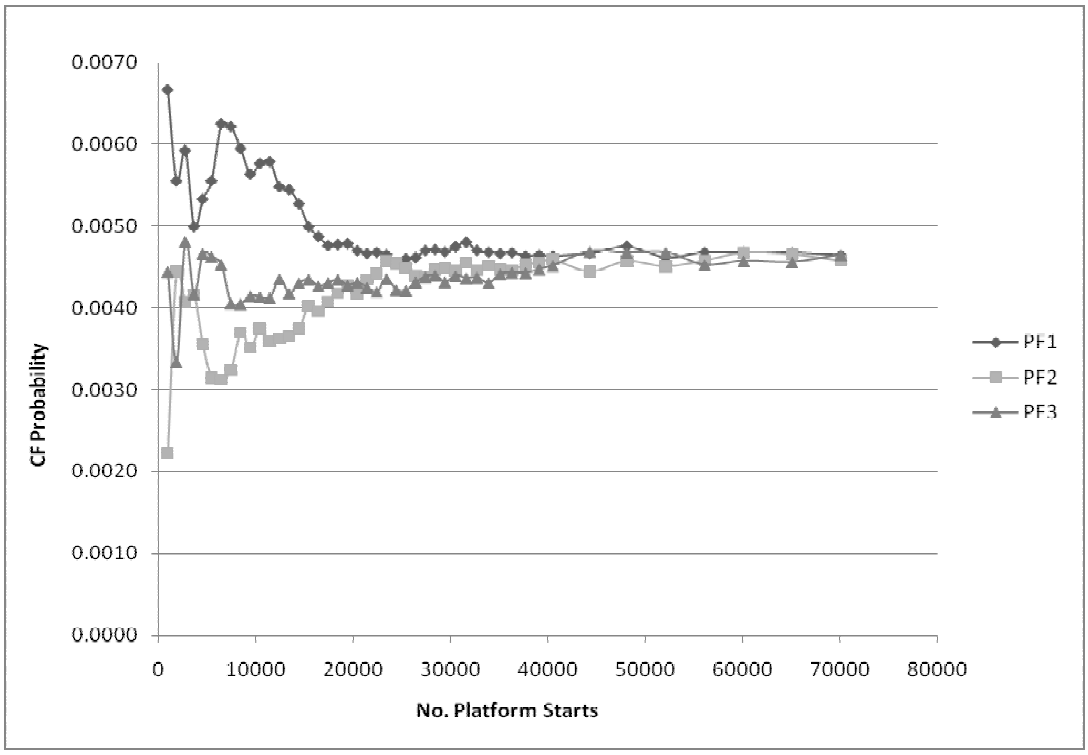


Figure 7.1 – Convergence of Platform Catastrophic Failure Probabilities

In order to demonstrate the nature of the process by which system reliability problems can be identified and potential improvements suggested, it was decided that the focus of the analysis of the first set of results would be on attempting to improve the catastrophic failure probability of the platforms. As such, a pareto analysis, whereby the most critical factors are identified in a hierarchy downwards, terminating with improvements to the most vulnerable items.

This process begins by observing the failure probabilities of the different platforms and identifying which, if any, of the MFOPs they perform are more or less prone to total loss of the aircraft. Table 7.5 shows this information.

Table 7.5 – Platform and MFOP Catastrophic Failure Probabilities

Platform	Category	No. Starts	No. CFs	CF Prob.
PF1	Platform	70100	323	0.0046
	MFOPs	210001	323	0.0015
	MFOP1	70100	98	0.0014
	MFOP2	70002	103	0.0015
	MFOP3	69899	122	0.0017
PF2	Platform	70100	322	0.0046
	MFOPs	209949	322	0.0015
	MFOP3	70100	125	0.0018
	MFOP2	69975	101	0.0014
	MFOP1	69874	96	0.0014
PF3	Platform	70100	325	0.0046
	MFOPs	209982	325	0.0015
	MFOP2	70100	111	0.0016
	MFOP3	69989	96	0.0014
	MFOP1	69893	118	0.0017

The results show that the total CF probabilities for each of the platforms are nearly identical. This is to be expected, as each platform performs the same MFOPs as each other, albeit in a different order. The results show very similar figures for each of the MFOP catastrophic failure probabilities for each platform. MFOP3 causes most problems in platforms 1 and 2, but fewest in platform 3. Similarly, MFOP1 causes fewest problems in the first two platforms, but most in the third. There is no discernible pattern between the first or last MFOPs performed by a platform causing most or fewest catastrophic failures. This lack of a pattern in the results suggests that any minor differences in the catastrophic failure probability of the different MFOPs are little more than natural variation. As such, there is no weight to place on any of the three MFOPs as being more or less of a problem than the others.

Considering this, the analysis then looked at the absolute MFOP failure probabilities independently of the platforms, and how the three different missions affect these. The convergence of each of the MFOP CF probabilities is demonstrated in Figure

7.2, while the numerical data on the MFOP CF probabilities and those of the missions within them is given in Table 7.6.

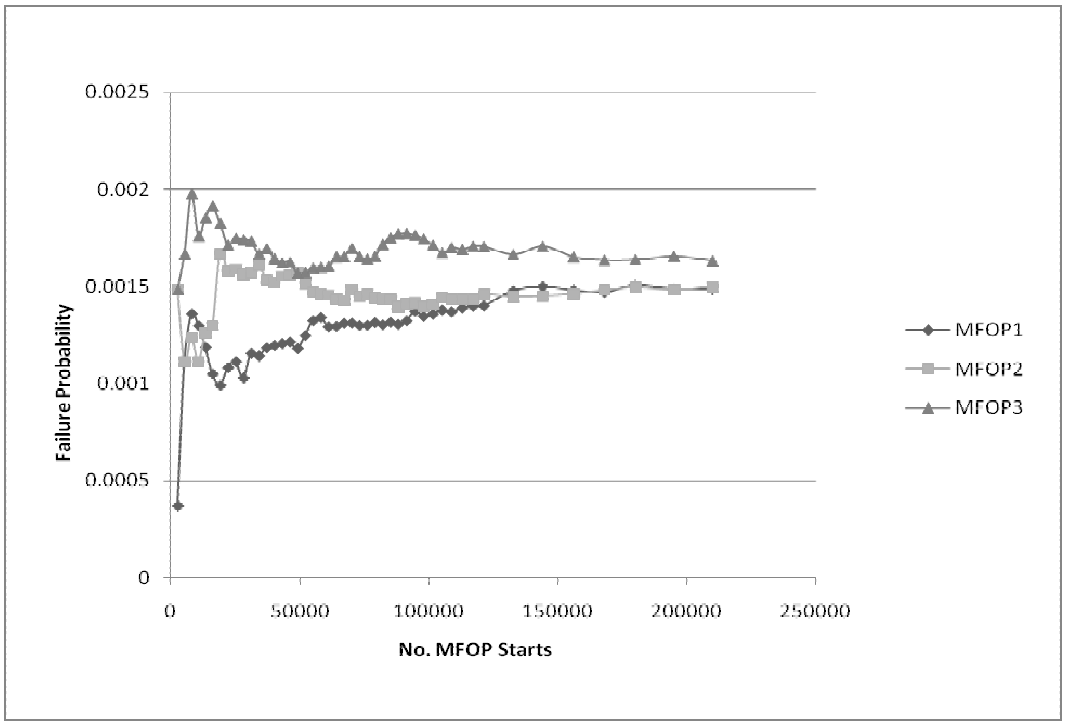


Figure 7.2 – Convergence of MFOP Catastrophic Failure Probabilities

Table 7.6 – MFOP and Mission Catastrophic Failure Probabilities

MFOP	Category	No. Starts	No. CFs	CF Prob.
MFOP1	MFOP	209867	312	0.00149
	Missions	583330	312	0.00053
	ASUW	209867	113	0.00054
	ASW	199925	102	0.00051
	SAR	173538	97	0.00056
MFOP2	MFOP	210077	315	0.00150
	Missions	584953	315	0.00054
	ASW	210077	97	0.00046
	ASUW	194091	124	0.00064
	SAR	180785	94	0.00052
MFOP3	MFOP	209988	343	0.00163
	Missions	579176	343	0.00059
	SAR	209988	90	0.00043
	ASW	194796	139	0.00071
	ASUW	174392	114	0.00065

These results show that there is little difference between the failure probabilities for MFOPs 1 and 2, but MFOP 3 has a slightly higher chance (0.00013) of failing. For MFOP1, there is little difference in the likelihood of total loss for each of the different missions. In MFOP2, however, the ASW mission has a fairly low failure probability, while ASUW has a fairly high value. In MFOP3, ASW and ASUW both have fairly high failure probabilities, while that for SAR is low.

Once again, however, there is no significant pattern or difference in any of the results which suggests that one MFOP or mission is a great problem more than any of the others. As such, all of the missions must be assessed equally for their absolute CF probability, and how this is affected by the phases within them.

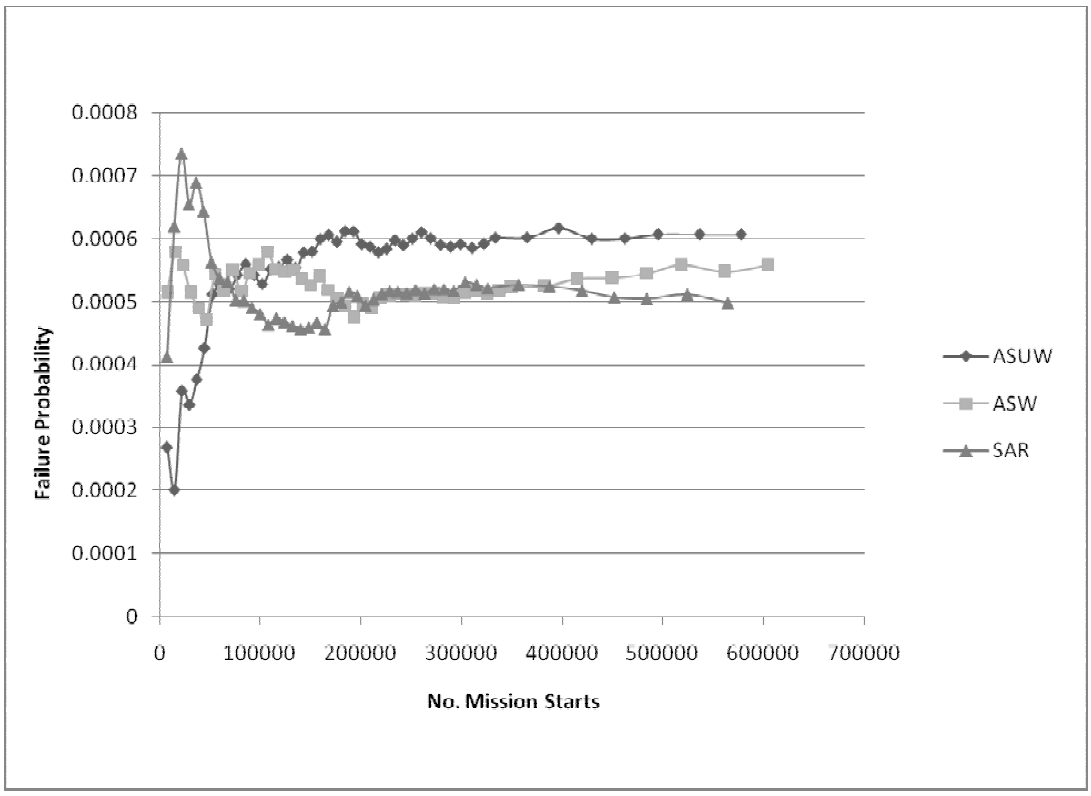


Figure 7.3 – Convergence of Mission Catastrophic Failure Probabilities

Figure 7.3 shows the convergence of the mission CF probabilities, which are shown numerically in Table 7.7. As can be seen, ASUW has the highest probability, followed by ASW and then SAR. However, there is still little difference in real terms between the failure probabilities – they are exceptionally low. In order to

understand the phases which are the common causes of failure, consider Figure 7.4 below.

Table 7.7 – Mission Catastrophic Failure Probabilities

Mission	Starts	No. CFs	CF Prob.
ASUW	578350	351	0.000607
ASW	604798	338	0.000559
SAR	564311	281	0.000498

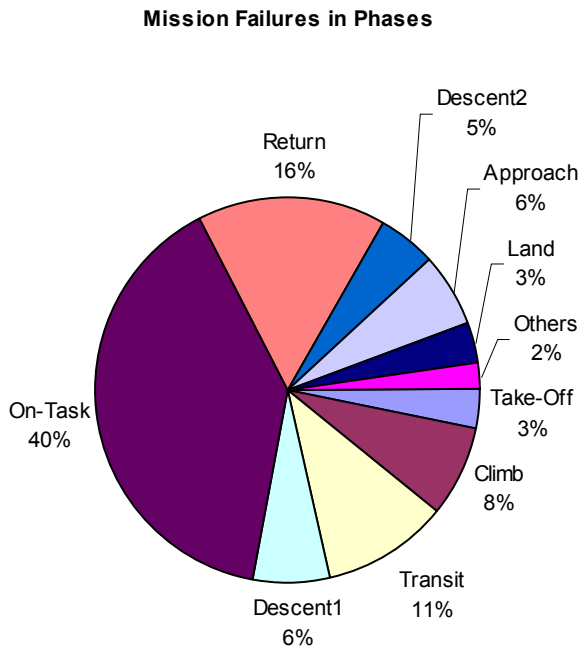


Figure 7.4 – Proportion of Mission Failures in Individual Phases

From Figure 7.4, it can be seen that the main contributors to mission failure are the phases On-Task, Return and Transit. These phases account for two-thirds of catastrophic failures of missions, and thus should be the focus of any attempts to improve platform CF probabilities.

The analysis took the most critical of the phases, On-Task, and looked at the systems which were causing its failure most often. The causes of these system failures were then analysed in turn, until the primary causes of failure were ascertained. For instance, the top event of the On-Task phase is “Aircraft 7”. The three subsystems of this are:

- 1.1-DEV_FP5 (failed in 20% of top event failures)
- 1.2-CFIT4 (40%)
- 1.3-STRCT7 (41%)

The results suggest that the latter two systems are more critical than the first (note that all gates are OR gates unless otherwise stated). Looking at 1.3-STRCT7, it has two events below it in the phase fault tree - 1.3.1STRC6 and 1.3.2FIRE7, which cause 74% and 26% of failures respectively. 1.3.1STRC6 has four subsystems:

- SD_GEN_SYS4 (16% of failures)
- G647-2 (84%)
- SD_LOW_ZHA (0%)
- SD-FCS (0%)

This shows that SD_LOW_ZHA and SD-FCS are not critical at all, while SD_GEN_SYS4 is much less important than G647-2. All of the failures of this gate are caused by SDSTOREDAM1, which has only one input (SDWEP1). SDWEP1 has two inputs, SD_WEPS1 and SD_NON_WEP, which cause two-thirds and one-third of its failures respectively. SD_NON_WEP has two component inputs which cause it to fail an equal number of times, while SD_WEPS1 has one subsystem and two component inputs. The two components, G023 and G618, have a modest effect on SD_WEPS1, causing 14% and 17% of failures respectively. The subsystem, SD_WEP_PL1, however, prompts SD_WEPS1 to fail 69% of the time.

Continuing this approach across the three phases earmarked for improvement yields a list of systems and components which are the main causes of total loss. This list is shown in Table 7.8, which shows the estimated phase criticality for the given item, and the method proposed to remedy the problem. The place of the items within the phase fault trees can be seen in Appendix B.1.

Table 7.8 – Items Most Critical for Total Loss of Aircraft

Item	Type	Phases Affected	Estimated Phase Criticality[1]			Remedy
			Transit	On-Task	Return	
G594	System	All Flight[2]	25%	12%	19%	Redundancy
G625	Component	All Flight	2%	1%	1%	Improve Inherent Reliability
NVEPITDISP & NVECDISP	Systems	All Flight	24%	14%	20%	Create LRI, SLR – 50h
NVMMRSYSERR	System	All Flight	10%	6%	8%	Redundancy[3]
CDD1EFISERR	Component	All Flight	14%	9%	12%	Prognostics
CDD2EFISERR	Component	All Flight	7%	4%	6%	Prognostics
CDERRPFD2HW	Component	All Flight	7%	5%	6%	Prognostics
WEP_LAUNCH	System	On-Task, Attack	-	15%	-	Redundancy
SD_NON_WEP	System	On-Task, Attack	-	9%	-	Redundancy
EXP_WEP	System	On-Task, Attack	-	4%	-	Abandon mission if either of two inputs fails
G621	Component	On-Task, Attack	-	6%	-	SLR – 70h
G642-GATE1	Component	Return	-	-	11%	Improve IR
G642-G178	Component	Return	-	-	14%	Improve IR
G274NMVTUPR	Component	All Flight	7%	3%	19%	Prognostics

7.3.2.2 Second Generation

The proposed remedies were applied to a second generation of simulations, as can be seen in Appendix B.2. A set of 76,800 simulations were performed with this new information, and results obtained in a similar fashion to those of the first generation. The convergence of the platform failure probabilities can be seen in Figure 7.5.

---

[1] The per cent figure given estimates how often, proportionally, the corresponding item contributed to the failure of the phase.

[2] Flight phases are Climb; Transit; Descent1; On-Task; Attack; Off-Task; Return; Descent2; Approach; Land.

[3] NVMMRSYSERR has two subsystems, NVM1SYSERR and NVM2SYSERR, the failure of either of which causes system failure. The redundancy puts in place a third sub-system, which comes online when either of the other two sub-systems fails.

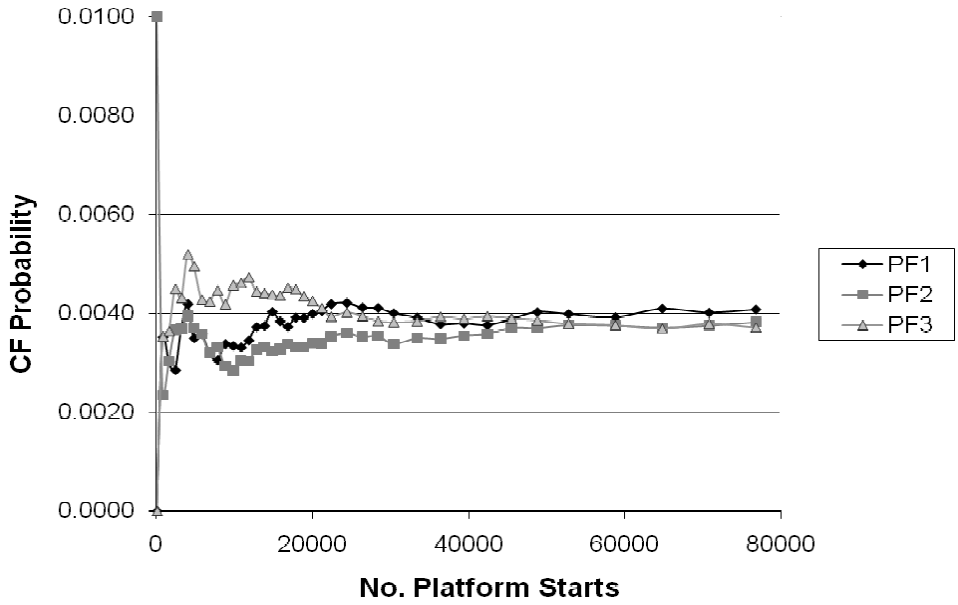


Figure 7.5 – Convergence of Platform CF Probabilities for Second Generation of Simulations

These results were first used to gain an understanding of how the remedies proposed in Table 7.8 have affected the catastrophic failure probability. Following this, a new analysis was performed on MFOP and mission abandonment, to see how this could be improved.

Table 7.9 compares the platform, MFOP and mission total loss probabilities from the first and second generations of simulations.

Table 7.9 – Reduction in CF Probabilities between Generations of Simulations

Item	1st Generation CF Prob.	2nd Generation CF Prob.	Relative Reduction
PF1	0.0046	0.0041	11.3%
PF2	0.0046	0.0038	16.7%
PF3	0.0046	0.0037	20.0%
MFOP1	0.0015	0.0013	12.6%
MFOP2	0.0015	0.0011	24.7%
MFOP3	0.0016	0.0015	11.1%
ASUW	0.00061	0.00048	20.5%
ASW	0.00056	0.00052	6.6%
SAR	0.00050	0.00039	21.7%

Each platform, MFOP and mission shows a modest reduction in the failure probability, which suggests that the remedies of the primary causes of total loss are having some effect. In order to show the level at which the MFOP enablers which have been employed are successful, a note was made of the improvement in reliability performance for each of the items:

- The prognostic components can be measured by how many times they failed overall, and how many times the prognostics system in place led to MFOP abandonment and component replacement.
    - CDD1EFISERR had 59 critical failures and was replaced by the prognostics system 299 times.
    - CDD2EFISERR had 33 critical failures and 168 replacements.
    - CDERRPFD2HW had 31 critical failures and 129 replacements.
    - G274NMVTUPR had no critical failures and was replaced 8 times.

  From this, it can be inferred that the prognostics system can prevent potential mission-critical component failures approximately 80% of the time.

- The effectiveness of the redundant items which have been put in place can be measured by finding the number of failures of the parent systems, and the number of times the redundancy switched or failed.
    - The system G594 was changed in the second generation of simulations, such that it became the parent of G594-MAIN and G594-RED. The latter is a carbon copy of the former, with redundant versions of the sub-components. With a sensed redundancy, 2393 switches were made, all of which were successful, and no failures of G594 occurred. However, results show that only 138 of these switches were made due to an actual failure of G594-MAIN. The NFF rate of the sensor was 1×10^{-4} hr^{-1}, as it was for all the other sensors. This value may be unrepresentative of a real-life situation, but does show the importance of reducing the NFF rate of sensors detecting system failures.
    - Similarly, SD_NON_WEP was converted to contain a main system and a redundant system, switched by a sensor. This had 2339

switches with a 100% success rate. This time, however, only 47 failures of the primary sub-system, SDNW_MAIN, occurred, with no failures of the redundant system.

- NVMMRSYS had manual switchovers to a redundant system. With this system, as mentioned in footnote 3 of Table 7.8, NVMMRSYS has two subsystems, NVM1SYSERR and NVM2SYSERR, and if either of these fails, a third sub-system, NVM3SYSERR, is activated. 66 switchovers took place, of which 61 were successful. On top of the five failed switches, two more failures of the overall system took place, due to NVM3SYSERR failing.
- WEP_LAUNCH was set up such that it was the main system in SD_WEP_PL1. WEP_LAUNCH failed 122 times, and 118 of the switches to the redundant item were successful. No failures of SD_WEP_PL1 took place.

These examples show the important differences between bringing online systems either automatically or manually. Manually-switched redundancies are more prone to human error not bringing the redundant item online. Automatic systems will very rarely fail to bring online the redundant item, but may well cause NFFs the majority of the time.

- Improvements in inherent reliability were applied to items G625, G642-GATE1 and G642-G178. In the first generation of simulations, these items failed 10, 22 and 22 times respectively (adjusted to compensate for the difference in the numbers of simulations performed). In the second generation, the items failed 3, 4 and 1 times respectively. These values reflect a very good improvement in the number of failures.

- NVEPITDISP and NVECDISP were put into an LRI, such that if one of the items failed, the other item was replaced at the same time. The idea behind this is that because a failure of either of these items will cause the parent system to fail, the replacement of the other child system should improve its reliability performance, hopefully bringing about a reduction in parent system reliability through a change in maintenance regime alone. In this

scenario, however, the introduction of a LRI seems to *increase* the number of failures of the parent system, G442/443 by more than three and a half times. A possible explanation for this is that if either of the items NVEPITDISP or NVECDISP has a long life before failure, the replacement of it as part of an LRI, before it has failed, leads to a good chance that the replacement component will have a shorter time to fail. Because each simulation regards the platform over a specified interval, this may make it less likely in any given simulation that either of the components could survive until the end of the three MFOPs, and more likely to cause parent system failure.

- A feature which may have exacerbated the problems mentioned in the previous point is that the LRI was scheduled to be replaced automatically after every 50 hours in service. Component G621 also had scheduled life replacements, every 70 hours, though these only had a modest effect, reducing the number of failures from 32 down to 25.

After focusing on the causes of total loss, the analysis moved on to MFOP and mission abandonments. The process begins in the same way as that for total loss. Figure 7.6 shows the convergence of the MFOP abandonment probabilities over the set of simulations, while Table 7.10 shows the numerical results. It can be seen that MFOPs 1 and 2 have an approximately equal abandonment probability, while MFOP 3 has a probability which is lower by about 0.075. An apparent reason for this is that the search and rescue mission, SAR, is the final mission in both MFOPs 1 and 2, and the "FlightCheck3" phase is abandoned a much larger percentage of the time the later in the MFOP it occurs. There is no such large increase in the abandonment probabilities for missions ASUW or ASW depending on position within the MFOP.

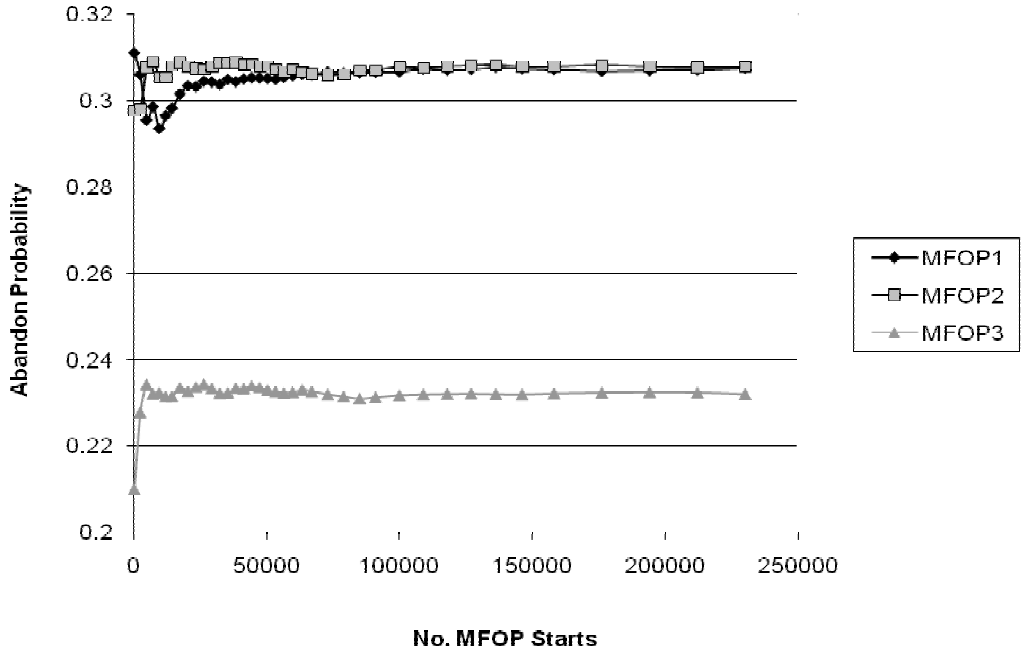


Figure 7.6 – Convergence of MFOP Abandonment Probability for Second Generation of Simulations

Table 7.10 – Abandonment Probabilities of MFOPs

MFOP	Category	No. Starts	No. Abandons	Abandon Prob.
MFOP1	MFOP	230030	70727	0.30747
	Missions	638859	70727	0.11071
	ASUW	230030	10828	0.04707
	ASW	219105	29262	0.13355
	SAR	189724	30637	0.16148
MFOP2	MFOP	230215	70835	0.30769
	Missions	640958	70835	0.11051
	ASW	230215	17567	0.07631
	ASUW	212558	14285	0.06721
	SAR	198185	38983	0.19670
MFOP3	MFOP	230136	53388	0.23198
	Missions	635609	53388	0.08400
	SAR	230136	16251	0.07061
	ASW	213804	22005	0.10292
	ASUW	191669	15132	0.07895

Table 7.11 shows the overall abandonment probabilities of the three different missions. As expected, SAR has the highest probability, of 13.9%, followed by ASW at 10.4% and ASUW at 6.3%. The phases in which these abandonments typically take place are shown in Figure 7.7. From this, it can be seen that the vast proportion (81.6%) of mission abandonments take place in the two phases "Flight

Check" and "On-Task". These, then, should be the focus of efforts to improve the mission abandonment figure.

Table 7.11 – Abandonment Probabilities of Missions

Mission	No. Starts	No. Abandons	Abandon Prob.
ASUW	634257	40245	0.063452
ASW	663124	68834	0.103803
SAR	618045	85871	0.138940

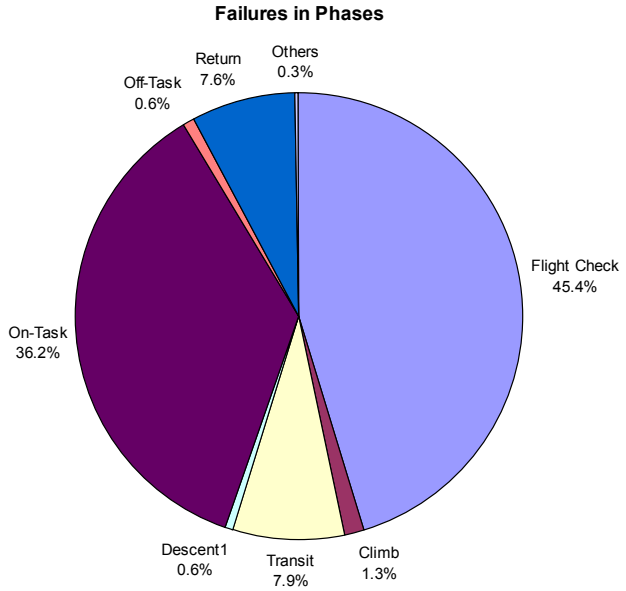


Figure 7.7 – Proportion of Mission Abandonments in Individual Phases

The method used in section 7.3.2.1 to establish the primary causes of MFOP abandonment was reapplied to create another list of systems and components. These items, their effect on phases On-Task and Flight Check and the remedies put in place to prevent their high criticalities are shown in Table 7.12.

Table 7.12 – Items Most Critical for Abandonment of MFOP

Item	Type	Sub-system	Estimated MFOP Criticality[4]	Remedy
APR	Component	DASS[5]	2.9%	Redundancy
SUPERHET	System	DASS	2.5%	Redundancy
DRR	Component	DASS	0.9%	Improve IR
CBR	Component	DASS	0.6%	Improve IR
TQG	Component	DASS	7.4%	Improve IR; Redundancy
LRA	Component	DASS	6.2%	Improve IR
MADCOMPAMP	Component	MAD[6]	3.0%	Redundancy
DETECTHEAD	Component	MAD	4.2%	Improve IR
SMGMTPROC	Component	STORES[7]	1.8%	Improve IR; Redundancy
SPINCHAN	System	ESM[8]	6.1%	Redundancy
MRX	Component	ESM	2.6%	Improve IR
REC-EXC	Component	RADAR-IFF[9]	4.4%	Redundancy
RFTRANS	Component	RADAR-IFF	5.5%	Improve IR; Redundancy
RFSCAN	Component	RADAR-IFF	6.2%	Improve IR; SLR – 80h
TUR1	Component	EOSDS[10]	8.3%	Improve IR; SLR – 80h
PDU1	Component	EOSDS	1.8%	Redundancy; LRI with Redundant Item
SCU1	Component	EOSDS	13.6%	Improve IR; SLR – 50h; Redundancy

[4] This value is the approximate proportion of MFOP abandonments contributed to by the specified item.

[5] DASS sub-system features in the following phases: All FlightCheck, Climb, Transit, Descent1, All On-Task, All Attack, Off-Task and Return.

[6] MAD sub-system features in phases FlightCheck1, On-Task1 and Attack1.

[7] STORES is used in phases FlightCheck1, FlightCheck2, Attack1 and Attack2.

[8] ESM is used in phases FlightCheck2, On-Task2 and Attack2.

[9] RADAR-IFF is used in FlightCheck2, FlightCheck3, On-Task2, On-Task3 and Attack2.

[10] EOSDS is used in FlightCheck3 and On-Task3.

In addition to the recommended improvements shown in Table 7.12, two more LRIs are to be created, containing:

- LRI_RWR – APR, SUPERHET, the redundancy for SUPERHET, and SIGNAL REC
- LRI_TRDS – DEC, TQG, DASSLNCH.

7.3.2.3 Third Generation

The remedies suggested were implemented in a third generation of simulations, as can be seen in Appendix B.3. 70,000 simulations were carried out, until all the results had a satisfactory level of convergence, as shown in Figure 7.8.

Table 7.13 shows a comparison between the different MFOP abandonment probabilities for the second and third generations.

Table 7.13 – Comparison of Abandonment Probabilities between Simulations

Item	2nd Generation Abandon Prob.	3rd Generation Abandon Prob.	Relative Reduction
MFOP1	0.307469	0.109258	64.5%
MFOP2	0.307691	0.106816	65.3%
MFOP3	0.231985	0.087392	62.3%
ASUW	0.063452	0.019814	68.8%
ASW	0.103803	0.045108	56.5%
SAR	0.138940	0.039231	71.8%

As can be seen from the results, the measures brought in to reduce the abandonment probabilities, shown in Table 7.12, have been very successful. While MFOPs are still abandoned around 11% of the time, this is a huge reduction from the previous value of between 23% and 31%. The typical relative probability reduction is nearly two-thirds for the MFOPs, and reaches a value of nearly three quarters in the case of mission SAR.

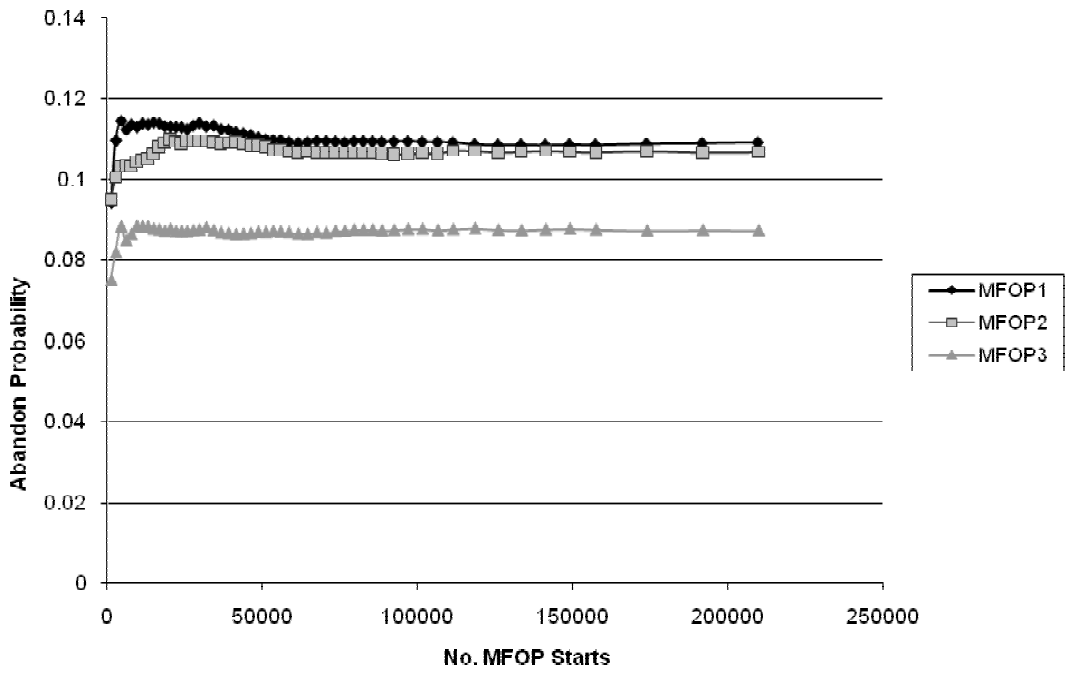


Figure 7.8 – Convergence of MFOP Abandonment Probability

The success of the various remedies can be approximated, although in many cases items had several remedies put in place at the same time, so the effect of any one is difficult to ascertain.

- The failure rate of many of the components was reduced to a tenth of its previous value. This resulted in a large reduction in the number of failures for each of these components, as shown in Table 7.14. Note that in this Table, the number of failures for the second generation has been adjusted to account for the different numbers of simulations performed. A vast reduction in the number of failures per item can be seen, which plays a large role in the reduction of MFOP abandonments.

Table 7.14 – Comparison of Component Failure Numbers

Component	2nd Generation Failures	3rd Generation Failures
CBR	1177	19
DRR	1645	39
TQG	12917	331
LRA	10844	247
DETECTHEAD	12644	304
SMGMTPROC	6066	122
MRX	5002	117
RFSCAN	14087	328
RFTRANS	12643	307
TUR1	22237	541
SCU1	36138	874

- The three new LRIs can have their performance measured based on the number of failures of the parent system:
    - LRI_RWR contains the items for which RADWARNR is the parent. The number of failures of RADWARNR reduced by 90%, though it is difficult to say how much of this can be ascribed to the LRI.
    - Similarly, LRI_TRDS contained the sub-items for TRDS. TRDS failed 67% less in the third generation than the second.
    - The problem, mentioned in section 7.3.2.2, of LRIs replacing long-life items with shorter-lived components, reared its head with PDU1. In the third generation, PDU1 failed 8% more often than in the second generation.

- Redundancies were used regularly in the third generation of simulations. A way of measuring the performance of the redundancies is to look at the reduction in the number of parent system failures over the set of simulations. Table 7.15 shows these values, in addition to other parameters concerning redundancies, such as No Faults Found and the number of failed switches. The "Type" column refers to whether a redundancy is sensed (S) or manual (M).

Table 7.15 – Reduction of Failures due to Redundancies

Item	Type	No. Switches	NFFs	No. Failed Switches	Parent System Failures		Reduction
					2nd Gen	3rd Gen	
APR	S	8417	2423	0	13340	1276	90.4%
TQG	S	3798	2377	0	32600	10769	67.0%
MADCOMPAMP	S	12485	2287	0	22889	2839	87.6%
SMGMTPROC	S	3031	2385	0	6083	26	99.6%
REC-EXC	S	13472	2215	0	65071	33792	48.1%
RFTRANS	S	3734	2372	0	65071	33792	48.1%
PDU1	M	5075	0	490	63372	4686	92.6%
SCU1	S	6312	2238	0	63372	4686	92.6%
SUPERHET	M	5094	0	484	13340	1276	90.4%
SPINCHAN	M	12836	0	1317	16789	1959	88.3%

The Table shows that each of the redundancies has had an effect on the number of parent system failures, with the lowest reduction being just under half, for REC-EXC and RFTRANS. The failures for the parent system of SMGMTPROC, SMGMTSYS, were reduced to almost zero.

The NFF rate was the same for the sensors used in the third generation as for the second. Here, however, while the number of NFFs is around the same, it has less impact as the number of switches due to a real fault is typically much larger than those for catastrophic failures. Nonetheless, a great deal of abandoned missions would be prevent through the improvement of the NFF rate for sensors controlling redundancy switchovers.

Ultimately, the main point of these improvements to the system design is to decrease the number of MFOPs that are abandoned. Table 7.16 shows a comparison between the MFOP failure probabilities (for either abandonment or total loss) in the first set of simulations, and those for the third set.

Table 7.16 – MFOP Failure Probability Comparison

MFOP	1st Generation Failure Prob.	3rd Generation Failure Prob.
MFOP1	30.7%	11.1%
MFOP2	31.0%	10.8%
MFOP3	23.6%	8.9%

As the table shows, the probabilities of failure of the three different MFOPs, which were so problematic for the first generation, have been reduced to a much more satisfactory level in the third generation. Further improvements could be made, potentially, to reduce this value even more.

7.4 Summary

The modelling method presented in Chapter 6 and implemented in the form of a simulation program (as explained in Appendix A), has been applied to several systems in this chapter.

First, the model was applied to those systems shown in Chapter 5 for the simple modelling method, in order to demonstrate it is capable of producing the same values. The success of this led to the model being applied to a much larger, more realistic system. A method by which the model could be used to investigate the effectiveness of different strategies to try to decrease the MFOP failure probability is presented. Through the use of various MFOP enablers, the MFOP failure probabilities were able to be reduced from around 30% to around 10%. Furthermore, the modelling method was able to give more detailed information regarding the performance of the system, such as suggesting that use of LRIs may increase the expected number of failures of components.

The ability to investigate the application of many different reliability technologies to a real-world system is one which provides a great deal of value. It has great potential to be used in the design stage of a new platform, to find the most effective (both in terms of reliability and cost) ways of increasing its availability.

Chapter 8 Conclusions and Further Work

8.1 Summary

The concept of maintenance-free operating periods (MFOPs) has been introduced as a length of time in which no maintenance other than simple replenishment of fuel, stores, and so on can take place. A system performing an MFOP must complete all of the missions scheduled within that time without incurring any failures which prevent any further operation with a high, specified, likelihood. Once an MFOP is completed, a maintenance recovery period (MRP) takes place. During this time, any sub-system failures which occurred during the mission are fixed, while inspections and tests ensure that no part of the system is likely to cause failure during the forthcoming MFOP.

MFOPs were introduced in order to help plan platform operations and the support resources required. It also attempts to make reliability concepts more appropriate for the end user. The ability to have a high degree of confidence that a platform will be able to carry out all assigned tasks, before undergoing highly organised and scheduled maintenance, is an ideal future situation. Previous research has indicated that the most effective way to provide a high-value, high-confidence MFOP is through the use of technologies such as prognostics, which aim to accurately predict future failure; diagnostics, which identify and locate current failures; and redundant systems to replace the functionality of failed systems.

A second concept, phased missions, has also been demonstrated. A phased mission is one where a system operates to complete a sequential set of objectives over different time intervals. The system will use different sub-systems at different times in order to complete each of the individual objectives, and so will have a failure logic model which changes with each of the phases.

The probability of mission failure cannot be found by simply multiplying the phase unreliabilities, because at phase change times, the system must be in a state which allows it to function in both the exiting and the entering phase. This statistical

dependence requires a more complex analysis method than multiplying phase unreliabilities.

Several existing techniques are commonly used to solve the phased missions scenario, such as fault tree analysis, Markov methods and binary decision diagrams (BDDs). However, drawbacks of these methods include factors such as an explosion in the size of a Markov model as the system complexity increases, or the inability of fault trees to consider complicating factors such as component dependencies. Other complexities of phased missions include the insertion of a phase within the mission, following the failure of an item within the system; non-deterministic phase durations; phase choices, where the subsequent phase is selected from a group, based on random events or the current system state; and the two different failure modes of mission abandonment and catastrophic failure.

The twin concepts of phased missions and MFOPs combine well, such that an MFOP can be redefined to be a period of a certain number of missions in a given order, during which there is a high, specified, probability that none of the missions can fail such that emergency maintenance is required or the system is lost. In order to consider all of the factors which would affect the performance of MFOPs with phased missions, a modelling method is required which is more powerful than either fault trees, BDDs or Markov methods.

A new method of modelling MFOPs with the phased mission situation has been proposed. This uses Petri nets (PNs), a digraph with two types of node (places and transitions), which allow the storage and transmission of data, in the form of "tokens". The position and number of tokens within the various places in the net at any one time represents the currently modelled system state. This simple, yet powerful, method is capable of denoting many possible situations.

The new modelling method consists of three different types of net:

- A component PN (CPN), which contains all of the data relating to component states, such as "working", "failed", "replacing", and so on, and

the transitions to switch between these states. The CPN also contains information on whether the platform is undergoing repair or not.

- A set of phase PNs (PPNs), which are representations of phase fault trees. These take inputs from component failure places, and combine these to model whether the overall platform has failed in any phase. Failure consists of two possible modes: total loss and abandonment. As such, each PPN typically contains two PN fault trees.

- A master PN (MPN). This contains the models of each of the platforms, the MFOPs they have to perform, the missions within the MFOPs, and the phases within the missions. If a phase failure top event occurs while that phase is active, the MPN models mission and MFOP failure (either catastrophic or abandoned).

The model can consider a fleet of identical platforms, through the use of "coloured" tokens. One colour is considered completely independently of another, and in this application there is no possible interaction between tokens of different colours. The use of coloured tokens significantly reduces the overall size of the PN model, as one set of PPNs, CPN and MPN can be used for the whole fleet. Each different platform performs a different sequence of MFOPs, after which the modelling ends.

The various complexities of both MFOPs and phased missions are modelled in PN form. For instance, it is possible that a certain combination of failures may occur such that the mission always fails, regardless of which phase it is in. These can be described as general failures and can be stored under a General PPN, again in the form of either abandonment or total loss.

Phase choices can take place through a small alteration of the PN mechanism: "probability arcs" allow a single transition to output to just one of a selection of possible output places, according to a given probability for each. As such, a probabilistic phase choice can take place, while phases can also be chosen based on the current system state. These event-driven phase choices can be made to occur

based on a specified conditional fault tree, stored within the relevant PPN, where the presence or absence of the top event alters which phase is chosen to perform next.

It is considered unrealistic for an abandoned mission to end instantly – for example, an aircraft in mid-flight is unable to immediately appear at the base after aborting. Instead, the aircraft must go through a series of phases which are different from those it was intending to perform. As such, the model allows emergency phases to be carried out in the event of mission abandonment, ending with MFOP abortion and an unscheduled MRP. As part of this, it is possible for a phase's duration to mirror that of another phase, such that, for instance, the length of a "return" phase is exactly equal to that of a "transit" phase, regardless of whether the earlier phase was cut short.

Another feasible scenario is one where a phase must be inserted into the mission, before continuing with the previous objectives. An example of this is in-flight refuelling, something which is commonly performed with long-range aircraft such as Nimrod. The PN model considers this to taking place when the top event of a conditional fault tree has occurred. The phase is immediately carried out, interrupting the previous phase, which continues where it left off.

An event may occur whose appearance cannot be described according to a time-based probability distribution, but as in response to the occurrence of another event, with a certain probability. These are here termed "discrete events", and are modelled in PN form using probability arcs to propagate a trigger event (which may be, for example, the start of a mission or phase, a component failure, and so on) to either "event occurs" or "event does not occur", based on a randomly sampled value and the assigned probabilities. These are useful to represent factors such as weather, human responses to stimuli or preventable maintenance issues (such as failing to replace a panel, for instance), and can be used as inputs to PPNs.

It is common practice within many industries to remove a group of components in close physical proximity as one, due to the failure of one or more of them. These are termed "Line Replaceable Items" (LRIs). The modelling method allows components to be replaced together regardless of their operational state, with new failure times

being sampled following replacement. These replacements can also be scheduled to occur after a certain length of time in flight, so as to prevent a serious failure event occurring.

If a failure event does occur, its occurrence and location may need to be found for a variety of reasons. For instance, it may not be repairable until the failure is seen, or the failure may require a redundant system to be brought online. These diagnostics are here modelled in two forms: sensors, and inspections. Sensors are physical components which can themselves fail, but if they are operational, will immediately detect a failure in the component or system they are monitoring. Another issue is that of No Faults Found (NFFs) – the sensor detects a main system failure where none has occurred. These spurious trips can be a large problem where the sensor, for instance, causes mission abandonment upon failure detection. A NFF rate can be specified, modelling NFFs and their effect on the system. Inspections, however, are performed only within MRPs, and can, for instance, prompt item replacement if a failure is found. Inspections can be set to be performed after a certain length of time, after a given number of MFOPs, or in response to a discovered failure in a higher system.

If a component or system fails, it may be necessary to bring a previously non-operational item online to replace its function. These items are called redundant, and can be useful, albeit with a penalty of cost, bulk or weight. The redundant systems are modelled as being brought online either manually or automatically. Automatic transition occurs through the use of sensors. As mentioned above, these can fail and thus not bring the redundant system online. Also, NFFs may take the main system offline where no failure has occurred. Manual transition is carried out by an operator, and as such is subject to human error. A success rate of the transition taking place is specified, and so redundant items will fail to be brought online a certain proportion of the time. However, spurious trips will not occur with manual transitions.

Prognostic systems aim to predict the future failure of components, which systems will be affected by this and whether any phases and missions will potentially fail. A modelling method for this has been proposed based on the concept of levels of wear.

It is assumed that components degrade, that this wearing is detectable and can be categorised into levels of wear. It is further assumed that the levels of wear occur at certain stages within the life of the component, expressed as a percentage range of its failure time. Once a component has reached a level of wear at a certain time, predictions can be made as to how much longer it may have before it fails. Predictions of the length of the current or future missions can then be made, and the effect of the component failure upon the mission estimated. If a current mission is predicted to fail, it can be abandoned and the failing component replaced. During an MRP, if any mission is predicted to fail, components can be replaced to prevent this.

All of these considerations have been represented in PN form, and designed to take into account other complications (such as components which are part of an LRI undergoing inspections). The modelling method has been applied in the form of a program, which uses a Monte-Carlo simulation approach to randomly sample transition switching times for components, phases, and so on. These timed transitions are switched, followed by any enabled immediate transitions, until each simulation ends. If enough simulations are performed, an accurate picture of the performance of the platform can be seen. The program has been used with a real-life system to establish whether it is a useful tool and that the modelling method works as intended, to provide valuable information at the design stage on the MFOP performance of a platform or fleet of platforms.

8.2 Conclusions

The aim of the research was to produce a modelling method capable of considering the two concepts of MFOPs and phased missions in-depth, and giving important data on whether a system undergoing these is performing as required. The following conclusions can be made:

- A full-scale, in-depth model capable of considering both MFOPs and phased missions has not been previously produced. The modelling method shown in this thesis is novel, and can provide valuable information on a new reliability metric which has not seen widespread use in any industry to date.

- While Petri nets have previously been applied to phased missions, there has not been, to date, a method of providing models with a consistent structure and mode of creation. This thesis has built upon previous works in order to provide a way of modelling phased missions which is easily understood and can be repeatedly applied to different scenarios.

- An adaptation of the Petri net has been suggested, clarifying the situation where two immediate or identically-timed transitions are enabled from the same place, where only one transition can take the token within it. Instead, these transitions are replaced by a single one, which has "probability arcs" leading to output places. Only one of the places connected to a probability arc will take the token when the transition switches, according to a randomly sampled number.

- Methods of modelling the complexities of phased missions, such as abandonment to phases, phase insertion, phase selection, and so on, have been produced in the form of Petri nets. Even outside the scope of MFOP, these have uses in the form of a simulation program.

- The enablers of MFOP, such as redundancy, prognostics, inspection, automated sensing, line replaceable items, and so on, have all been considered within the model shown. The methods of modelling these in PN form are new, and may have uses outside of the scope of this thesis.

- In particular, a prognostic PN model has been produced which has a wear-based method of predicting future component failure. This is then combined with phase fault trees to establish whether a component likely to fail before the mission or MFOP has ended will cause a critical system failure. If so, the current MFOP can be abandoned or, if the prognostic system is being used during an MRP, the component can be replaced.

- The suggested model has been implemented in the form of a Monte-Carlo simulation program. This takes inputs such as phase fault trees, component

failure data, sensors, discrete events, MFOP enablers, and fleet, MFOP, mission and phase data. This program is capable of providing a great deal of information regarding the implementation of various MFOP enablers, mission structures, abandonment and maintenance regimes, and so on. It is hoped that this will be a useful tool to designers considering reliability issues, in order to find the most effective ways of having a high-value MFOP with a high degree of confidence. It is possible that it may provide data which is counter-intuitive, such as LRIs potentially increasing the expected number of failures of a sub-component within a certain timeframe. As a result, it is of great value to understand how MFOPs may be implemented in real-world systems.

8.3 Further Work

The scope of this research leads to the possibility of further areas of investigation. Potential directions are discussed in the following sections.

8.3.1 Verification of Repairable Larger System Results

A larger system was modelled as described in section 5.3.3. As the system and mission profile used in that example was so complex, the results from the simulations could not be compared to theoretical figures. It would be worthwhile for mathematical values to be produced in order to compare these with the results from the simulations.

8.3.2 Cost modelling

While the model shown here considers many of the important areas of both MFOPs and phased missions, a factor absent from this is the costs associated with various maintenance regimes, employment of manpower, total loss of platform, cost of phase insertion or abandonment, and so on. It would be useful to expand the research to include cost factors, and potentially other important considerations.

8.3.3 Reconfigurability

There is no direct consideration of reconfigurability outside of the scope of redundant systems. While the model is capable of handling many different variations of redundancies, such as several systems sharing a single redundant output, a redundancy itself having a redundancy, or a single system having several redundancies, it is possible that there are further complications specific to reconfigurable systems which require direct modelling.

8.3.4 Phase and Component Dependency

While the model takes account of components which affect each others' failure rates, there is no consideration within the model of how different phases may affect component failure rates, or how the failure of components or system may lengthen or shorten a phase's duration. It is probable, for instance, that a jet engine is more likely to fail mid-flight than when the aircraft is idling. Similarly, if one of several jet engines fails mid-flight, it is likely that the aircraft will take more time to return to base.

8.3.5 Degradation

The model has a very binary approach to component failures. In reality, a common issue is that the performance of components gradually reduces over time. This may manifest itself by reducing system effectiveness, lengthening phases or possibly affecting phase choice. A realistic approach would be to allow the wear-levels of components as shown in the prognostic model to affect the outcome of systems or phases.

8.3.6 Prognostics

The prognostics model used within this research has been necessarily simplified, in order for it to be included as part of a broader whole. It is certainly possible that there are much more complex ways of predicting component failures and their effects on systems, or that several approaches need to be all considered at the same time within a single model. A broader, more accurate prognostic model may be capable of being produced.

8.3.7 Platforms and MFOPs

The model restricts the number of MFOPs that a platform may perform, and in doing so can only provide an observation of the platform over a relatively short period of time. The ability to consider many more MFOPs, potentially an unlimited number, over the full lifecycle of a platform, would be a useful feature. This would lend itself well to consideration of whole life cost.

8.3.8 MFOPs and Missions

The results from section 7.3.2 provided an indication that the placement of a mission within an MFOP may significantly affect the probability of the MFOP failing, especially where the failure is categorised as an abandonment. It would be useful to investigate the extent of this dependency on mission order, to find if there is an easy way to establish the optimum MFOP survivability figure. An example of how this might be carried out would be to have eight MFOPs, each with seven instances of “Mission 1”, and one instance of “Mission 2”. “Mission 2” would be in a different location in each MFOP, and the resulting failure rates for each of the MFOPs would be compared.

8.3.9 Maintenance & Logistics Support

The maintenance regimes within the model shown here are simplified. In the real world, the maintenance of the platforms will be an issue which is critical to how they then go on to perform MFOPs. Factors such as availability of spares or manpower, operation of different bases, repair only of necessary components or systems in a "just in time" capacity, potential for inspection to fail to find a present fault, and so on, could all be modelled more accurately, providing a much better picture of how maintenance issues could affect MFOP performance.

8.3.10 Importance Measures

An important aspect of the modelling capability is the necessity of indicating which MFOPs, missions, phases, systems, components and so on are the most important. The ability to give various types of importance measure for each of these items, for both catastrophic failure and abandonment, would be useful. Furthermore, the ability to indicate the effectiveness of a particular enabler, such as a redundancy, would provide a valuable indicator.

8.3.11 Optimisation

The fault trees that the model uses are converted to PN form in such a way that they exactly represent the original fault tree. There are, however, many optimisation techniques which exist in order to reduce the size of fault trees and thus the number of calculations required to find the top event occurrence probability. These same optimisation techniques would reduce the size of the phase PNs and speed up the simulations.

8.3.12 System of Systems

A recent development of reliability within the military environment is that of systems of systems. The approach considers several distinct types of system working together to achieve a common mission. A particular system may provide a valuable function at an early part of the mission, which has a given timeframe to complete, otherwise another system cannot perform its function. This interaction of different systems is complex to model. It is probable that the phased mission and MFOP concepts may fit comfortably within this area. Similarly, a PN approach may be able to consider all factors necessary to modelling each mission. An investigation into how these issues may work with the system of systems concept would be useful.

References

[1] Hazardous Installations Directorate, "HID's Approach to ALARP Decisions", *Technical Note SPC/Permissioning/09,* 2002.

[2] Andrews, J.D., and Moss, T.R., "Reliability and Risk Assessment, Second Edition", Professional Engineering Publishing Limited, Bury St. Edmunds, UK, 2002, pp. 540.

[3] Henley, E.J., and Kumamoto, H., "Probabilistic Risk Assessment", IEEE Press, 1992.

[4] Watson, H.A., "Section VII", *Launch Control Safety Study,* Vol. 1, Bell Telephone Laboratories, Murray Hill, NJ, USA, 1961.

[5] Vesely, W.E., "A Time-Dependent Methodology for Fault Tree Evaluation", *Nuclear Design and Engineering,* Vol. 13, 1970, pp. 337-360.

[6] Esary, J.D., and Ziehms, H., "Reliability Analysis of Phased Missions", *Reliability and Fault-Tree Analysis,* 1975, pp. 213-236.

[7] Dinesh Kumar, U., Knezevic, J., and Crocker, J., "Maintenance free operating period - an alternative measure to MTBF and failure rate for specifying reliability?" *Reliability Engineering and System Safety,* Vol. 64, 1999, pp. 127-131.

[8] Fussell, J.B., "How to Hand Calculate System Reliability and Safety Characteristics", *IEEE Transactions on Reliability,* Vol. R-24, No. 3, 1975, pp. 169-174.

[9] Rauzy, A., "New Algorithms for Fault Tree Analysis", *Reliability Engineering and System Safety,* Vol. 40, 1993, pp. 203-211.

[10] Sinnamon, R.M., and Andrews, J.D., "Improved Efficiency in Qualitative Fault Tree Analysis", *Quality and Reliability Engineering International.,* Vol. 13, No. 5, 1997, pp. 293-298.

[11] Sinnamon, R.M., and Andrews, J.D., "Improved Accuracy in Quantitative Fault Tree Analysis", *Quality and Reliability Engineering International.,* Vol. 13, No. 5, 1997, pp. 285-292.

[12] Appleton, D.P., "Future Offensive Aircraft - maintenance free operating periods", *Proceedings of the R, M & T for Future Projects Seminar,* Northumberland House, London, 8 November 1996.

[13] Appleton, D.P., “An Alternative Approach to Setting Reliability and Maintainability Requirements for Combat Aircraft”, *Advances in Safety and Reliability: ESREL '97, Lisbon,* Pergamon, Oxford, UK, 1997, pp. 923-930.

[14] Turner, T., Hockley, C., and Burdaky, R., “The Customer Needs A Maintenance-Free Operating Period”, *1997 Avionics Conference and Exhibition,* Vol. 97, ERA Technology, 1997, pp. 2.2.1-2.2.10.

[15] Knowles, D.I., “Should We Move Away from “Acceptable Failure Rate”?” *Communications in Reliability, Maintainability and Supportability,* Vol. 2, No. 1, 1995, pp. 23-28.

[16] Crocker, J., “Maintenance Free Operating Period - Is this the way forward?” *7th International Symposium, University of Exeter: Centre for Management of Industrial Reliability, Cost and Effectiveness,* University of Exeter, 1997, pp. 96-102.

[17] Dinesh Kumar, U., “New trends in aircraft reliability and maintenance measures”, *Journal of Quality in Maintenance Engineering,* Vol. 5, No. 4, 1999, pp. 287-295.

[18] Hockley, C.J., and Appleton, D.P., “Setting the requirements for the Royal Air Force's next generation aircraft”, *Annual Reliability and Maintainability Symposium 1997,* IEEE, New York, NY, USA, 1997, pp. 44-49.

[19] Hockley, C.J., “Design for success”, *Proceedings - Institution of Mechanical Engineers,* Vol. 212, No. G, 1998, pp. 371-378.

[20] Cini, P.F., and Griffith, P., “Designing for MFOP. Towards the Autonomous Aircraft - Achieving Customer Expectations”, *Proceedings of 1999 Avionics Conference and Exhibition;* Vol. 99-0815, ERA Technology, Leatherhead, UK, 1999, pp. 7.1.1-7.1.13.

[21] Newton, C.O., “The potential impact of MFOPs (maintenance free operating periods) on the reliability of electronic systems”, *Proceedings of 1999 Avionics Conference and Exhibition,* Vol. ERA REPORT NO 99-0815, ERA Technology, Leatherhead, UK, 1999, pp. 9.1.1-9.1.9.

[22] Murray, A.J.F., “Durability”, 2003, BAE Systems Warton (Technical Note).

[23] Warrington, L., Jones, J.A., and Davis, N., “Modelling of maintenance, within discrete event simulation”, *48th Reliability and Maintainability Symposium (RAMS);* IEEE, Piscataway, NJ, 2002, pp. 260-265.

[24] Relf, M.N., “Maintenance-Free Operating Periods - The Designer's Challenge”, *Quality and Reliability Engineering International,* Vol. 15, 1999, pp. 111-116.

[25] Dagg, R.A., and Newby, M., “Optimal overhaul intervals with imperfect inspection and repair”, *IMA Journal of Mathematics Applied in Business and Industry,* Vol. 9, No. 4, 1998, pp. 381-91.

[26] Jones, J.A., Warrington, L., and Davis, N., “Integrated modelling of system functional, maintenance and environmental factors”, *48th Reliability and Maintainability Symposium (RAMS);* IEEE, Piscataway, NJ, 2002, pp. 399-403.

[27] Todinov, M.T., “A new reliability measure based on specified minimum distances before the locations of random variables in a finite interval”, *Reliability Engineering & System Safety,* Vol. 86, 2004, pp. 95-103.

[28] Burdick, G.R., Fussell, J.B., Rasmuson, D.M., “Phased mission analysis: a review of new developments and an application”, *IEEE Transactions on Reliability,* Vol. R-26, 1977, pp. 43-49.

[29] Veatch, M.H., “Reliability of periodic, coherent, binary systems”, *IEEE Transactions on Reliability,* Vol. R-35, No. 5, 1986, pp. 504-35.

[30] Montague, D.F., and Fussell, J.B., “A methodology for calculating the expected number of failures of a system undergoing a phased mission”, *Nuclear Science and Engineering,* Vol. 74, No. 3, 1980, pp. 199-209.

[31] Dazhi, X., and Xiaozhong, W., “A practical approach for phase mission analysis”, *Reliability Engineering & System Safety,* Vol. 25, No. 4, 1989, pp. 333-47.

[32] Kohda, T., Wada, M., and Inoue, K., “A simple method for phased mission analysis”, *Reliability Engineering & System Safety,* Vol. 45, No. 3, 1994, pp. 299-309.

[33] Somani, A.K., and Trivedi, K.S., “Boolean Algebraic Methods for Phased-Mission System Analysis”, *Proceedings of Sigmetrics,* 1994, pp. 98-107.

[34] Y, M., and K.S, T., “An algorithm for reliability analysis of phased-mission systems”, *Reliability Engineering & System Safety,* Vol. 66, No. 2, 1999, pp. 157-70.

[35] Zang, X., Sun, N., and K.S, T., “A BDD-based algorithm for reliability analysis of phased-mission systems”, *IEEE Transactions on Reliability,* Vol. 48, No. 1, 1999, pp. 50-60.

[36] Xing, L., and Dugan, J.B., “Comments on PMS BDD generation in `A BDD-based algorithm for Reliability Analysis of phased-mission systems'“, *IEEE Transactions on Reliability,* Vol. 53, No. 2, 2004, pp. 169-73.

[37] Xing, L., and Dugan, J.B., “A separable ternary decision diagram based analysis of generalized phased-mission reliability”, *IEEE Transactions on Reliability,* Vol. 53, No. 2, 2004, pp. 174-84.

[38] Dunnett, S.J., and Andrews, J.D., “A binary decision diagram method for phased mission analysis of non-repairable systems”, *Proceedings of the Institution of Mechanical Engineers, Part O : Journal of Risk and Reliability,* Vol. 220, No. 2, 2006, pp. 93-104.

[39] Prescott, D.R., Andrews, J.D., and Downes, C.G., “Multiplatform phased mission reliability modelling for mission planning”, *Proceedings of the Institution of Mechanical Engineers, Part O : Journal of Risk and Reliability,* Vol. 223, 2009, pp. 27-39.

[40] Prescott, D.R., Remenyte-Prescott, R., and Andrews, J.D., “A systems reliability approach to decision making in autonomous multi-platform systems operating a phased mission”, *Proceedings of the Annual Reliability, Availability and Maintainability Symposium (RAMS)*, Las Vegas, USA, IEEE, Jan 2008, pp. 8-14.

[41] Prescott, D.R., Remenyte-Prescott, R., Reed, S., “A reliability analysis method using binary decision diagrams in phased mission planning”, *Proceedings of the Institution of Mechanical Engineers, Part O : Journal of Risk and Reliability,* Vol. 223, 2009, pp. 133-143.

[42] La Band, R.A., and Andrews, J.D., “Phased mission modelling using fault tree analysis”, *Proc. of the 15th Advances in Reliability Technology Symposium (ARTS),* Mech. Eng. Publications for IMechE, 2003, pp. 81-97.

[43] Andrews, J.D., “Identifying the major contributions to risk in phased missions", 2006, *Proceedings of the Annual Reliability, Availability and Maintainability Symposium (RAMS)*, 2006, IEEE, pp. 624-629.

[44] Andrews, J.D., “A ternary decision diagram method to calculate the component contributions to the failure of systems undergoing phased missions”, *Proceedings of the Institution of Mechanical Engineers, Part O : Journal of Risk and Reliability,* Vol. 222, No. 2, 2008, pp. 173-187.

[45] Dugan, J.B., "Automated analysis of phased-mission reliability", *IEEE Transactions on Reliability,* Vol. 40, No. 1, 1991, pp. 45-52.

[46] Somani, A.K., "Simplified Phased-Mission System Analysis for Systems with Independent Component Repairs", *International Journal of Reliability, Quality and Safety Engineering,* Vol. 4, 1997, pp. 167-191.

[47] Vaurio, J.K., "Fault tree analysis of phased mission systems with repairable and non-repairable components", *Reliability Engineering & System Safety.,* Vol. 74, No. 2, 2001, pp. 169-180.

[48] Clarotti, C.A., Contini, S., and Somma, R., "Repairable Multiphase Systems - Markov and Fault-Tree Approaches for Reliability Evaluation", edited by G. Apostolakis S. Garribba and G. Volta, *Synthesis and Analysis Methods for Safety and Reliability Studies*, Plenum Press, New York, 1980, pp. 45-58.

[49] Gray, J.N.P., "Continuous-time Markov methods in the solution of practical reliability problems", *Reliability Engineering,* Vol. 11, No. 4, 1985, pp. 233-252.

[50] Wells, C.E., and Bryant, J.L., "Reliability characteristics of a Markov system with a mission of random duration", *IEEE Transactions on Reliability,* Vol. R-34, No. 4, 1985, pp. 393-396.

[51] Alam, M., and Al-Saggaf, U.M., "Quantitative reliability evaluation of repairable phased-mission systems using Markov approach", *IEEE Transactions on Reliability,* Vol. 35, No. 5, 1986, pp. 498-503.

[52] Kim, K., and Park, K.S., "Phased-mission system reliability under Markov environment", *IEEE Transactions on Reliability,* Vol. 43, No. 2, 1994, pp. 301-309.

[53] Somani, A.K., Ritcey, J.A., and Au, S.H.L., "Computationally-Efficient Phased-Mission Reliability Analysis for Systems with Variable Configurations", *IEEE Transactions on Reliability,* Vol. 41, No. 4, 1992, pp. 504-511.

[54] Mura, I., and Bondavalli, A., "Hierarchical modeling and evaluation of phased-mission systems", *IEEE Transactions on Reliability,* Vol. 48, No. 4, 1999, pp. 360-8.

[55] Mura, I., Bondavalli, A., Zang, X., "Dependability Modeling and Evaluation of Phased Mission Systems: A DSPN Approach", *Proc. International Federation for Information Processing International Conference on Dependable Computing And Fault Tolerant Systems (DCCA-7),* Vol. 12, IEEE Computer Society Press, 1999, pp. 319-338.

[56] Smotherman, M., and Zemoudeh, K., "A non-homogeneous Markov model for phased-mission reliability analysis", *IEEE Transactions on Reliability,* Vol. 33, No. 5, 1989, pp. 585-590.

[57] Smotherman, M.K., and Geist, R.M., "Phased mission effectiveness using a nonhomogeneous Markov reward model", *Reliability Engineering and System Safety,* Vol. 27, 1990, pp. 241-255.

[58] Petri, C.A., "Kommunikation mit automaten", PhD Thesis, 1962.

[59] Schneeweiss, W.G., "Design of Petri-Nets (for Reliability Engineering) [Tutorial Notes]", *2001 Annual Reliability and Maintainability Symposium,* IEEE, 2001, pp. 1-11.

[60] Schneeweiss, W.G., "Petri nets for reliability modeling: in the fields of engineering safety and dependability", LiLoLe Verlag, Hagen, 1999.

[61] Bobbio, A., "System modelling with Petri nets", edited by A.G. Colombo and A. Saiz de Bustamante, *System Reliability Assessment*, Kluwer, Dordrecht, Netherlands, 1990, pp. 102-143.

[62] Leveson, N.G., and Stolzy, J.L., "Safety Analysis Using Petri Nets", *IEEE Transactions on Software Engineering,* Vol. 13, No. 3, 1987, pp. 386-397.

[63] Ramchandani, C., "Analysis of asynchronous concurrent systems by timed Petri nets", PhD Thesis, 1974.

[64] Merlin, P.M., "A study of the recoverability of computing systems", PhD Thesis, 1974.

[65] Berthomieu, B., and Menasche, M., "An enumerative approach for analyzing time Petri nets", *Proceedings of the 9th IFIP Congress,* North-Holland, Amsterdam, Netherlands, 1983, pp. 41-46.

[66] Sifakis, J., "Use of Petri nets for performance evaluation", *Proceedings of the Third International Symposium on Measuring, Modelling and Evaluating Computer Systems,* North-Holland, Amsterdam, Netherlands, 1977, pp. 75-93.

[67] Coolahan, J.E., and Roussopoulos, N., "Timing requirements for time-driven systems using augmented Petri nets", *IEEE Transactions on Software Engineering,* Vol. 9, No. 5, 1983, pp. 603-619.

[68] Razouk, R.R., "The derivation of performance expressions for communication protocols from timed Petri net models", *Proceedings of the ACM SIGCOMM symposium on Communications architectures and protocols: tutorials & symposium,* Vol. 211, 1984, pp. 210-217.

[69] Beyaert, B., Florin, G., Lonc, P., "Evaluation of computer systems dependability using stochastic Petri nets", *Digest 11th Annual Symposium on Fault-Tolerant Computing,* IEEE Computer Society, 1981, pp. 79-81.

[70] Molloy, M., "Performance analysis using stochastic Petri nets", *IEEE Transactions on Computers,* Vol. 31, No. 9, 1982, pp. 913-917.

[71] Ajmone-Marsan, M., Conte, G., and Balbo, G., "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems", *ACM Transactions on Computer Systems,* Vol. 2, No. 2, 1984, pp. 93-122.

[72] Sörensen, K., and Janssens, G.K., "Automatic Petri Net Simulation Model Generation for a Continuous Flow Transfer Line with Unreliable Machines", *Quality and Reliability Engineering International,* Vol. 20, No. 4, 2004, pp. 343-362.

[73] Volovoi, V., "Modeling of system reliability Petri nets with aging tokens", *Reliability Engineering and System Safety,* Vol. 84, No. 2, 2004, pp. 149-161.

[74] Malhotra, M., and Trivedi, K.S., "Dependability modeling using Petri-nets", *IEEE Transactions on Reliability,* Vol. 44, No. 3, 1995, pp. 428-440.

[75] Volovoi, V., "Modeling multiphased missions using stochastic Petri nets with aging tokens", *Proceedings of the Annual Reliability and Maintainability Symposium 2004,* IEEE, Piscataway, NJ, USA, 2004, pp. 232-236.

[76] Mura, I., and Bondavalli, A., "Markov Regenerative Stochastic Petri Nets to Model and Evaluate Phased Mission Systems Dependability", *IEEE Transactions on Computers.,* Vol. 50, No. 12, 2001, pp. 1337-1351.

[77] Jensen, K., "Coloured Petri nets: A high level language for system design and analysis", *Lecture Notes in Computer Science: Advances in Petri Nets 1990,* Vol. 483, 1990, pp. 342-416.

[78] Kumar, V., and Aggarwal, K.K., "Petri net modelling and reliability evaluation of distributed processing systems", *Reliability Engineering and System Safety,* Vol. 41, No. 2, 1993, pp. 167-176.

[79] Muppala, J., "Performance and dependability modeling using stochastic reward nets", PhD Thesis, 1991.

[80] van der Aalst, W.M.P., "The Application of Petri Nets to Workflow Management", *Journal of Circuits Systems and Computers,* Vol. 8, No. 1, 1998, pp. 21-66.

[81] Corradini, A., "Concurrent Computing: from Petri Nets to Graph Grammars", *Electronic Notes in Theoretical Computer Science,* Vol. 2, 1995, pp. 56-70.

[82] Lopez-Benitez, N., "Petri-Net Based Performance-Evaluation of Distributed Homogeneous Task Systems", *IEEE Transactions on Reliability,* Vol. 49, No. 2, 2000, pp. 188-198.

[83] Horton, G., "A new paradigm for the numerical simulation of stochastic Petri nets with general firing times", *Simulation in Industry: 14th European Simulation Symposium,* SCS-Europe Publishing House, Erlangen, Germany, 2002, pp. 129-136.

[84] Liu, T.S., and Chiou, S.B., "The application of Petri nets to failure analysis", *Reliability Engineering and System Safety,* Vol. 57, No. 2, 1997, pp. 129-142.

[85] Bondavalli, A., Mura, I., and Nelli, M., "Analytical modelling and evaluation of phased-mission systems for space applications", *Proceedings of the IEEE High-Assurance Systems Engineering Workshop 1997,* IEEE Comput. Soc, Los Alamitos, CA, USA, 1997, pp. 85-91.

[86] National Transportation Safety Board, "In-flight Breakup Over The Atlantic Ocean, Trans World, Airlines Flight 800, Boeing 747-131, N93119, Near East Moriches, New York, July 17, 1996", Aircraft Accident Report NTSB/AAR-00/03, Washington, DC. http://www.ntsb.gov/Publictn/2000/AAR0003.pdf

Appendix A Description of Simulation Program

A.1 Introduction

The creation of a complex PN model is one which would require a great deal of time to recreate even one scenario of component failures. One of the benefits of performing these tasks on a computer simulation program, as outlined in Section 1.5, is that thousands or millions of simulations can be performed in a short timeframe. A simulation program using Monte-Carlo methods can, with sufficient numbers of simulations, build up an accurate picture of the reliability performance of the platform.

These facts led to the creation of a program which applies the Petri net modelling methods explained in Chapter 5 and Chapter 6 in the form of Monte-Carlo simulations. This is the means of providing the data to accurately analyse all aspects of the combination of phased missions and MFOPs being performed by a series of platforms.

The following sections of this appendix will explain the procedures of the program, including:

- Inputting phase fault trees, component data, mission, MFOP and platform data, and information on MFOP and phased mission complexities
- Conversion of inputted information into a useful set of PNs
- Changing options regarding running the simulations
- Performing simulations on the platform
- The way in which the inputted data is stored, and what parameters from the simulations are stored
- Outputting the results obtained or information on the system

In this appendix, code is always displayed in `a different font`, and is written as it would typically be in C++, except where abbreviated code is indicated in square

brackets. Comments are shown either on the same line as code, using two forward slashes `//`, or as a block of text, which is bookended by a forward slash and an asterix:

```
/* A block of text following /* will be ignored by the compiler
   until ended with the characters: */
```

Similarly, text which forms the inputs or outputs to the program is also written in a different font.

A.2 Basic Information

The simulation model was programmed in C++ using Microsoft Visual Studio 6.0. This is an object-oriented language, which differs from structured languages. Most programming languages provide the capability for creating variables of integer, floating point, Boolean and string types, amongst others. In a structured language, it is possible to define a *structure* type which contains many different sub-variables. For instance, consider the following code:

```
struct simple_structure
{
    int a;
    float b;
    bool c;
    char d;
};
```

A structure has been defined of type `simple_structure` which has the sub-variables `a`, `b`, `c` and `d`. Values can be assigned to these sub-variables and functions can be written to use them. In an object-oriented programming language, this ability to bind together sub-variables under an umbrella structure is still present. Also defined under that umbrella, however, are the functions capable of being performed on the sub-variables. Instead of a structure, this is known as an *object* of a *class*. This can be defined in C++ as follows:

```
class simple_object
{
public:
    void function1();
    int function2();
    float function3();
private:
    int a;
    float b;
    bool c;
    char d;
};
```

The `public` and `private` operators are there to define what aspects of the class non-object functions can access. Best practice states that object functions should be accessible, or "public", to any part of the program, while the variables should be "private", or only directly accessible to the object. Typically, the only way to access a variable in an object is if a function exists to return the variable or its memory pointer. Continuing with the above example, the object would be defined in the following way:

```
void external_function()
{
    simple_object object_name;    //Defining the object
    object_name.function1();      //Accessing a function of the
    object
}
```

Coding in an object-oriented language allows for a great deal of flexibility and memory efficiency. Every effort has been made with the code to establish a good balance between time to run all of the simulations and the memory used in doing so. Dynamic memory allocation (DMA), where variables, arrays, and objects are only defined at run-time to be the size that they need to be, has been used copiously. Using DMA allows for the possibility of *memory leaks*, where memory is allocated but is not released after it is no longer needed. All memory leaks have been removed from the program, so that its RAM usage remains stable while running a batch of simulations.

The interface of the program is not a standard Windows application. Like most simple programs coded in C++, the graphical user interface (GUI) is, instead, a screen resembling DOS, an early text-based operating system, as shown in Figure A.1.

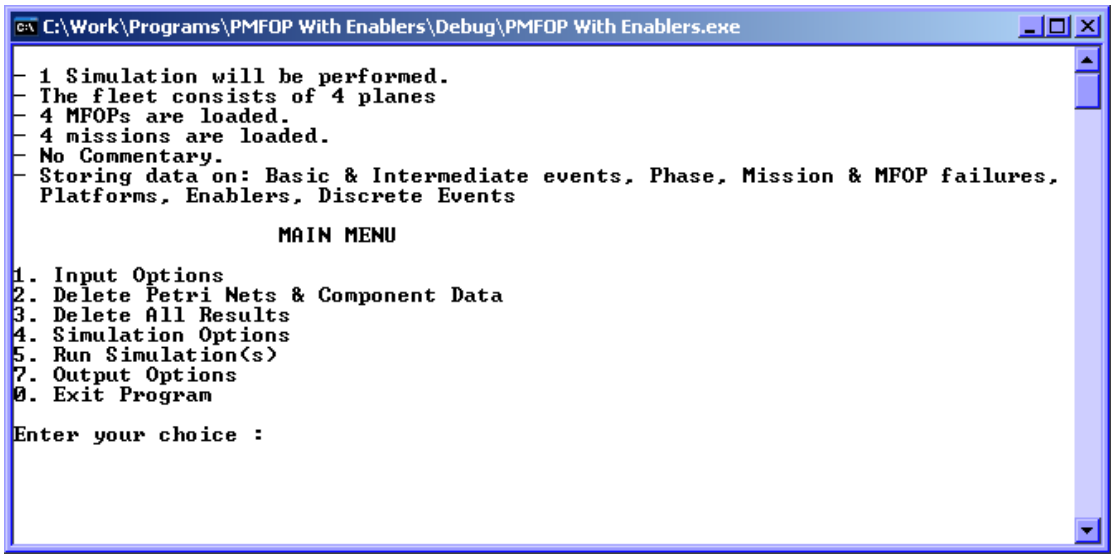


Figure A.1 – Typical GUI of Simulation Program

The status of the inputted model is shown at the top of the program screen, with information such as the number of simulations to be performed and fleet, MFOP and mission data. The user options are presented in the form of menus, each one of which has an assigned number. The number is typed in where the screen says "Enter your choice", and pressing enter takes the user to that option. Some options result in sub-menus, while others perform tasks directly.

Upon execution, the program first creates a main simulation object named "`Sim`", of class "`Input_Output`". This single object controls the entire program and operates everything else. The whole of the rest of the code, including all other objects, is a part of `Sim`. These other objects are:

- A Petri Net object, containing all the information on the various PNs
- Objects for Platforms, MFOPs, Missions and Phases
- Objects for LRIs, Systems, Components and Discrete Events
- Objects for redundancies and prognostics.

`Sim` also contains variables such as the numbers of platforms, MFOPs, missions, phases, LRIs, systems, components and discrete events, in addition to the confidence level mentioned in Section 6.3.8.3, and the number of simulations to be performed. The PN object is created at run-time, although it is initially empty.

A.3 Inputting Data to the Program

The first step necessary to operating the program is to input data to allow it to create the various Petri Nets used to model the fleet of platforms and their missions. All inputs are in the form of text files, which are easily inputted to the program, but must be in a very specific format for each type of input. As Figure A.2 shows, entering the input menu, the user is faced with two options: return to the previous menu, or input the phase failure & abandonment fault trees, and sensor and component failure data.

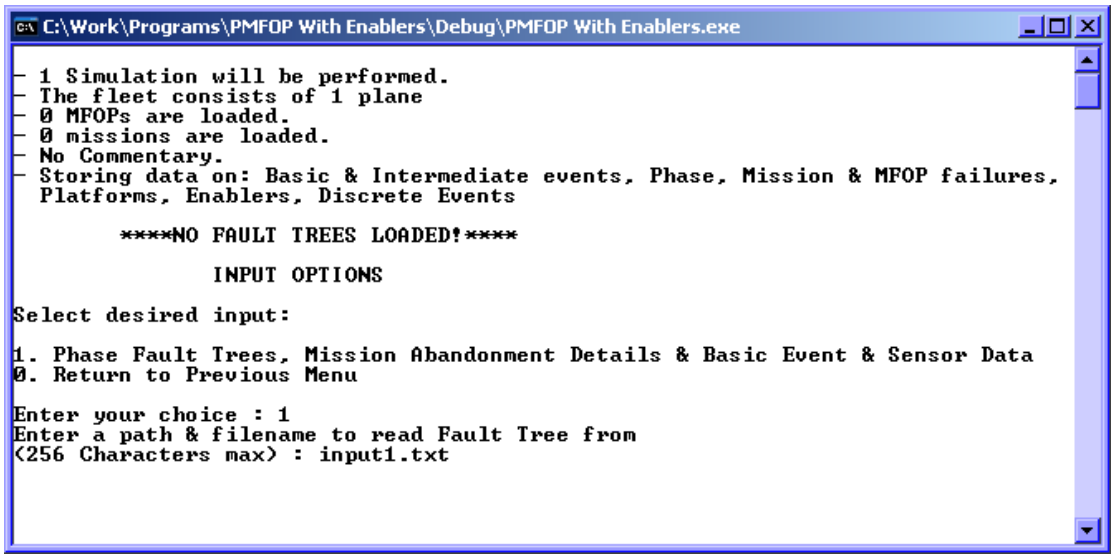


Figure A.2 – Input Menu

A.3.1 Phase Fault Tree, Component and Sensor Data

Figure A.3 shows the process by which the Phase fault trees, component failure rates, sensor data, discrete events and component dependencies are inputted to the program. The following sections describe this in more detail.

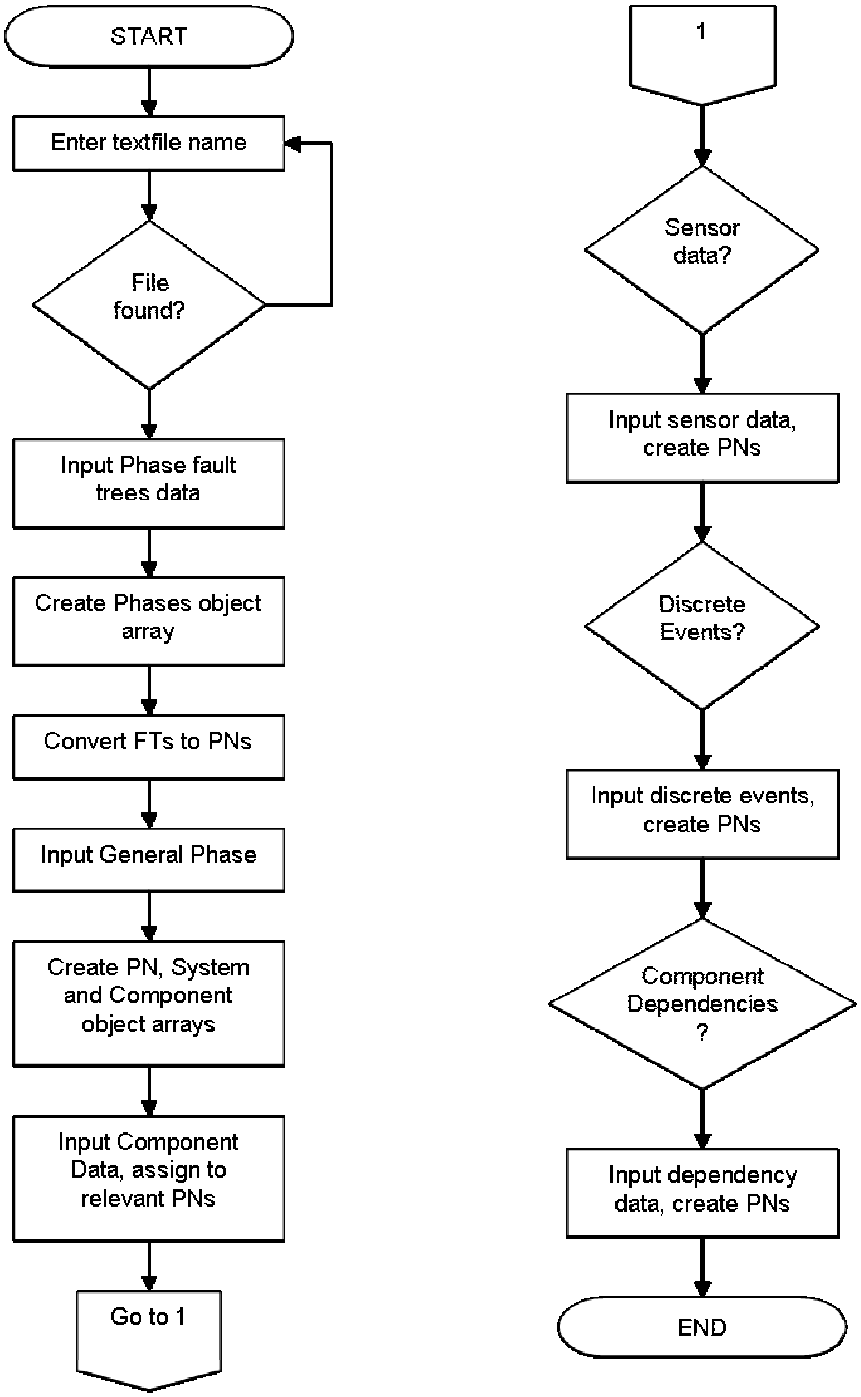


Figure A.3 – Flowchart describing input processes for phase fault trees, component data, sensors, discrete events and component dependencies

Entering the name of a text file which contains this information begins the process of creating the first set of PNs. This file must contain the following information, in the following order:

- Number of phases to be inputted.
- Catastrophic Failure Fault tree information for each phase
- If an abandonment specific to the phase is possible, the details of the fault tree for that.
- Fault tree information for general total loss and general abandonment
- Failure information for each basic event specified in the phase fault trees
- Information on sensors, including what they monitor and failure rates
- Data on discrete events
- Specifics of dependencies between components

A.3.1.1 Phase Fault Trees

The layout of the text file is very precise. For inputting the phases, this is as follows:

```
PHASES              <# PHASES>

PHASE               <PHASE NAME>   <# EVENTS>

<EVENT 1 NAME>      <GATE TYPE>    <# EVENTS >  <EVENT 1>   <EVENT 2>   ...
<EVENT 2 NAME>      <GATE TYPE>    <# EVENTS >  <EVENT 1>   <EVENT 2>   ...
...

ABANDONS            <# EVENTS>

<EVENT 1 NAME>      <GATE TYPE>    <# EVENTS >  <EVENT 1>   <EVENT 2>   ...
...

PHASE               <PHASE NAME>   <# EVENTS>
...
```

In practice this is typed as:

```
PHASES 6

PHASE A   5

GTOPA     + 5  G1001     G1002   A   B   C
G1001     * 2  G1003     D
...

ABANDONS  1

ATOPA     * 3  A1001     C       E
A1001     + 5  A         B       D   F   G
...
```

The number of phases ("`no_phases`") inputted at the top of the file allows for the creation of an array of objects relating to phases. The number in the array is `no_phases+1`, which accounts for the general phase in addition to the others. While each of these objects stores a great deal of information, the only data that can be saved immediately is the name of each phase and the number of the system which represents the abandonment fault tree top event.

For each phase, the fault trees for total loss and abandonment are written in text format. Each line represents one event, top or intermediate, of the fault tree, with the name of that event written first. Each of these has a logic gate attached to it, into which various other events input. If the logic gate is an AND gate, this is written as '`*`', while if it is an OR gate, this is typed as '`+`'. Following this is the number of event inputting to the logic gate and the names of each of these. The data are inputted and the events are separated into their different types. All top events are the names of the first lines in each fault tree. Intermediate events are the names of the other lines in each fault tree, while basic events are those which have no logic gate inputs.

The program assigns a number to each event, based on their type and the value of a global non-variable identity called `NO_BASICS`. This number, set to a preliminary value of 1000, defines the maximum number of basic events which may be inputted.

Each basic event number lies in the range 0 to NO_BASICS-1, while each top or intermediate event number lies in the range NO_BASICS to 5×NO_BASICS-1. These numbers are attached not just to events but to places which represent those events, and are used for identification.

Once the phase fault trees have been inputted, the general failure and abandonment fault trees are taken in. These follow exactly the same format as above, swapping the words "PHASE <PHASE NAME>" for "GENERAL FAILURES". The number of basic events throughout all the different fault trees is added up, taking account of any repetitions, and this forms the number of components ("no_compts"). As mentioned in section 6.2.3, the other events from the fault trees are considered to be systems, and so that number is summed (again accounting for repetition of events), and this becomes "no_systems". These values are used to create two objects arrays.

The only information that can be stored on each component or system, at this point, is its name and number. In addition, the program scans through the fault trees, and sets each component object to store the numbers of the systems it occurs in. Each system object stores data on each system and component inputting to it, as well as the number of times that system is repeated, and the phases in which it occurs.

Following the logging of data for each phase, system and component, the program begins converting each event in each fault tree into PN form. The PN object mentioned above contains two object arrays – one for places, and one for transitions, in addition to other variables. Each place object is quite small, containing data only on the name of the place, its number, the marks it contains for each platform and the numbers of the transitions to which it connects with three different types of arc. A transition object also contains this arc data on places linked to it, as well as whether it is timed or immediate, probabilistic or not, its time to switch, any probability arcs, the probability distribution from which the time to switch is sampled and whether it is enabled or not. These objects are set up in a two-dimensional array. The first dimension is the number of phases plus three, which relates to each PPN, the General PPN, the Master PN and the Component PN. The second dimension is the number of places or transitions within each of these.

The process of fault tree conversion to PN form begins by creating in the component PN the three places, shown in Figure 6.10, which represent "MFOP Active", "MRP Active" and "Next MFOP". Two loops are then entered, running through each phase in turn and, within that phase, each event. As according to the phase PN information shown in Section 5.2.1.1, a place is created for each fault tree event, with the logic gates set up according to the inputted information. Sub-event places are also created, and if these are basic events, the "up" and "down" places, as well as the failure and repair transitions are added and connected to each other. As each place or transition is set up, the relevant object (phase, system, component) is made aware of that node's number, expressed as a PN and place, in the array. Most of these are stored with an accompanying "type" number. This number is arbitrary, as long as it is different from other place and transition "types", and can be recalled when the program needs the node index in another function. For instance, the "up" and "down" places for a component are types 0 and 1 respectively.

A.3.1.2 Component Failure Data

Once each of the fault trees has been set up in PN form, and all the data extracted and organised in arrays of objects, the next stage is to input the component data. The information must be arranged as shown below:

```
COMPONENTS

<COMPONENT 1 NAME>  <R/U>  <ON DIST>   <PARAS>  <OFF DIST>   <PARAS>
<COMPONENT 2 NAME>  <R/U>  <ON DIST>   <PARAS>  <OFF DIST>   <PARAS>
...
```

where `R/U` is whether the component's failure is revealed or unrevealed.

`ON DIST` is the component's online failure distribution type, followed by its parameters

`OFF DIST` is, if it exists, the component's offline failure distribution type, followed by the relevant parameters. This is used with components which are set offline by a redundant system and have warm or hot offline failures.

The distribution types are of six different types, each entered in a specific way:

- `EXP` is the exponential distribution. It requires one parameter, which is the failure rate per hour.
- `SET` is a deterministic time, followed by a time in hours. The component will fail at exactly this time. This is used mainly for experimentation and troubleshooting, and is not a realistic description of a component's actual failure rate.
- `NORM` is the normal distribution. It has two parameters, the average failure time and the standard deviation.
- `LNORM` is the lognormal distribution, with the same two parameters for the logarithm of the function.
- `WEIB2` and `WEIB3` are the two-parameter and three-parameter versions of the Weibull distribution. The first two parameters, common to both, are the shape and scale parameters respectively, while the third one, used only with `WEIB3`, is the location parameter, which shifts the distribution such that no failure can occur before this point.

Examples of how these component distributions are laid out are shown below.

```
COMPONENTS

A             EXP     5.9E-5        EXP  7.2E-8
B             SET     50
C             NORM    600      70        NORM    1500  40
D             LNORM   6.9      0.1
E             WEIB2   2.4      700
F             WEIB3   2.4      700  300
...
```

The program has calculated how many components to expect during the previous stage, so this information is not necessary to input. After the first set of probability distributions is inputted, the program checks whether the next input is the name of any of the distribution types or not. If it is, then it knows to input the rest of the data for the offline failure transition and create this for the component. Otherwise, there

are no PNs to set up; the information is added to the relevant timed transition objects.

A.3.1.3 Sensor Data

Once component data has been saved, the next set of information to be taken in is that for sensors. Sensors are themselves components, and so failure data needs to be specified for them, in addition to the items that the sensors monitor. The standard layout for this information in the inputting text file is as follows:

```
SENSORS

<SENS 1 NAME>  <SENSED ITEM>  <R/U>  <FAIL DIST>  <PARAS>  <NFF DIST>  <PARAS>
<SENS 2 NAME>  <SENSED ITEM>  <R/U>  <FAIL DIST>  <PARAS>  <NFF DIST>  <PARAS>
...
```

An example of this would be:

```
SENSORS

SENS_A         A            R     EXP        7E-8    EXP     6E-3
SENS_G1001     G1001        U     EXP        4E-11   EXP     8E-4
...
```

The number of sensors does not need to be inputted; the program will input each line at a time, create the PNs for the component as needed, and loop through this procedure until it is recognised that the section for sensors has ended. The sensor could have been included as a component in the phase fault trees entered earlier. If it was, the failure data is applied to the component's online transition as above. If it is a new component, then the PN is set up for it with the inputted failure data. Following this, the failure place for the item which the sensor monitors is found, and two new transitions and a place are created to allow the detection to take place, as shown in Figure 6.23. Every place for the sensor is set to have a number lying in the range of `5×NO_BASICS` to `6×NO_BASICS-1`.

The No-fault-found (NFF) rate describes the frequency with which sensors "find" a failure which does not exist. One of the major problems regarding availability of

platforms, NFF may force evasive action or bring online redundant systems when it is not necessary too. The probability distribution which describes the likelihood of an NFF occurring is inputted after the main failure distribution. If a sensor is perfect and does not give spurious trips, the NFF rate does not need to be entered.

A catastrophic failure fault tree describes events which must happen in order for the platform to undergo total loss. It is a real-world, definite result of a series of component and system failures. An abandonment fault tree, however, is a description of the events which must happen before the current mission is aborted, and this is based on the information which is available at the time. As such, if the sensor is mentioned in a catastrophic failure fault tree, it is assumed that the sensor's failure is critical to the phase. If, however, the sensor inputs to an abandonment fault tree, it is assumed that the data the sensor provides is most important. As such, the arcs from the sensor's failure place are all transferred instead to the "failure sensed" place just set up.

Sensors are components, and so their data on names, numbers, failure data, and so on, is stored as part of the components object array.

A.3.1.4 Discrete Events

The next stage in the inputting of the first text file is the discrete events. As described in Section 6.3.2, these can be set up to be triggered by almost any event describable by the model. However, many of these events have not yet been set up. As a result, it is possible to input at this stage only discrete events which are triggered by component, sensor or system failures. Because it is called at two different stages, the function for inputting discrete events is external to, and called by, the function which inputs this text file. The discrete event data is inputted using the following format:

```
DISCRETE EVENTS   <# EVENTS>

<DE 1 NAME>       <TRIGGER NAME>   <SUCCESS PROBABILITY>
<DE 2 NAME>       <TRIGGER NAME>   <SUCCESS PROBABILITY>
...
```

An example of this might be:

```
DISCRETE EVENTS   4

DE1               A       0.76
DE2               G2001   0.5
...
```

The input function loops for the number of DEs to be entered. Taking each line at a time, the trigger event name is searched for among the sensors, components and systems. When the correct item is found, the failure place is logged. After this, a search is performed to establish whether the DE has already been inputted as a component input to a fault tree. If it has, the component net is converted into a discrete event PN; if not, it is created as new in the component PN. The conversion process makes the "Component Failure" place into the "Yes" place and the "Component Operational" place into the "No" place. The timed failure transition becomes a probabilistic, immediate decision transition, while the repair transition is the transition which clears the Yes place when the trigger place is emptied. A transition is created to empty the No place. The arcs are altered to be in the right place, leaving the next stage to be performed, which links the trigger place to the DE net, enabling the decision transition and inhibiting the yes and no place clearance transitions.

Objects are created for each discrete event, and store information on its name, number (the number for each DE is set to be in the range `6×NO_BASICS` to `7×NO_BASICS-1`), the associated probability of the event succeeding and each of the places and transitions.

A.3.1.5 Component Dependencies

A component dependency, explained further in Section 5.2.1.2.1, describes the way a component's failure rate may change when another component fails. The nature of this change is capable of being expressed, to the program, in two ways. The first of these is by specifying another distribution for the component to be used when the other component fails. The second is by inputting a "stress level". The "stress" on a component can be thought to be a combination of all of the factors which can cause

an item to fail faster or slower: temperature, humidity, vibration, and so on. If the stress on a component doubles when another component fails, the failure time decreases by half.

The method of inputting data on component dependencies is shown in an example below:

```
DEPENDENCIES

A             2   D            EXP        1.2E-3
                  E            NORM       1.1E-3   100
                               EXP        0.018
C             1   E            1.3
```

The number of components with dependencies does not need to be given. As the component dependency details are the last set of data to be inputted, the input function will continue input dependency data until the end of the file is reached.

A component name is taken in, followed by the number of components on which it is dependent. If the number is one, then the program inputs one component name followed by the stress or failure data for one transition. If, however, it is dependent on the upper limit of two components, then there are three extra transitions needed. The relevant number of transitions is created and linked to the "operational" and "failure" places for each component dependency, as shown in Figure 5.5. If a stress value is entered, as a proportion of the normal value, the new failure transition is given the same distribution information as the original failure transition. The component object which has the multiple failure transitions is then set to store this information.

A.3.2 MFOP Enabler Data

Once the first set of information has been installed, the program allows the user to specify a second text file which contains information on the MFOP Enablers, as shown in Figure A.4.

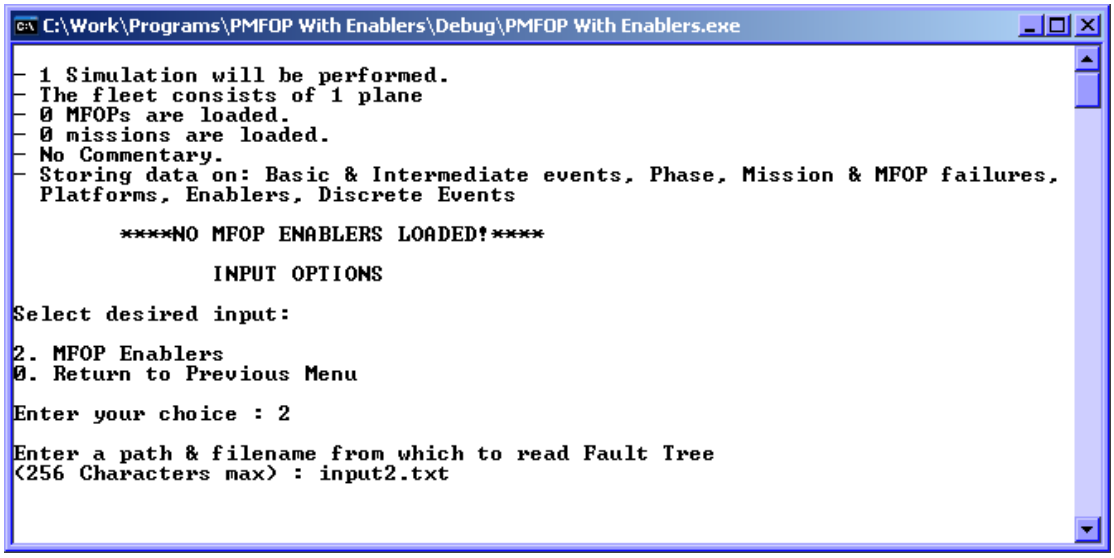


Figure A.4 – Inputting MFOP Enablers Text File

Pressing '2' and then entering the full name of the file begins the process. The program allows five types of item to be specified at this point, in the following order:

1. Line Replaceable Items (LRIs)
2. Scheduled Life Replacements (SLRs)
3. Inspections
4. Redundancies
5. Prognostics

The data is inputted as described in the flow chart in Figure A.5.

A.3.2.1 LRI Data

LRI data must be set out in the text file as shown below:

```
LRI            <# LRIs>

<LRI 1 NAME>   <VOTE NO> /   <# ITEMS>   <ITEM 1 >   <ITEM 2>   ...
<LRI 2 NAME>   <VOTE NO> /   <# ITEMS>   <ITEM 1 >   <ITEM 2>   ...
...
```

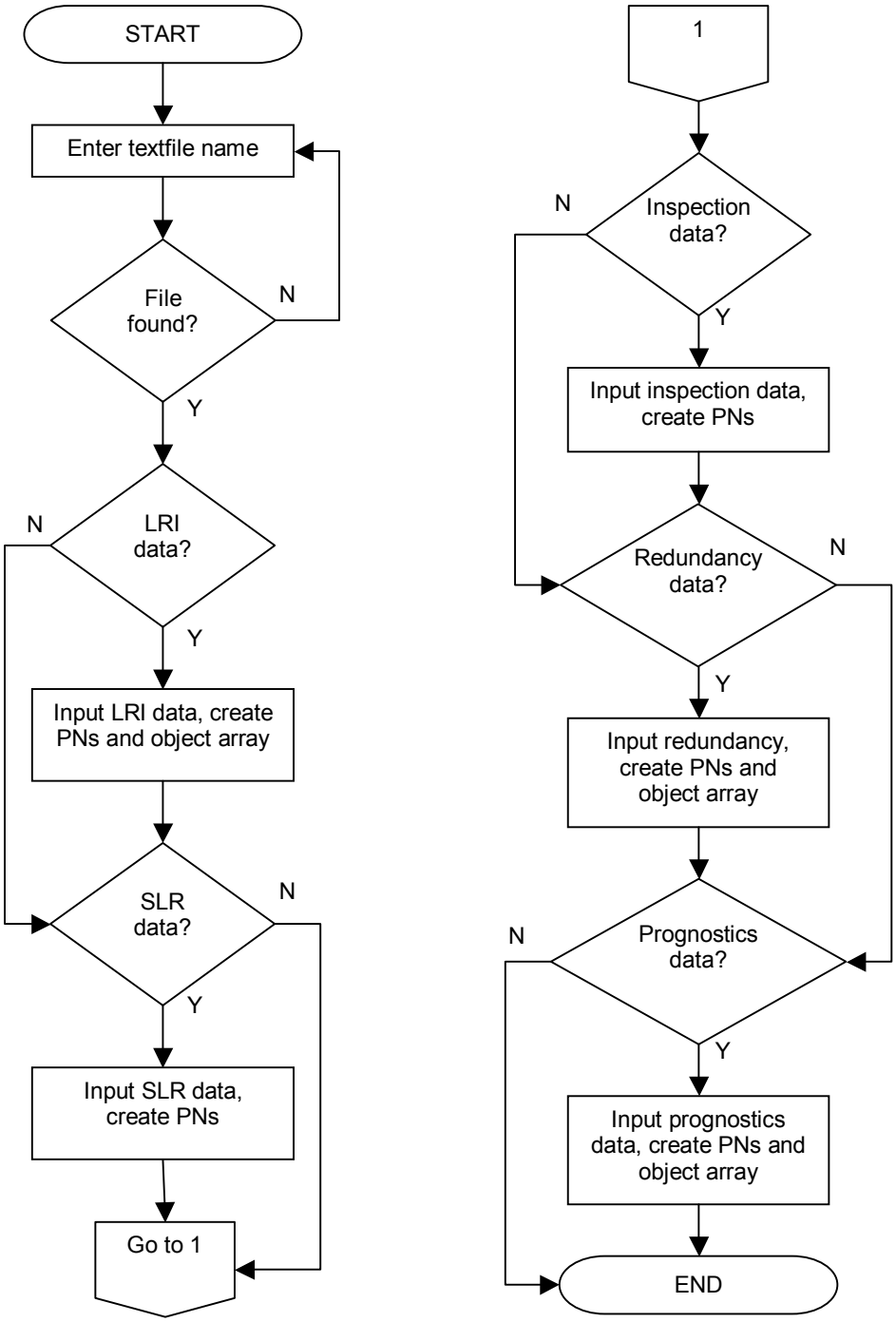


Figure A.5 – Flowchart describing input process for MFOP Enablers Text File

An example of this would be:

```
LRI        2

LRI_1      2/3   LRI_2   B   G3001
LRI_2      1/4   A       G   D       A4010
```

The number of LRIs must be stated on the first line, after the word "`LRI`". Following this, rather than taking each line in turn and creating each LRI one at a time, certain sections of all the LRIs are generated in sequence. This is due to the interactions that may exist between different items; the program must be aware of all of the LRIs it is making.

An array of objects for the LRIs is set up, and the vote number, number of events and the event names are inputted and stored locally. Each LRI object is given a name and the vote number, after which each of the items which are contained in the LRI are categorised according to whether they are components, systems or child LRIs. Each LRI object stores the array index numbers of its child items, and, if it exists, its parent LRI. The total number of sub-items is stored also at this point.

Another loop is entered, one pass for each LRI, and the number of places and transitions to be created is established. This is done so that the arrays for the place and transition objects, contained in the `Petri_Net` object as mentioned above, can be extended to contain the new nets.

The next stage is creating the PNs to replace components, systems and child LRIs. Running through each LRI one at a time, component and system failure places are located, as are LRI main places. A first input transition is created and the index number is stored in the LRI object – other input transitions do not need these numbers stored, as their index numbers increment by one from this value for each child item. If the vote number equals the number of events, then the logic input to the LRI is created as an AND gate, and this one input transition will perform the task. If the vote number equals one, the logic is an OR gate, and a new input transition is needed for each input. The transitions are linked to the input places and the LRI main place, as shown in Figure 6.18. The figure shows that systems have control places created for the inputs, and so these are created after all of the other input items have been managed. If the vote number is greater than one and less than the number of events, then the logical input to the LRI is as a voting gate. The input transition already created is linked to by the vote place, as shown in Figure 6.19. The child item places are linked to vote transitions and control places as shown in

the figure. Each LRI object is set to store the index numbers for each place and transition used in inputting to the main place.

The subsequent step is to set each LRI with child LRIs to place a token in the child's main place, using an immediate transition. After this, all component inputs, either directly or indirectly through being a part of a child system, are set to be replaced whenever the LRI place is marked, as shown in Figure 6.20. The new place and transition index numbers are stored by the matching component object. The final step is to create the transitions which remove the tokens from the LRIs when all sub-components have been replaced.

A.3.2.2 Scheduled Life Replacements

If the operator specifies individual LRIs or components to undergo regular replacements, regardless of its working or failed state, that information is entered here. Its simple format is exemplified below:

```
SCHEDULED LIFE   2

A                150
LRI_2            300
```

The number of SLRs to be created is given after the words "`SCHEDULED LIFE`". The first line is taken in, and the first term, which should be the name of the component or LRI, is compared to the object arrays and found. The PNs are created as shown in Figure 6.21, with the relevant component or LRI object made to remember the places and transitions. If a component is also a member of an LRI, an account is made of this as shown in Figure 6.22. The timed transitions which, when they switch, mark the item as requiring replacement are given the "`SET`" distribution, with the parameter being the second term entered from each line.

A.3.2.3 Inspections

When inputting data on inspections to be carried out, the text file must have its information set out in the following way:

```
INSPECTIONS    <# INSP>

<ITEM 1 NAME>  T/M/S      <PARAMETER>
<ITEM 2 NAME>  T/M/S      <PARAMETER>
...
```

The number of inspections to be inputted must be stated after the title "INSPECTIONS". "T/M/S" specifies the type of inspection that the item is to have: 'T' refers to an inspection carried out after a given time, specified where the line says "<PARAMETER>". 'M' is an inspection made after a number of MFOPs, while 'S' allows a system to be specified which, if found to be failed, prompts the inspection of this item. An example of how the inspection data is inputted is shown below.

```
INSPECTIONS    6

G2001          T   300
B              M   4
C              S   G2001
...
```

Once the information is all inputted, the program checks to see whether the item is a system or component, and if it is a member of an LRI or not. The places and transitions to be created are counted and the object arrays extended again. Initially, inspection places are created and linked to the relevant component or system item, accounting for LRI membership. The PNs for this are shown in Section 6.3.6.2. If the inspection place becomes marked through a count of time or MFOPs, the transitions to allow this are created next. If, however, inspections are based upon the discovery of failure in a higher system, then the program must complete the process of creating nets for inspection of the higher system if this is specified. Only after this loop has ended can another loop be entered, linking system inspection, "failure sensed" or failure places with a transition to input to the lower item. As with all other inputs, the LRI, system and component objects store the index numbers of places created for them, where this is necessary.

A.3.2.4 Redundancies

Redundancies and prognostics, which are covered in the next section, are handled slightly differently to the other MFOP enablers inputted. Each of these has its own object containing information about the enabler. Component (which can have both redundancies and prognostics) and system (which can only have the former) objects have a pointer to each of these built-in and set to zero at run-time. Only when the item is defined as having redundancies or prognostics is the object for the enabler created and the pointer set to the newly-allocated memory point.

The process of inputting the details of redundant items begins with the information given in the text file. This is laid out, typically, as shown below.

```
REDUNDANCIES <# REDS>

<ITEM NAME>  S/M      <PARAMETER> <# OUTPUTS> <OUTPUT 1> ...
```

An example of this is shown below:

```
REDUNDANCIES 4

A     S SENS_A     2 B C
C     M 0.89       1 D
G2002 S SENS_G2002 1 G2003
G3004 M 0.56       3 C F H
```

Redundancies, as explained in Section 6.3.7, are either automatically switched upon failure detection by a sensor, or they are manually switched by an operator. The former scenario is selected by using an 'S' and specifying a sensor component to link to. The manual switching requires a parameter to specify the probability that the redundancy will be successfully brought online. After this is given, the number and name of the items which are brought online when the main system fails are inputted.

After the data is inputted, the program enters a loop which is bounded by the number of items with redundancies. Almost the entirety of each redundancy PN is created in this loop, one at a time. The failure places for the item is located, and the

net allowing it to be turned off upon failure is set up, depending upon whether the item is a system or component, and the automatic or manual nature of the switching. This includes linking the “X Op” place to component online and offline transitions as shown in Figure 6.34. The redundancy object in the item is made to store the index number for each place or transition created.

Another loop is then entered, within the previous one, which is terminated by the number of redundant item outputs the main item has. Each output item has its redundancy nets created and linked to the “Yes” or “No” places, which represent the successful or unsuccessful switching to the redundant item(s), as shown in Figure 6.35 and Figure 6.36. The “Y Op” place, like the “X Op” place earlier, is linked to the component(s) which are turned on when the redundancy is brought online. These details of these nodes are stored in the item’s redundancy object.

The final part of the function creates the PNs which link the “X Op” place for a subsystem with redundancies to that of a higher system with redundancies. As shown in Figure 6.41, the X Op place for the child item must be unmarked when the parent system fails. Because this is linking different redundant items, this must be performed outside of the previous two loops. The function ends after this, and prognostics information may be entered.

A.3.2.5 Prognostics

As explained in section 6.3.8, the prognostics modelling method devised within this research is based on the idea of detectable deterioration in components. This deterioration, or wear, must be expressed in the form of levels, with confident predictions based on the typical point in an item’s life when the wear level occurs. Thus, in order to set up nets for prognostics, it is necessary for the wear levels to be specified in the input text file:

```
PROGNOSTICS 2

C 3 15 25 40 50 70 92
I 2 10 20 80 90
```

This information is set up with the number of prognostic components stated immediately after the title, and the names of each item specified first on each line. Following the component name, the number of wear levels is given, followed by the lower and upper bounds for each level in turn. Thus, in the example shown, component 'C' has three wear levels, with the bounds 15%-25%, 40%-50% and 70%-92% of the component's life. 'I' has two wear levels, the first between 10% and 20%, and the second between 80% and 90% of the component's life.

This is all the information that the program needs to set up the PNs. By this point, all diagnostic methods (sensors and inspections) should have been declared and inputted, and the program uses this information, if it exists, in generating the PNs for prognostics. The input function enters a loop terminated by the number of prognostics, and creates the set of nodes and arcs entirely within that loop.

As mentioned earlier, the components which feature prognostics have an object created to store such data as the wear levels, the current predicted failure time, PN information, and so on. The correct component is found in the component object array by searching for the inputted name, and, if it does not already exist, the net for component replacement is set up. If it does already exist, then the component is probably part of a LRI, and so the LRI number is located. The places and transitions for "Wear Level (Actual)" and "Action" are set up, as well as the transitions which remove the tokens from these when the replacement place is marked. If the component has sensed failures, the PNs are set up to use the sensor to detect the wear. If, instead, regular inspections are performed, the inspection place is connected as shown in Figure 6.43, with the "Wear Level (Detected)" place set up as well.

If the component was earlier found to be a part of an LRI, then when the prognostics model requires the component to be replaced, a transition is set up to force the LRI to replaced, regardless of the state of its other inputs. Otherwise, the action place is linked to the "Replace component" transition and the function exits. This ends the second input stage, following which the missions data can be entered.

A.3.3 Platform, MFOP and Mission Data

Once the MFOP enablers information has been entered, the final stage of the input process is telling the program the details of the fleet, the MFOPs they will perform, the missions within those MFOPs and the phases in the missions. The program prompts for the input as shown in Figure A.6.

```
C:\Work\Programs\PMFOP With Enablers\Debug\PMFOP With Enablers.exe

- 1 Simulation will be performed.
- The fleet consists of 1 plane
- 0 MFOPs are loaded.
- 0 missions are loaded.
- No Commentary.
- Storing data on: Basic & Intermediate events, Phase, Mission & MFOP failures,
  Platforms, Enablers, Discrete Events

        ****NO MISSION DATA LOADED!****

                INPUT OPTIONS

Select desired input:

3. Phased Mission details
0. Return to Previous Menu

Enter your choice : 3
Enter a path & filename to read Fault Tree from
(256 Characters max) : input3.txt_
```

Figure A.6 – Inputting Platform, MFOP and Mission data

Entering '3' and then the name of the file containing the information begins the process. Figure A.7 shows a flow chart describing the process by which the data inputted is used to create the Master PN. All details, excluding those for discrete events, are taken in and stored before any PNs are created. How the data must be set out is explained in the following sections.

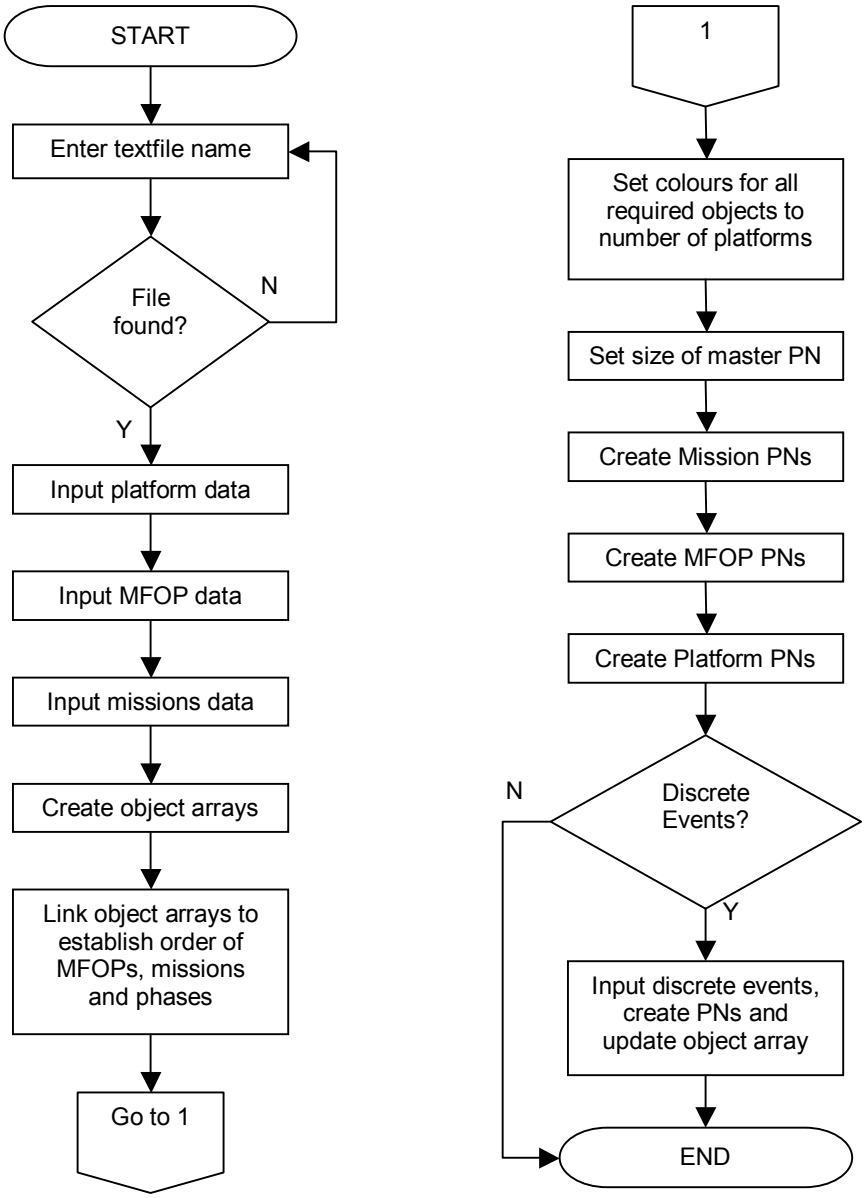


Figure A.7 – Flowchart describing input process for Platform, MFOP and Mission Data Text File

A.3.3.1 Fleet and MFOP information

Everything in the program must be pre-determined. Platforms perform a set sequence of MFOPs, following which the simulation is complete. This data takes the form exemplified below:

```
FLEET 4

PLTFM1 3 MFOP1 MFOP2 MFOP3
PLTFM2 4 MFOP2 MFOP2 MFOP3 MFOP3
PLTFM3 2 MFOP1 MFOP1
PLTFM4 2 MFOP1 MFOP4
```

The number of planes in the fleet is given next to that title. After this the name of each platform is given on a line, followed by the number of MFOPs it is to perform and the names of these. The MFOP data is laid out in a similar way:

```
MFOPS 4

MFOP1 3 ASUW ASW MISSION3
MFOP2 4 MISSION3 ASW ASUW ASUW
MFOP3 2 MISSION3 MISSION3
MFOP4 1 MISSION4
```

The number of MFOPs is entered next to the title "`MFOPS`". Each set of information for an MFOP is written on a line, with the name coming first, followed by the number of missions within it and the names of each of them.

The platforms and MFOPs each have their own objects in arrays. At this stage, however, the only information that can be entered are, for the fleet, the names of each platform and the number of MFOPs they perform. For the MFOPs objects, the data stored is their names and the number of missions within them. A function also tells each platform object of the sequence of MFOP indices which it performs.

A.3.3.2 Missions Information

Because of the phased mission complexities allowed by the model, as specified in Section 6.2, factors such as phase choices or insertions make inputting missions data much more involved than the respective process for MFOPs and platforms. The first thing to declare is the mission name and the number of phases contained within it:

```
MISSION1 7
```

Each phase is then declared on a line, followed by the phase length distribution and parameters. If a phase is identified as definitely following after, then this is also declared:

```
Alpha SET 1.5 Beta
```

The phase probability distributions can also "match" or "reverse" another phase's duration, as stated in Section 6.2.1. If this type of distribution is required, it is inputted by typing either "MATCH" or "REV" and then the name of the phase as the parameter. If there is a probabilistic phase choice, then instead of stating one follow-on phase, the word "OUTCOME" allows these to be stated, with a name and probability for each outcome:

```
Alpha SET 1.5
      OUTCOME Good_Weather 0.9 Beta
      OUTCOME Heavy_Rain 0.1 Delta
```

Like the platforms and MFOPs, missions have an object array which stores important data used in running the simulations. One subset of this is the order of the phases in the mission, which includes the possibilities and probabilities for which phase follows the current one. Thus, the number of outcomes, their names and probabilities are stored for each phase, with the indices of the phases as they appear in the phase object array also remembered. Note that if a phase is an ending for the mission, the "next phase" must be declared as "END".

The information explained thus far is necessary – all phases in the mission must be inputted with a phase duration probability distribution and at least one subsequent phase. Extra information may also be entered, however. One example of this, introduced in section 6.2.9, is the idea that phases may be abandoned without immediately ending the mission. Instead, a phase is declared to be started upon abandonment in the following way:

```
Alpha SET 1.5 Beta
      ABANDON Delta
```

If no abandonment phase is specified, the program will assume that the mission ends immediately when the phase's abandonment fault tree top event place is marked.

In addition to this, it is possible to state conditions under which one of three things will happen:

- The phase may abandon to an alternative phase than the one specified above.
- A different subsequent phase may be carried out (this is equivalent to the "event-driven phase choice explained in section 6.2.12.1).
- A phase may be immediately inserted, with the current phase resumed once the inserted phase (or sequence of phases) have finished.

These options are invoked by using the "IF" statements outlined below:

```
Alpha SET 1.5 Beta
      ABANDON Delta
      IF (A + C) OUTCOME Epsilon
      IF (B + D) ABANDON Gamma
      IF (A + A1004) INSERT Mu
```

When the program encounters an "IF" word, it prepares to input a string of characters which are interpreted in a special way. Each "IF" statement has a set of logical conditions and a resulting action. While the action is categorised easily enough, the input conditions are more difficult to handle. The logical conditions entered give a fault tree-like structure to the causes of the action. In order to better organise the conditions, they are separated, within each statement, into "levels", "groups" and "events". The way it works is best explained through the use of an example. Consider the following conditional statement:

```
IF D * (I + A4003) * ( B + C * (G2001 + H)) INSERT Epsilon
```

All names of components, systems and phases are turned into events. The example shows a set of events causing the insertion of phase Epsilon into the current phase. A "level" indicates proximity to the top event of the logical conditions. For instance, because '`D`', indicating the failure of component D, occurs at level 0, it directly inputs to the action. Where a open-bracket is encountered, this indicates that another level must be created. Thus, "`(I + A4003)`" occurs on level 1. However, "`B + ...`" also occurs on level 1, but is not in the same bracket. Thus, these form different "groups" within level 1. Each group must have only one logic type, so if the program encounters multiple types within the same group, it will put the events surrounding the second logic type into a new level. This can be seen with the

statement "`( B + C * (G2001 + H))`", where an OR and AND symbol feature in the same set of brackets. The program effectively puts a set of brackets around the second logic symbol, so it would read "`( B + (C * (G2001 + H)))`". '`C`' is then on level 2, and "`G2001 + H`" is on level 3.

The net for each conditional statement is set up in the PPN of the phase to which it applies. Figure A.8 shows the PN for the conditional statement given above.

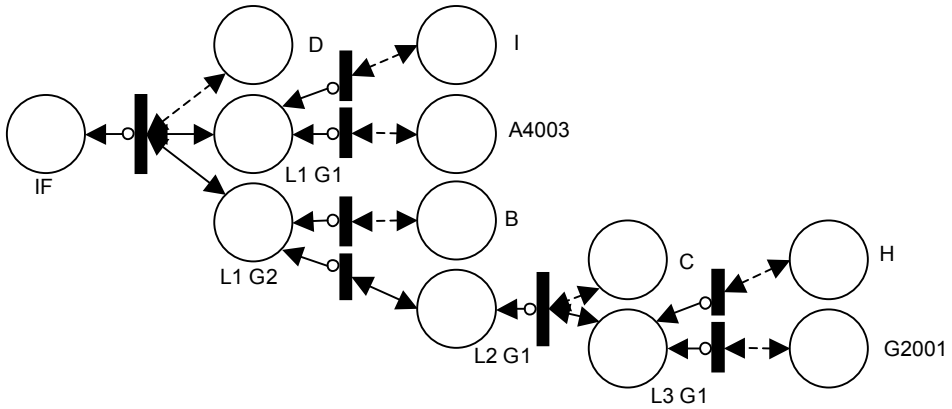


Figure A.8 – PN for Example Conditional IF Statement

As the Figure shows, each "`IF`" statement has a place which enforces the action it is tasked to perform. In order for this to be marked, immediate transitions are set up which represent the logic in each group on each level. Each group on each level has a place created for it, while each event failure place directly enables the logic transition for the group to which it inputs.

In this way, then, the program takes in actions based on user-defined conditions. The only other complexity regarding inputting the data on each phased mission is the ability to repeat phases. The usefulness of this comes, for instance, if a phase has only two possible lengths rather than a distribution, and so two `OUTCOME`s could be specified with a certain probability. The first phase name is inputted as stated above, while subsequent phases have a suffixed index given in square brackets after the name:

```
Beta        SET   1.5     Gamma   ABANDON Delta
            IF    G       OUTCOME Epsilon [2]
Epsilon     MATCH Beta    Beta    [2]
Beta [2]    REV   Epsilon Alpha
Epsilon [2] MATCH Gamma   Gamma
```

As the example shows, a repeated phase is consistently referred to using the "`[2]`", "`[3]`", etc., after the name. The phase will be failed and abandoned using the same fault trees as the normal occurrence.

In summary, a typical mission profile may look like that shown in the example below. The program takes in all of these details and only once all mission data has been inputted does the process of creating the Master PN begin.

```
ASUW 7

Alpha       SET     1.5
            OUTCOME Success 0.9 Beta
            OUTCOME Heavy_Rain 0.1 Delta
            ABANDON Epsilon [2]
            IF D * ( B + C ) INSERT Epsilon
            IF A INSERT Delta
Beta        SET 1.5 Gamma ABANDON Delta
            IF G OUTCOME Epsilon [2]
Gamma       SET 1.5 Delta ABANDON Delta
            IF G2001 + ( B * (E+D)) INSERT Epsilon [2]
Delta       REV Epsilon [2]
            OUTCOME Success    0.8 END
            OUTCOME No_Land    0.2 Delta
Epsilon     MATCH Beta Beta [2]
Beta [2]    REV Epsilon Alpha
Epsilon [2] MATCH Gamma Gamma
```

A.3.3.3 Creating the Master Petri Net

Once all of the platform, MFOP and mission data has been taken in, the program links the different objects together. For instance, if MFOP 1 contains missions A and B, but not C, then the MFOP 1 object will store the indices to the first two

mission objects, and the order in which they appear, but not the third. Each MFOP object is also linked to the relevant phase objects for each part of the mission, as well as the platforms which perform it. Platform objects create an array of mission object indices, while mission objects contain an array of platform and MFOP objects indices. Similar arrays are also set up for phases and systems.

When the program has stored these links, it counts the number of transitions and places which are required in the various PNs and sets the number of colours throughout the model as being equal to the number of platforms. The first loop entered creates the nets for the missions, starting with the places for "mission failure", "mission success" and "mission active". Each phase place and timed transition is set up in the order in which they were inputted, with arcs linking the phases in the sequence as desired. If a transition has a probabilistic phase choice, the timed phase transition is set up with probability arcs. Following this, general & phase abandonment immediate transitions are established, and linked to either the "mission success" place or the phase specified to be carried out instead. The nets for the IFs, as explained above, are created, with a hierarchical structure also set up to prevent more than one of these being able to affect a phase at the same time. This is based on the order which the conditional statements were entered, and works by each IF top place inhibiting the transition which activates the action of all those IFs below it in the hierarchy. Each phase object is made aware of the IFs which affect it, with all necessary details, such as transitions, outcomes, and so on.

MFOP and platform nets are then set up as explained in section 6.2. This is followed by the nets for catastrophic failures, at which point the procedure ends. The only possible information left to be inputted, should they be required, are discrete events relating to the items inputted either by the enablers or the missions text files.

A.3.3.4 Discrete Events

Entering discrete events in the missions text file uses the same function as explained in section A.3.1.4. If they are entered, the same procedures are followed, except that more trigger events can be used:

- If an MFOP or phase name is entered, the system has no way of knowing which of the potentially many repetitions of the "MFOP/phase active" places is marked. To prevent this causing problems, a new place is set up, to which each of the occurrences of the MFOP or phase input to in an OR logic. This then acts as the trigger to set off the discrete event.
- LRI and mission names may be entered also, so that when an LRI is set to be replaced, or the "mission active" place is marked, the discrete event will be triggered.
- Component and system names can also be entered here as well, though there is no reason for these not to have been entered in the phase fault trees & component data text file.

Once all information is inputted, the function ends and the user returns to the main menu. No more data may now be inputted, unless the user selects option '2' from the main menu. This option deletes all inputted information.

A.4 Performing Simulations

A.4.1 Simulation options

After inputting all of the necessary information to create all of the PNs, and store information in the object arrays for the overall PN, places, transitions, platforms, MFOPs, and so on, the system will allow the operator to begin running simulations. Before this is started, the user can alter various options which affect the simulation performance. The options menu is found by entering '4' on the main menu. Figure A.9 shows the options menu.

The number of simulations to perform is preset by the user. This must be an integer and is entered by typing '1' and then the number of simulations required. The confidence interval which is used by the prognostics functions is set in option '2'. This is inputted as a percentage, and must be a multiple of five. If it is not entered in this way, the nearest multiple of five is set instead.

```
C:\Work\Programs\PMFOP With Enablers\Debug\PMFOP With Enablers.exe

- 1 Simulation will be performed.
- The fleet consists of 4 planes
- 4 MFOPs are loaded.
- 4 missions are loaded.
- No Commentary.
- Storing data on: Basic & Intermediate events, Phase, Mission & MFOP failures,
  Platforms, Enablers, Discrete Events

                SIMULATION OPTIONS

1. Set number of simulations to perform
2. Set confidence interval for simulations
3. Change commentary options
4. Change data recording options
0. Return to Previous Menu

Enter your choice :
```

Figure A.9 – Simulation Options Menu

It is possible to add "commentary" to the simulations, which describes what is happening in the simulations as it runs through. Entering '3' leads the user to a sub-menu which allows each possible commentary item to be specified, as shown in Figure A.10. Each item can be individually selected, or they can all be turned on at once by entering '9'. The status at the top of the program screen shows the items which have been selected to be commented on. The default setting is to have no commentary. Note that if "Component Failures/Repairs" or "Timed Transition Switching" is selected, the program will halt whenever a component fails, an MRP is entered, or a timed transition switches. When the user presses any key, the simulations will continue.

```
C:\Work\Programs\PMFOP With Enablers\Debug\PMFOP With Enablers.exe

- 1 Simulation will be performed.
- The fleet consists of 4 planes
- 4 MFOPs are loaded.
- 4 missions are loaded.
- No Commentary.
- Storing data on: Basic & Intermediate events, Phase, Mission & MFOP failures,
  Platforms, Enablers, Discrete Events

                COMMENTARY OPTIONS

Select options to provide commentary on:

1. Component Failures/Repairs
2. Intermediate Events
3. Place Marking
4. Timed Transition Switching
5. Immediate Transition Switching
6. Phases
7. Missions
8. MFOPs
9. Select All
0. Return to Previous Menu

Enter your choice : _
```

Figure A.10 – Commentary Options

The user can also state the options for data recording which are required. The default for this setting is that simulation information on everything is recorded, but this can be changed by entering '4' on the simulation options menu. This enters another sub-menu, as shown in Figure A.11.

```
C:\Work\Programs\PMFOP With Enablers\Debug\PMFOP With Enablers.exe

- 1 Simulation will be performed.
- The fleet consists of 4 planes
- 4 MFOPs are loaded.
- 4 missions are loaded.
- No Commentary.
- Storing data on: Basic & Intermediate events, Phase, Mission & MFOP failures,
  Platforms, Enablers, Discrete Events

              DATA STORAGE OPTIONS

Select desired input:

1. Basic Event failures
2. Intermediate Event failure
3. Phase failures
4. Mission failures
5. MFOP failures
6. Platforms
7. Enablers
8. Discrete events
0. Return to Previous Menu

Enter your choice : _
```

Figure A.11 – Data Storage Options

The individual items are turned on or off by entering the corresponding number. This will, however, only minimally affect the speed with which the simulations are performed, so it is not advised to change any of these options unless there is a clear reason for doing so.

A.4.2 Simulation Process

A.4.2.1 Initial Processes

Once the options are set as required, the simulations can begin by entering '5' on the main menu. This enters a function which does not exit until all the simulations have completed. Figure A.12 describes the process by which simulations take place. The process begins by setting some initial conditions. A function in C++ allows the current time point, expressed as the number of seconds since January 1, 1970, to be given. This is recorded, with the intent of allowing the duration of the simulations to be calculated. An array of the indices of all the timed transitions in the overall PN is set up, allowing each of their times to switch, if they are enabled, to be quickly found.

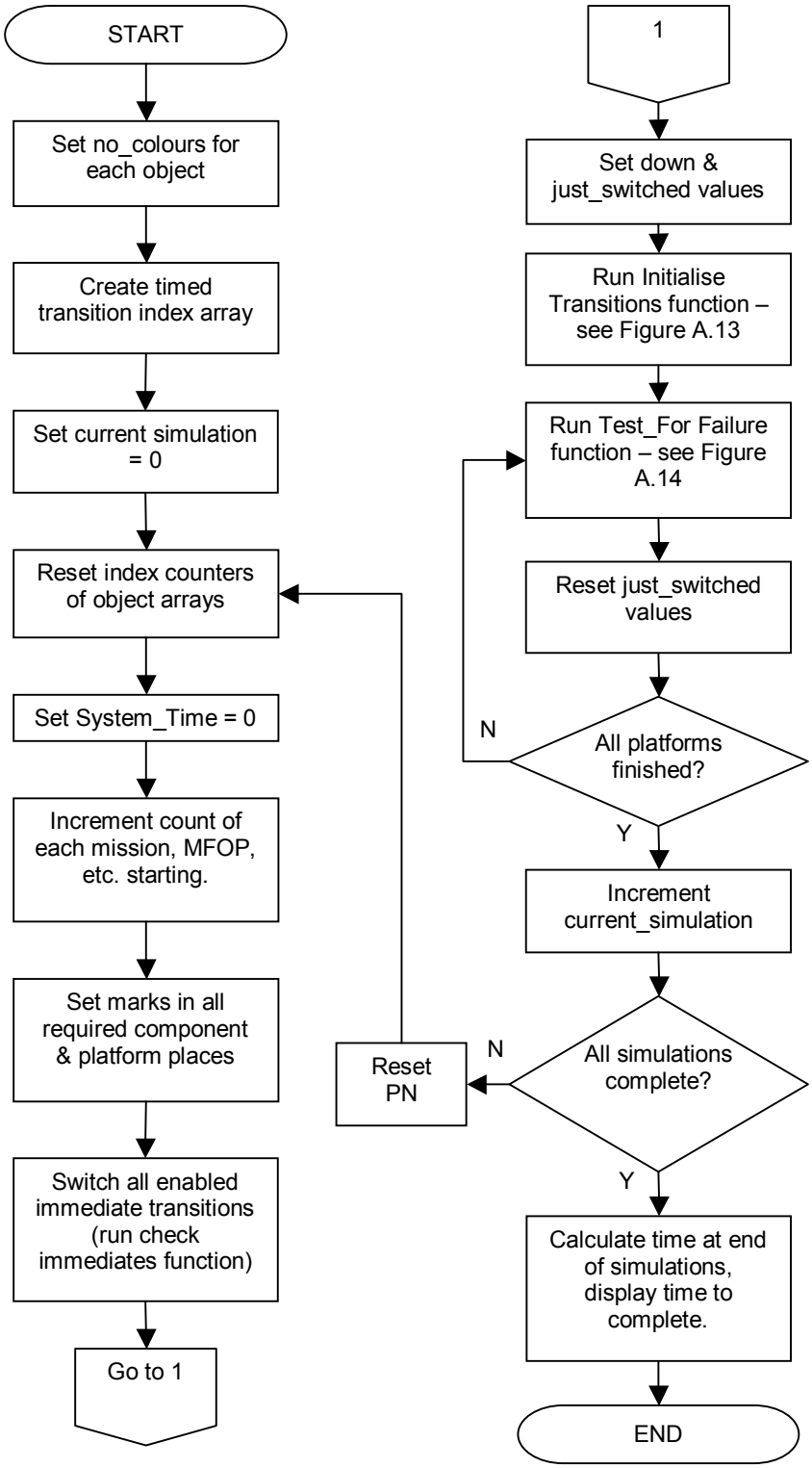


Figure A.12 – Flow chart describing process by which simulations take place

Each platform, MFOP and mission object contains, respectively, a list of all the MFOPs, missions and phases they are to carry out. This is expressed in two ways for each object – the first is the indices of the sub-items as they appear in their own object arrays, while the second is compared to the order in which they appear. For

instance, consider the following arrays of MFOPs and missions, with the numbers expressing the indices of the relevant objects, noting that in C++, all arrays number their indices by starting at zero:

Table A.1 – Example MFOP & Mission Arrays

MFOP Object Array	Mission Object Array	MFOP Mission Index Array	MFOP Mission Order
0. MFOP1	0. MISSION1	{3, 0}	{0, 1}
1. MFOP2	1. MISSION2	{0, 1, 2}	{0, 1, 2}
2. MFOP3	2. MISSION3	{2, 0, 3}	{0, 0, 1, 2}
3. MFOP4	3. MISSION4	{2, 0}	{0, 1, 1}

The first two columns in Table A.1 show the arrays of the MFOP and mission objects in the program. In the third column is, for each MFOP object, a list not of the order in which the missions occur, but of the indices to the objects for all the missions contained within the MFOP, ignoring any repetitions. The lists in the fourth column show the specific order of the missions performed by each MFOP, with the numbers relating to the point in the Mission Index Array. For instance, MFOP1 has the mission order {0,1}. Each number is compared against the Mission Index Array, so MFOP1 performs MISSION4 and then MISSION1. MFOP 3 performs MISSION3 twice, then MISSION1 and MISSION4.

The platform and mission objects have similar arrays for MFOPs and phases respectively. This allows the program to keep track of which phases, missions and MFOPs are being carried out by which platforms at the current point in time. These current value variables are set to their initial values when the simulation process begins. The places which represent “Component Up” and “X Op” for each item with redundancies are all marked with one token in each colour. Platform start places are also marked with the corresponding colour.

The program operates by switching the timed transition with the earliest time, and then switching each immediate transition which becomes enabled as a result. After this, the next timed transition switches, and so on. For this to system to work, the program needs to put each timed transition’s time to switch into chronological

order. Marking the places as mentioned above causes the component failure transitions to be enabled, but not the phase transitions. As such, the immediate transitions must all be switched to prepare the nets for the upcoming simulation. The function that performs this task resembles the following algorithm:

```
1   int immediates_enabled=0,*phaseswitches=0,*transswitches=0;
    do
    {
        for (int a=0;a<immediates_enabled;a++)
        {
            if (check_enabled(imm_trans))
            {
                Switch_Trans(imm_trans);
                imm_trans.set_enabled(switch_colour,0);
10              check_scenarios();
                //If transition is no longer enabled, remove from list.
                if (!check_enabled(imm_trans))
                {
                    [delete imm_trans from array]
                    immediates_enabled--;
                    a--;
                }
            }
            else
20          {
                [delete imm_trans from array]
                immediates_enabled--;
                a--;
            }
        }
        immediates_enabled=0;
        [delete old imm_trans array if exists]
        [create new imm_trans array]
        for (int b=0;b<no_phases+3;b++)
30      for (i=0;i<no_transitions[b];i++)
        {
            bool itenabled=check_enabled(b,i,switch_colour);
            //If transition is enabled
            if (!Tran_array[b][i].return_timed() && itenabled)
            {
```

```
                immediates_enabled++;
                [add enabled imm_trans to array]
            }
40      }
    }while (immediates_enabled>0);
```

The algorithm describes a "`do...which`" loop (the C++ version of a do...until loop), which forcibly runs through once and then keeps looping unless the exit condition is satisfied. In this case, an integer parameter called "`immediates_enabled`", which is the count of the number of immediate transitions currently enabled, must be zero before the loop ends. On the first pass, `immediates_enabled` is zero, and so the `for` loop which switches all enabled transitions is bypassed. Instead, an array of the indices of the enabled immediate transitions is created, with `immediates_enabled` set to the size of that array. On the next pass, the `for` loop switches each immediate transition in the array in turn, checking each time that the switching of a previous transition has not disabled one of those set to be switched. If the switched transition is now disabled, it is removed from the list. A new array of enabled immediate transitions is then created again, and will do so until no more are enabled, when the loop exits.

Each discrete event, component, system, phase and mission object has two Boolean variables which allow the program to keep track of whether they are operational or failed, and whether this has happened just recently or not. These are called "`down`" and "`just_switched`" and are used to prevent data which must be recorded from being so more than once. The program's initial functions set these values to zero for each object. The platform objects each have a "clock", which remembers the last point at which an event occurred. This is used to update various objects with information such as the last repair time, time of failure, and so on. These are all set to zero at the start.

A.4.2.2 Sampling Transition Switching Times

When all of the immediate transitions have been switched and the platform clocks set to zero, the program is ready to generate switching times for each of the timed transitions. The flowchart describing the process involved is shown in Figure A.13.

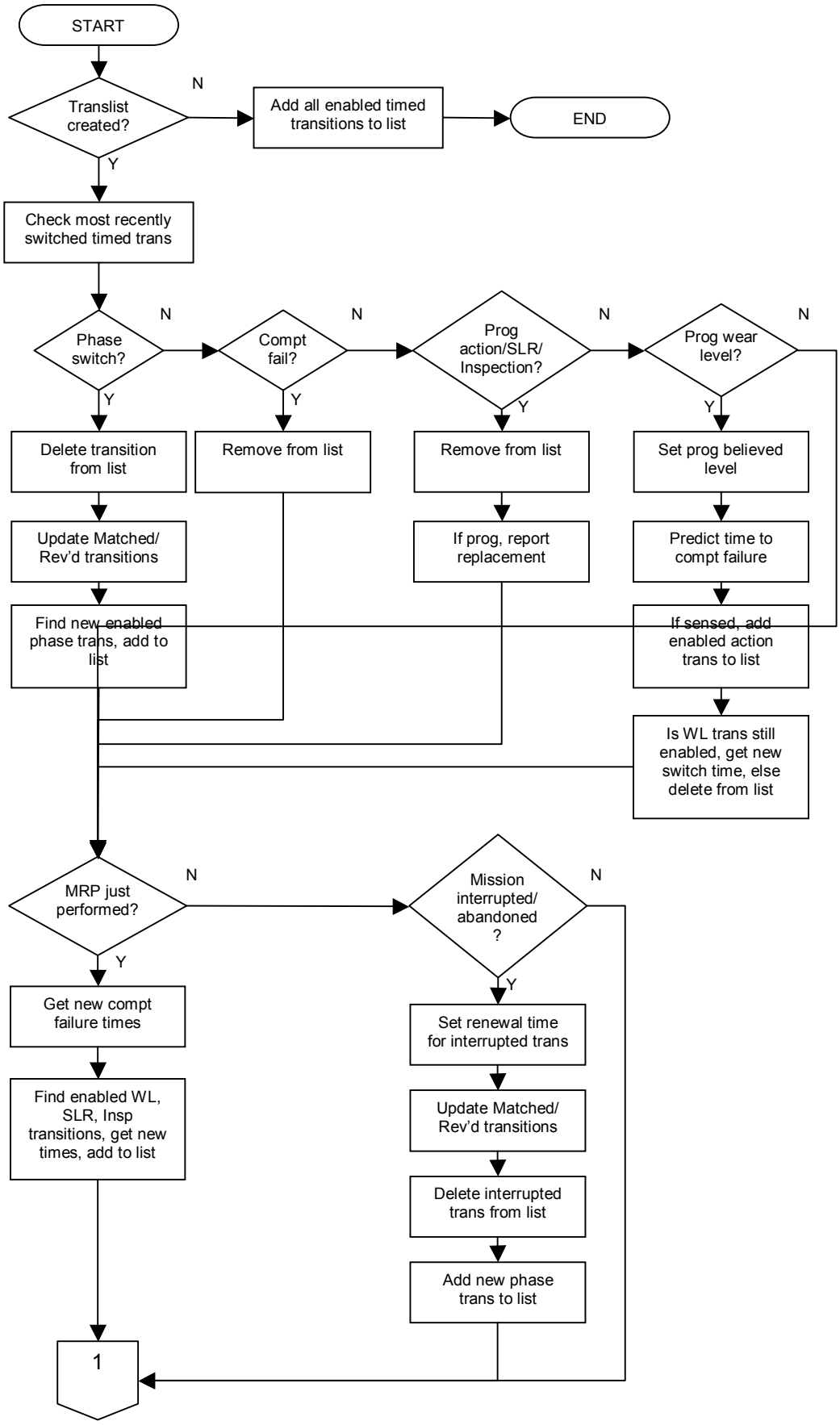


Figure A.13 (a)– Flow chart describing process by which transitions are initialised

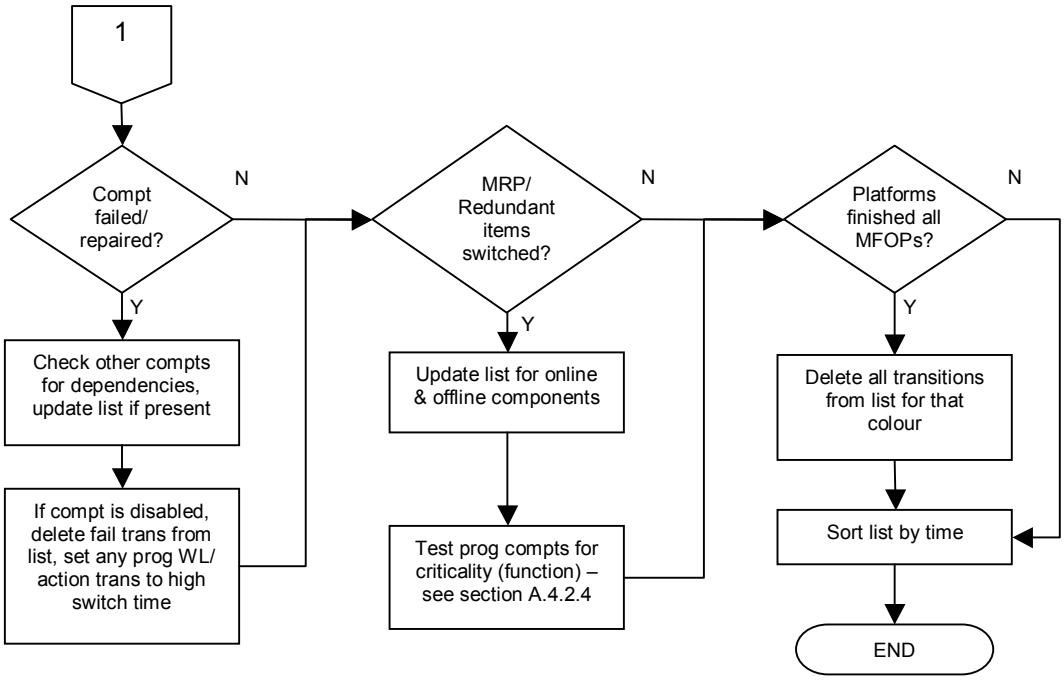


Figure A.13 (b) – Flowchart describing process by which transitions are initialised

The function which performs this, called "`Initialise_Transitions`" creates an array named "`translist`", which contains the details of each enabled timed transition, including its indices and its time to switch. When sampling the switching time for each transition, the function first checks if it is enabled, and upon confirmation it calls a function with the transition object called "`Get_Time`". This function looks at the distribution set for the transition and uses this to generate a time for failure in a particular way for each distribution type. This is typically based upon the Cumulative Distribution Function for the distribution type, and a randomly sampled floating-point number between 0 and 1, as explained in Sections 1.5 and 6.3.8.1. Timed transitions which are set to `match` or `reverse` another transition's switching time are updated each time the platform's clock is updated. This sets a disabled time to the transition which, when it becomes enabled, is added to the platform time to form the matched or reversed switching time. Once the list is created, the transitions have their Boolean "`enabled`" flags set to 1 and they are ordered in ascending switching time.

A.4.2.3 Simulation Performance

Once the initial parameters are all defined, the program enters a loop which does not exit until all of the simulations are complete. With each pass of the loop, a function is performed, called "`Test_For_Failure`". This function is part of the `Petri_Net`

object and it performs the core requirements of each simulation pass: switch a timed transition, switch all enabled immediate transitions, and record data. The general set of functions performed can be seen in the flowchart in Figure A.14.

When each transition is switched, a basic count of time, based on the number of flying hours, is kept up to date with this. The program will check to see if the switch was relevant in some way. For instance, if a component failure transition switches, a component has failed and this failure must be recorded. The data for this is useful not just to the relevant `component` object, which counts the number of failures and the average MTBF, but also any system objects to which it inputs, any sensors which may connect to it, and so on. Similarly, if a phase transition switches, this could be either a normal phase transition or an event-driven phase choice, as shown in Section 6.2.12.1. Whatever the cause, the starting phase must have its start recorded in its phase object and the old phase set as being successful. The results of the phase change may be the end of a mission, so if this is the case, the number of the current mission that the platform is undertaking must be changed, and the first phase of it activated.

If a phase or a component has switched, the "`just_switched`" Boolean value in the relevant object must be set to 1, and all others set to zero. After this, the "`check_immediates`" function mentioned in section A.4.2.1 is performed. Because simulations are now active, the "`check_scenarios`" function, called on line 10 of the `check_immediates` algorithm, becomes important. This function tests for various conditions after each immediate transition switch, which includes:

- Component or system inspection
- Component repair or replacement
- LRI replacement
- Phase insertion
- Phase failure
- Mission abandonment
- Starting and ending of MRP
- Inspection of prognostic components in MRP and criticality testing, which may lead to replacement (see Section A.4.2.4).

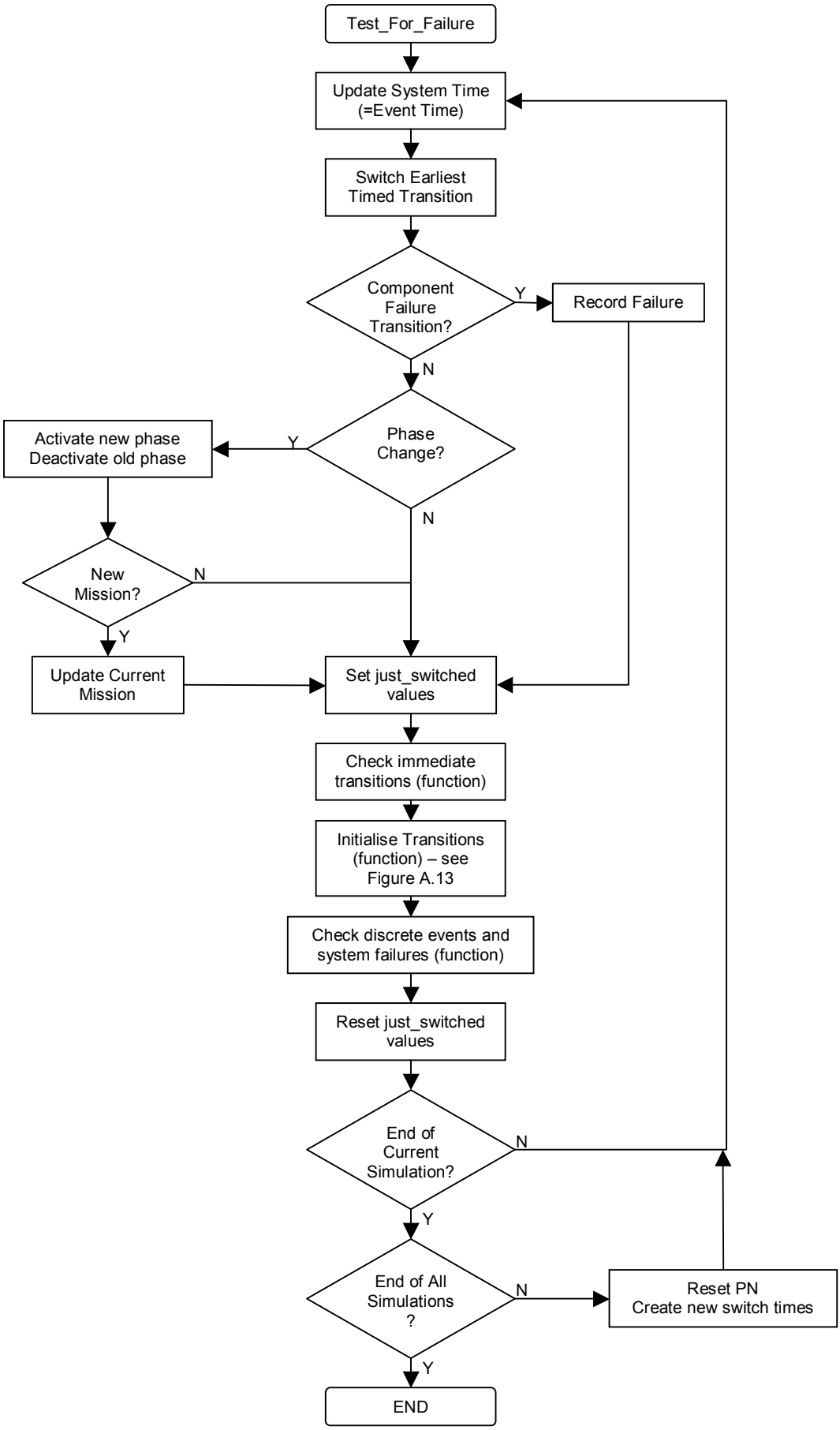


Figure A.14 – Test_For_Failure Function Flowchart

Once all the immediate transitions that are enabled have switched, and the data from the resultant marking of the net is recorded, the next stage is to run the `Initialise_Transitions` function mentioned in section A.4.2.2. When this is first run, it merely creates an array of transition times. If the array is found to already exist, a different part of the function is entered, which generates switching times for transitions based on those which have just switched. For instance, if a phase transition has just switched, a new phase transition will be enabled, so this transition is found and a time is sampled for it. Any transitions whose times are set to match or reverse either the ending or the starting phase are updated. Other conditions lead to different procedures being followed, and new transition times being created. The `translist` array which keeps a record of the transition switching times must have the old time removed and the new one put in place. The array is then sorted in ascending order of time to switch.

A final set of functions records the effects of component failures on systems, the effects of system failures on phases, and the switching of discrete events. When these functions are finished, the program resets the `just_switched` values for each object and tests to see whether the "Simulation success" place is marked. If it is not, the loop reruns `Test_For_Failure`. Otherwise, the overall PN is reset, necessary values are changed, and the initialisation process for the net, as described in sections A.4.2.1 and A.4.2.2 is performed ready for the next simulation. If the finished simulation is the last, a summary of the elapsed time for all the simulations to complete is displayed on screen, and a keypress returns the user to the main menu.

A.4.2.4 Assessing Component Criticality

Section 6.3.8 explains the methods by which the program assesses a component's predicted future failure. This includes modelling and detecting component wear, and using this to predict future failure times. Also explained is how the actions that the prognostic model can take – abandoning a mission immediately, or having a component replaced during an MRP – are carried out in Petri net form. This section explains how the program reaches its decisions on the criticality of components to the current or future missions, and assigns a switching time to the prognostic "action" transition accordingly.

When a platform undergoes an MRP, and all components have been repaired or inspected for wear levels; or when the `Initialise_Transitions` function is performed, the program will enter a function called "`test_criticality`". This function makes use of the confidence interval which has been defined by the user (as mentioned in section A.4.1).

The first task that the function performs is considering each of the phases within the mission and assigning a duration to them. If, for example, the confidence interval was 90%, the program would find the time at which 90% of phases with this duration probability distribution would have completed.

The second stage of the function is to create a list of phase sequences, taking into account any insertions, abandonments or event- or probability-driven phase choices. The method of this stage is explained by way of an example. Consider the mission with the phase sequence as shown in Figure A.15. The phases are Alpha, Beta, Gamma and Delta, with the phase indices 0,1, 2 and 3 respectively. '-1' is an index symbolising the end of the mission.

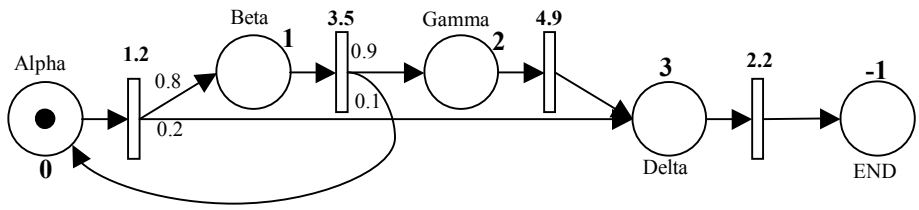


Figure A.15 – Example Mission Net

The function begins with the active phase, as the operational state of the current mission means that previous phases have not failed it, and thus cannot have any more effect on the mission performance. The program creates a series of arrays, each of which relates to the phase sequences. These include: a matrix of phase sequences, $\mathbf{M_{PS}}$ and the number of phases in, probability of, and estimated time duration of each sequence. For the first phase, these will look like equations A.1. Note that, in this example, the confidence interval is assumed to be 95%.

$$\begin{array}{cccc} \{0\} & \{1\} & \{1\} & \{1.2\} \\ \mathbf{M}_{PS} & \text{Number of phases in sequence, } \mathbf{N}_{PS} & \text{Probability array, } \mathbf{P}_{PS} & \text{Sequence time, } \mathbf{T}_{PS} \end{array}$$

A.1

With the second phase, however, there is a probabilistic phase choice. Because of this, the program creates a second line in each of the arrays and fills in the relevant information. This is done by multiplying each line in the probability array by the relevant probability of entering the new phase on that line, and by adding on that phase length to the sequence time array. If any sequence probability is less than 1×10^{-6}, that line is deleted from the arrays. The arrays will look like equations A.2 at this stage.

$$\begin{array}{cccc} \begin{Bmatrix} 0 & 1 \\ 0 & 3 \end{Bmatrix} & \begin{Bmatrix} 2 \\ 2 \end{Bmatrix} & \begin{Bmatrix} 1\times 0.8 = 0.8 \\ 1\times 0.2 = 0.2 \end{Bmatrix} & \begin{Bmatrix} 1.2+3.5=4.7 \\ 1.2+2.2=3.4 \end{Bmatrix} \begin{array}{l} \leftarrow \text{Active Line} \\ \leftarrow \text{New Line} \end{array} \\ \mathbf{M}_{PS} & \mathbf{N}_{PS} & \mathbf{P}_{PS} & \mathbf{T}_{PS} \end{array}$$

A.2

The sequence on which it concentrates remains the top line. At the next stage, there is another probabilistic phase choice. Phase Beta can either progress to Gamma or revert back to Alpha. The arrays then become those shown in equations A.3.

$$\begin{array}{cccc} \begin{Bmatrix} 0 & 1 & 2 \\ 0 & 3 & \\ 0 & 1 & 0 \end{Bmatrix} & \begin{Bmatrix} 3 \\ 2 \\ 3 \end{Bmatrix} & \begin{Bmatrix} 0.8\times 0.9 = 0.72 \\ 0.2 \\ 0.8\times 0.1 = 0.08 \end{Bmatrix} & \begin{Bmatrix} 4.7+4.9=9.6 \\ 3.4 \\ 4.7+1.2=5.9 \end{Bmatrix} \begin{array}{l} \leftarrow \text{Active Line} \\ \\ \leftarrow \text{New Line} \end{array} \\ \mathbf{M}_{PS} & \mathbf{N}_{PS} & \mathbf{P}_{PS} & \mathbf{T}_{PS} \end{array}$$

A.3

Phases Gamma and Delta are added to the arrays, until the mission end is found. This finalises the row:

$$\begin{array}{cccc} \begin{Bmatrix} 0 & 1 & 2 & 3 & -1 \\ 0 & 3 & & & \\ 0 & 1 & 0 & & \end{Bmatrix} & \begin{Bmatrix} 5 \\ 2 \\ 3 \end{Bmatrix} & \begin{Bmatrix} 0.72 \\ 0.2 \\ 0.08 \end{Bmatrix} & \begin{Bmatrix} 9.6+2.2=11.8 \\ 3.4 \\ 5.9 \end{Bmatrix} \leftarrow \text{Active Line} \\ \mathbf{M}_{PS} & \mathbf{N}_{PS} & \mathbf{P}_{PS} & \mathbf{T}_{PS} \end{array}$$

A.4

When a line finishes, the program re-arranges the arrays so that the finished sequences are ordered by increasing duration. The program will then sum the probabilities of the finished sequences, and if this sum is greater than the confidence interval, any lines with durations longer than the newly finished sequence are deleted. However, any unfinished lines with shorter durations are kept, as they may yet yield a shorter mission span than the currently finished ones. In the example shown, the sum of the finished line probabilities is 0.72, which is less than the confidence interval of 0.95. As such, the line with the shortest time is selected next (the second sequence), and the function continues. After the second sequence is completed and re-ordered, the arrays look like those in equations A.5.

$$\underset{\mathbf{M_{PS}}}{\begin{Bmatrix} 0 & 3 & -1 & & \\ 0 & 1 & 2 & 3 & -1 \\ 0 & 1 & 0 & & \end{Bmatrix}} \quad \underset{\mathbf{N_{PS}}}{\begin{Bmatrix} 3 \\ 5 \\ 3 \end{Bmatrix}} \quad \underset{\mathbf{P_{PS}}}{\begin{Bmatrix} 0.2 \\ 0.72 \\ 0.08 \end{Bmatrix}} \quad \underset{\mathbf{T_{PS}}}{\begin{Bmatrix} 3.4 \\ 11.8 \\ 5.9 \end{Bmatrix}} \leftarrow \text{Active Line} \qquad \textbf{A.5}$$

The sum of the finished lines is now 0.92, still less than the confidence interval of 0.95. The program selects the only unfinished line, the third one, and continues. When this is finalised, the arrays are those in equations A.6.

$$\underset{\mathbf{M_{PS}}}{\begin{Bmatrix} 0 & 3 & -1 & & & & \\ 0 & 1 & 2 & 3 & -1 & & \\ 0 & 1 & 0 & 1 & 2 & 3 & -1 \\ 0 & 1 & 0 & 3 & & & \\ 0 & 1 & 0 & 1 & 0 & & \end{Bmatrix}} \quad \underset{\mathbf{N_{PS}}}{\begin{Bmatrix} 3 \\ 5 \\ 7 \\ 4 \\ 5 \end{Bmatrix}} \quad \underset{\mathbf{P_{PS}}}{\begin{Bmatrix} 0.2 \\ 0.72 \\ 0.0576 \\ 0.016 \\ 0.0064 \end{Bmatrix}} \quad \underset{\mathbf{T_{PS}}}{\begin{Bmatrix} 3.4 \\ 11.8 \\ 16.5 \\ 8.1 \\ 10.6 \end{Bmatrix}} \leftarrow \text{Active Line} \qquad \textbf{A.6}$$

The sum of the probabilities of the finished lines is now 0.9776, greater than the confidence interval. However, both of the unfinished lines have times less than the greatest of these, 16.5 hours, and so the program continues. The fourth line is quickly finalised, and the fifth line expanded so the arrays look like equations A.7.

$$
\underbrace{\begin{Bmatrix}
0 & 3 & -1 & & & & & & \\
0 & 1 & 0 & 3 & -1 & & & & \\
0 & 1 & 2 & 3 & -1 & & & & \\
0 & 1 & 0 & 1 & 0 & 3 & -1 & & \\
0 & 1 & 0 & 1 & 2 & 3 & -1 & & \\
0 & 1 & 0 & 1 & 0 & 1 & 2 & 3 & -1 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & &
\end{Bmatrix}}_{\mathbf{M_{PS}}}
\underbrace{\begin{Bmatrix} 3 \\ 5 \\ 5 \\ 7 \\ 7 \\ 9 \\ 7 \end{Bmatrix}}_{\mathbf{N_{PS}}}
\underbrace{\begin{Bmatrix} 0.2 \\ 0.016 \\ 0.72 \\ 0.00128 \\ 0.0576 \\ 0.004608 \\ 0.000512 \end{Bmatrix}}_{\mathbf{P_{PS}}}
\underbrace{\begin{Bmatrix} 3.4 \\ 8.1 \\ 11.8 \\ 12.8 \\ 16.5 \\ 21.2 \\ 15.3 \end{Bmatrix}}_{\mathbf{T_{PS}}}
\begin{matrix} \\ \\ \\ \\ \\ \leftarrow \text{Delete} \\ \\ \end{matrix}
\qquad \textbf{A.7}
$$

The sixth line of the arrays in equations A.7 can be deleted, as the confidence interval time is still 16.5 hours, and the duration of this phase sequence is 21.2 hours. If sequence seven is then expanded, all three sequences will have times greater than 16.5 hours, so the program deletes these lines and exits the loop. The result from this part of the function is that at least 95% of possible phase sequences will have ended inside 16.5 hours.

The next stage of the function uses each of the finalised phase sequences to make estimates as to whether or not a particular component with prognostics is likely to have failed by the end of each phase in that sequence. Any that may have done are noted. The program then makes a note of the components and systems which have had revealed or sensed failures. Each phase in each sequence is then again taken in turn. The component PN and the relevant phase PN are then reset so that no components or systems in them are failed, except those that are known to be potentially failed by the end of them. The failures are cascaded up, and if the phase fails catastrophically, this is noted. After this has been done for each phase in each sequence, the PNs are set back to their previous values.

If a mission is currently active, all of the different phase sequences has failed catastrophically, then the prognostic action transition is given a switching time equal to the current system time – effectively an immediate switching. If none are failed, the action transition is given a very large switching time. If some are failed and some are active, the transition is made to switch at the notional mission end time. This has the effect of waiting to see how the mission progresses before potentially abandoning it.

If an MRP is taking place, any components which may cause mission failure, even in just one potential scenario, are replaced. Only if a component cannot cause any of the missions in the forthcoming MFOP to fail is it left to continue.

The approach described here is an attempt to automate and model what is, in reality, often a very complex decision-making process. This allows a degree of control and accuracy in the area of predicting future component failure times based on available information.

A.5 Outputs

When data is inputted or simulations are performed, it is possible for the program to create text files which provide data to the user. There are two types of output available: Petri net information and simulation results. Pressing option '7' on the main menu, as shown in Figure A.1, leads to the options menu displayed in Figure A.16.

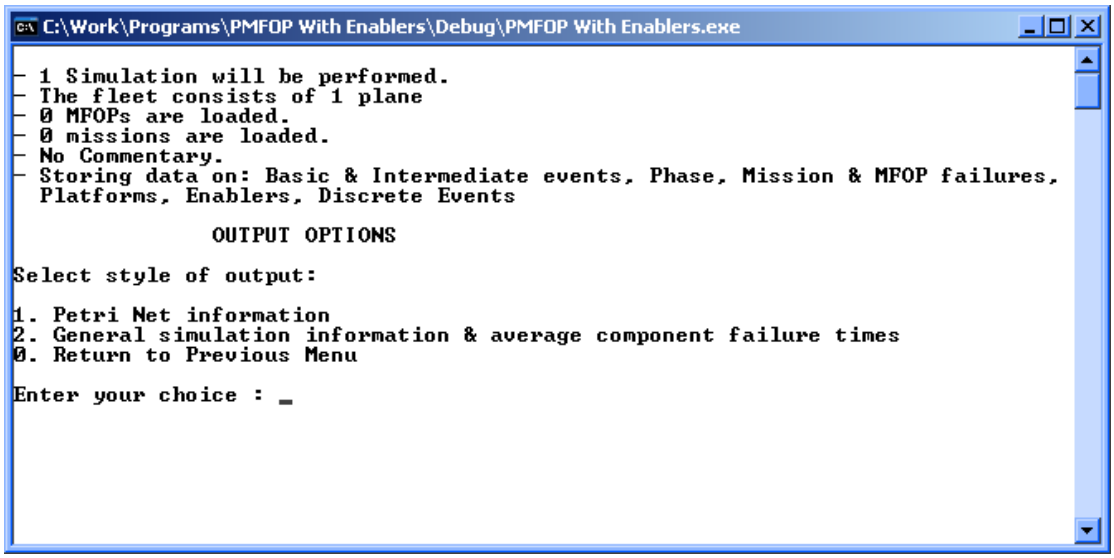


Figure A.16 – Output Menu

A.5.1 Petri Net information

One of the most important requirements of a program applying the Petri net method described in this thesis is to allow the Petri nets which have been created to be checked. This was important for debugging the program while it was being written, but is also vital when putting a system into the program for analysis, to check it has been converted as expected.

The format of the Petri net information text file follows a set pattern, which is shown below.

```
0: PHASE Alpha

PLACES
                        Gate
Place      Gate Name   Number      Marks
    0           GTOP    1000      (1,1,0)
    1          G1001    1001      (0,0,0)
...

TRANSITIONS
Transition
  Number      Timing     Stress
       0     Immediate      1
       1     EXP(0.03)      1
       2     SET(50)        1
       3     Immediate, Probability Transition (0.1,0.9,0)      1
...

INPUT/OUTPUT MATRIX

           0      1      2      3      4      5   ...
   0   1 0 1  1 0 1
   1          1 1 0  1 0 1                0 0 1
   2                        1 0 1  1 0 1
   3                                      0 1 0
...
```

```
LINKS

(0 0)-(6 7) - 0 0 1
...
```

Each PN is displayed in turn, with the number and name of the net acting as a title. The first subsection is a list of the places within the net. These are referred to by index within the net array of places, name, "gate number" and the marks for each colour as allocated at the start of simulations. The gate number helps the program to recognise the purpose of the place, according to a list of pre-defined ranges. These ranges are summarised in Table A.2, where *N* refers to the global defined parameter `NO_BASICS`, defined in Section A.3.1.1.

Table A.2 – List of Gate Numbers by Place Type

Range from	Range up to but not including	Type of place
0	N	Component
N	5N	System
5N	6N	Sensor
6N	7N	Discrete Event
7N	8N	LRI
50N	60N	Phase
60N	70N	Mission
70N	80N	MFOP
80N	90N	Platform
90N		IF

The "marks" column will not typically display any numbers greater than zero, unless the PN information is output after simulations have been run.

The next set of information is that on the transitions within the net. Each transition is identified by its location in the array for that PN. If it is timed, then the assigned distribution and parameters are shown. If not, it is labelled as "immediate". The "stress" assigned to the transition is also given. This will typically be 1, unless it is a failure transition for a component with dependencies. If the transition is probabilistic, then it is labelled as such next to the timing information. The probabilities for each of the output arcs it has are given in brackets.

A matrix of arc multiplicities for each of the places and transitions in the net is then shown. Along the top of the matrix are the index numbers of transitions, while down the left-hand side are the place indices. Because a text file can only have a certain number of characters on a line, only 140 transitions can be shown in each matrix. If this limit is exceeded, the program will show another matrix below, starting from transition 141, 281, and so on. Each element of the matrix has three values, corresponding to, from the perspective of the place, input arc multiplicity, output arc multiplicity, and inhibitor arc multiplicity. For instance, if place 4 has arcs {3,4,2} to transition 6, this is the equivalent of the net in Figure A.17. If the element is blank, all arcs are zero.

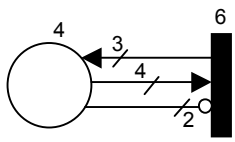


Figure A.17 – Net with arcs {3,4,2}

The final set of information is that for arcs which link between places in the current net and transitions in other nets. In the example given, the link is "`(0 0)-(6 7) - 0 0 1`". This indicates that a single inhibitor arc goes from place 0 in net 0 to transition 7 in net 6. From the information given by the text file, the full net can be reproduced in graphic form, and analysed for correctness.

A.5.2 Simulation Results

The other output from the program is a text file which shows the results of the simulations which have been run. This is option '2' on the output menu as shown in Figure A.16. This gives results in a table format, with the following data provided:

- Platform, MFOP and mission failures and abandonments
- Phase failures and the systems which caused these
- System and component failures
- Discrete event switches
- LRI failures and replacements

- Prognostic components, the number of abandonments and replacements
- Redundant items, the number of switches, and the success rate of these
- Inspected items, and the number of inspections and located failures.

Further information on each of these is given below.

A.5.2.1 Platform Failure Information

The first set of data given concerns the performance of each platform. This is tabulated for each one, and shows the number of starts, the number of times an MFOP or mission was abandoned, and the number of catastrophic failures. This information is also broken down for each of the MFOPs and missions that the platform undertook, as shown in the example below.

```
PLATFORM PERFORMANCE SUMMARY

  PLAT- | MISSION/ |        |  ABANDON- | FAIL- | UNRELI-
  FORM  | MFOP     | STARTS |   MENTS   | URES  | ABILITY
--------+----------+--------+-----------+-------+---------
 PLTFM1 |    MFOPs |     63 |        17 |    46 |   0.730159
        | Missions |     68 |        17 |    46 |   0.676471
        |          |        |           |       |
        |    MFOP1 |     50 |         8 |    42 |   0.84
        |    MFOP2 |      8 |         5 |     3 |   0.375
        |    MFOP3 |      5 |         4 |     1 |   0.2
        |          |        |           |       |
        | MISSION1 |     50 |         7 |    40 |   0.8
        | MISSION2 |      3 |         0 |     1 |   0.333333
        | MISSION3 |     15 |        10 |     5 |   0.333333
        |          |        |           |       |
 PLTFM2 |    MFOPs |    101 |        56 |    45 |   0.445545
...
```

As the example shows, a figure for unreliability is given. This is the total number of catastrophic failures divided by the number of starts for that MFOP or mission.

A.5.2.2 MFOP Failure Information

The information on MFOP failures is similar to that for platforms. An example table is shown below.

MFOP	MISSION/ PLATFORM	STARTS	ABANDON-MENTS	FAIL-URES	ABANDON PROB	FAILURE PROB
MFOP1		60	2	2	0.03333	0.03333
	Missions	176	2	2	0.01136	0.01136
	PLTFM1	15	1	1	0.06667	0.06667
	PLTFM3	30	1	1	0.03333	0.03333
	PLTFM4	15	0	0	0	0
	MISSION1	60	1	1	0.01667	0.01667
	Alpha	60	0	1	0	0.01667
	Beta	55	0	0	0	0
	Gamma	55	0	0	0	0
	Delta	71	1	0	0.01408	0
	Epsilon	0	0	0	----	----
	MISSION2	58	0	0	0	0
	Beta	58	0	0	0	0

...

The failure data for each MFOP is, like that for the platforms, split up so it can be seen which platforms and missions featured the most or the fewest failures. The number of times each platform started the MFOP is shown, as is the number of times the MFOP started each of its missions and the phases within each of these. Probabilities of the MFOP failing or being abandoned for a given platform or in a particular mission or phase are shown.

A.5.2.3 Mission Failure Information

The results for the individual missions is laid out as in the example below.

```
         | MFOP/     |         |           |        |          |
         | PHASE/    |         | ABANDON-  | FAIL-  | ABANDON  | FAILURE
 Mission | PLATFORM  | STARTS  |  MENTS    | URES   | PROB     | PROB
---------+-----------+---------+-----------+--------+----------+--------
-
MISSION1 |           |    138  |        4  |      5 | 0.02899  | 0.03623
         |   Phases  |    683  |        4  |      5 | 0.00586  | 0.00732
         |           |         |           |        |          |
         |   PLTFM1  |     41  |        2  |      2 | 0.04878  | 0.04878
         |   PLTFM3  |     30  |        1  |      1 | 0.03333  | 0.03333
         |   PLTFM4  |     15  |        0  |      0 |        0 | 0
         |   PLTFM2  |     52  |        1  |      2 | 0.01923  | 0.03846
         |           |         |           |        |          |
         |    MFOP1  |     60  |        1  |      1 | 0.01667  | 0.01667
         |    MFOP2  |     78  |        3  |      4 | 0.03846  | 0.05128
         |           |         |           |        |          |
         |    Alpha  |    138  |        2  |      2 | 0.01449  | 0.01449
         |-    Beta  |    121  |           |        |          |
         |-   Delta  |     13  |           |        |          |
         |     Beta  |    121  |        1  |      0 | 0.00826  | 0
         |    Gamma  |    122  |        0  |      3 |        0 | 0.02459
         |    Delta  |    162  |        1  |      0 | 0.00617  | 0.00617
         |-     END  |    131  |           |        |          |
         |-   Delta  |     29  |           |        |          |
         |  Epsilon  |      0  |        0  |      0 |   ---    |  ---
...
```

The table of results is once again very similar to those for platforms and MFOPs. Each mission's figures are broken down into the platforms, MFOPs and Phases which link to it. Values are given for numbers of starts, abandonments and catastrophic failures, with probabilities for the latter two. Something different to note, however, is that phases which have several options for which phase follows on show the number of times each output option was selected. This is denoted by a hyphen before the phase name. For instance, either of phases Beta or Delta can follow phase Alpha, and of the 134 times Alpha didn't fail, 121 led to Beta being performed, and 13 to Delta.

A.5.2.4 Phase Failure Information

The output given by the program relating to phase failure information is split into two parts. The first relates the phase failures to the missions, phase choices and platforms, while the second shows the systems which prompted the phase failures. The first set of information is similar to that shown in the previous three sections, an example of which is shown below.

```
              |  PLATFORM/    |           |           |
     PHASE    |   MISSION     |   STARTS  |  FAILURES | UNRELIAB.
--------------+---------------+-----------+-----------+-------------
        Alpha |               |       268 |        43 |   0.160448
              |     Missions  |       268 |        43 |   0.160448
              |               |           |           |
              |       PLTFM1  |        73 |        13 |   0.178082
              |       PLTFM2  |       129 |        25 |   0.193798
              |       PLTFM3  |        66 |         5 |   0.0757576
              |               |           |           |
              |     MISSION1  |       112 |        10 |   0.0892857
              |     MISSION3  |       156 |        33 |   0.211538
              |               |           |           |
              |         IF 0  |           |        10 |
              |         IF 1  |           |         0 |
...
```

As before, the number of times the item of concern has failed is related to each platform which performs it, and the missions which it forms part of. The number of times the phase failed in each of these is also shown, together with a figure for unreliability, conditional to the mission or platform. The program does not distinguish between an abandoned or a catastrophically failed phase. Another piece of information given here is the number of times that the phase was affected by an "`IF`" event as explained in Section A.3.3.2. These are listed in the order in which they were inputted, without any further information on whether they relate to a phase abandonment or insertion, or an event-driven phase choice.

The second set of information is typically laid out as in the example below.

```
              |              |           | CRITICAL
     PHASE    |    SYSTEM    | FAILURES  | FAILURES
--------------+--------------+-----------+-----------
        Alpha |              |        43 |        43
              |         GTOP |        33 |        12
              |        G1001 |         0 |         0
              |        ATOP1 |       231 |        31
              |              |           |
         Beta |              |         6 |         6
              |        GTOP2 |         0 |         0
              |        G2001 |         0 |         0
              |        G2002 |         0 |         0
              |        G2003 |         0 |         0
              |        ATOP2 |         7 |         5
```

This table shows the “systems” (top and intermediate gates in the phase fault trees) which can contribute to phase failure. Each time one of these systems fails, the “failures” value in the relevant phase object is incremented. If the phase fails, the program searches through each of the systems which could contribute to its failure, and notes those which have failed. Each of these is classed as a “critical failure”, even if it did not necessarily help cause phase failure. In the example above, system “`ATOP1`” in phase Alpha failed 231 times, but only caused Alpha to fail or be abandoned 31 times. This could be due to Alpha not being active before another phase failed or an MRP was entered.

A.5.2.5 System Failure Information

System failure information is given in terms of the phases each system can cause failure in, and the sub-systems and components which are inputting events to it. A typical table of information is shown below.

SYSTEM	PHASE/LOWER SYSTEM/ COMPONENT	FAILURES	CRITICAL FAILURES	MTTF
GTOP		33		5.44432
	Alpha	12		
	G1001	0	0	
	A	33	33	
	B	0	0	
	D	0	0	
G1001		0		0
	Alpha	0		
	B	0	0	
	D	0	0	
ATOP1		231		4.12259
	Alpha	31		
	SENS_E	36	34	
	-E			
	F	216	197	

...

The number of times the system failed is shown at the top. Each time a failure is recorded, the time since the last repair is added to an average value, which is shown in the last column, under “MTTF”, or Mean Time to Failure. The sub-systems which could cause the failure of this item are shown. As with the phase failure information, both “failures” and “critical failures” are recorded. The former is the number of times the child item failed, while the latter is the number of times the child item was failed when the parent item failed.

The components which could cause system failure are then shown. This list includes not only the components which are direct inputs to the system, but those inputting to

sub-systems. Failure and critical failure information is then given as with the sub-systems. For instance, in the example shown, component 'A' has failed 33 times, and because it is a direct input to system "GTOP", which has an OR failure logic, each of these was a critical failure. GTOP subsequently failed phase Alpha 12 times.

Note that where the system is part of an abandonment fault tree, sensors which are children of that system will input to it when the item they sense fails, and not when the sensor itself fails. This is explained in Section A.3.1.1. When outputting, the program highlights the item which is sensed by the sensor component below its name in the table. In the example above, "ATOP1" has input component "SENS_E", which passes on the failures of component 'E'.

A.5.2.6 Component Failure Information

Information showing the simulation results pertaining to the individual components is shown in a table similar to that for the systems. An example of this is given below.

```
   COMPONENT  |    SYSTEM    |    FAILURES    |   MTTF
--------------+--------------+----------------+-----------
            A |              |        33 (0)  |   5.44432
              |         GTOP |             33 |
              |        GTOP2 |              0 |
              |        G2001 |              0 |
              |        G2002 |              0 |
              |        GTOP4 |              0 |
              |        GTOP5 |             33 |
              |     AGENERAL |              0 |
```

The data in this table relates the component failures only to those systems to which it inputs, directly or indirectly. For instance, in the example above, component A is a direct input to system G2002, which is a sub-system of both G2001 and GTOP2, and so these are also listed. The number of failures of component A, together with a figure for the average mean time to fail is given. Below this, the number of times the failure of A caused a failure of the systems for which it is an input is given. In the example shown, A failed GTOP and GTOP5, but none of the other systems.

If a component is redundant, it is possible that an attempt to bring it into operation may fail, and thus cause parent system failure, as explained in Section 6.3.7. If this is the case, the number of times the component was not brought online is shown in brackets next to the number of actual failures.

A.5.2.7 Discrete Event Information

Every time a discrete event switches, the results of that switch are recorded by the program into the corresponding discrete event object. The data is compiled and added to the simulation below the component failure information, taking a form similar to that shown below.

EVENT	NUMBER SWITCHES	NUMBER YES	NUMBER NO	SYSTEM	FAILS
D1	33	24	9	TOP	14
				G1001	14
D2	0	0	0		
D3	112	7	105		

The number of times each event switched and the number of times it resulted in a positive or negative outcome is shown. If the discrete event is an input to one or more systems, the number of times its failure prompted the system to fail is shown.

A.5.2.8 LRI information

Similarly to the discrete events, the number of times an LRI is switched and the inputs which prompted these are shown in a table, similar to the example below.

LRI NAME	# REPLA-CEMENTS	EVENT NAME	NUMBER OF FAILURES
LRI_1	0	LRI_2	0
		LRI_3	0
LRI_2	31	LRI_4	0
		LRI_5	31

A.5.2.9 Prognostics Information

Each component which has its performance estimated by a prognostics system will have the data from this outputted to a table similar to the one below.

COMPT	STARTS	WEAR LEVEL	NUMBER	REPLACE-MENTS	MFOP ABANDONS	REPLACE RATE
C	150	1	3	0	0	0
		2	2	0	0	0
		3	1	1	0	1

The table shows the name of the component, and the number of times that it "started"; taking account of the amount of times a simulation began for each platform, and the number of times it was repaired. Each wear level the component can experience is shown with the number of times it was found to be worn to that stage. If the component's predicted future failure in any of the wear levels prompted its replacement or an MFOP to be abandoned, the number of times this happened is shown. Also displayed is a rate, equal to the number of replacements divided by the number of times the wear level was entered.

A.5.2.10 Redundancies Information

Each item which has redundancies assigned to it will have the details of these shown in a table similar to that below.

ITEM NAME	TYPE	NUMBER OF SWITCHES	SUCCESS	FAILURE	SUCCESS RATE
A	Manual	31	20	11	0.645161
E	Sensed	26	26	0	1

The table lists the items (components or systems) which have redundancies. The type of redundancy, manual or sensed, is shown in the next column. The number of times the switch occurred, and how many of these succeeded or failed, is shown, together with a success rate. Note that the "failure" number should match the

number in brackets for the redundant item in either of the component or system failure information tables.

A.5.2.11 Inspections Information

Each item which undergoes inspections will have the details of these given. The data shown is the component or system name, the number of times it was inspected, and how many item failures were found during each inspection. The rate of replacement with respect to number of inspections is then shown in the next column. The table below demonstrates this.

ITEM NAME	NUMBER OF INSPECTIONS	FAILURES FOUND	REPLACEMENT RATE
F	86	86	1
G3001	115	115	1

The data on item inspections is the last of the sets of results that the program exports.

A.6 Summary

A program has been created for the task of simulating a fleet of platforms undergoing phased MFOPs. This applies the Petri net models explained in Chapter 5 and Chapter 6, using inputs, in the form of text files, containing phase fault trees, component failure data, MFOP enabler information, and mission configurations. Using these, it creates arrays of objects, which can store information on each item such as a place, transition, phase, component or system.

The Petri nets it creates are simulated by randomly assigning switching times to all enabled timed transitions. These are switched in the order of these times, with all subsequently enabled immediate transitions being switched in between. By doing this over a satisfactory number of simulations, the expected performance of each platform, MFOP and mission can be estimated. When compared with the results from different system and enabler configurations, conclusions may be drawn over

the approach which appears to be most apt. This data has the potential to allow a great deal of improvement and demonstration of quality of systems, and could thus become a very valuable part of the design process.

Appendix B Data Input Files

This section contains the data files inputted to the simulation program. These were used to obtain the results given in Chapter 7. No explanation is given, as these fault trees are based on restricted material. It is felt that the short length of event names should be sufficiently informative without compromising sensitive data.

There are three generations of files. The second and third generations have minor differences from the first, and so the sections detailing these files will only highlight the differences from the previous set. The file “nimmish.txt”, shown in the first generation, did not undergo any changes.

B.1 First Generation

B.1.1 Fault Tree, Component Data and Discrete Events file

```
PHASES        19
PHASE         FlightCheck1       51

AIRCRAFT1     +  2  1.3-STRCT1   1.4-ON_GND
1.3-STRCT1    +  2  1.3.1STRC1   1.3.2FIRE1
1.4-ON_GND    +  1  G368
1.3.1STRC1    +  2  SD_GEN_SYS1  SD_LOW_ZHA
1.3.2FIRE1    +  2  EXPLOSION    FIRE_DAM1
EXPLOSION     +  1  G592-1
SD_GEN_SYS1   +  2  SD_ENG       SD_LGS1
FIRE_DAM1     +  2  FIRE_FLUID1  FIRE_OTHER1
SD_ENG        +  5  G155         G157 G164    G160    G415
FIRE_FLUID1   +  3  G694.        G714 G005-1
SD_LGS1       +  1  SD_LG1
SD_LG1        +  2  G449         G368
FIRE_OTHER1   +  3  G481         G009-1       G226
SD_LOW_ZHA    +  2  G660.        G641
G157          +  2  G157NONFBO   G637
G368          +  3  LG-LHGRSTRC  LG-RHGRSTRC  LG-ASSY1
G160          +  2  G160-G286    G160-G287
G009-1        +  1  G009-G316
G009-G316     +  3  G009FRSPGND  G009-G11     G009-TIT
G009-G11      *  2  G009-G15     G009-G12
G009-TIT      *  2  CF-APU-TIT   FP-FIRE14
G226          +  4  G226-G1      G226-G264    G226-G369      G226-G474
G592-1        +  2  G592-002-1   G592-003-1
G592-002-1    +  1  G592-004
G592-003-1    +  5  G592-006-1   G592-009     G592-083       FU-PMPFRXP
                                     FU-TMPSNSFR
G592-004      *  2  G592-005     CF-EXTFIRE
G592-005      +  8  FU-BURSTDR   FU-DRAINVLK  FU-XPOPPLK     FU-LBURST     FU-LFLAMEAF
                                     FU-RBURST      FU-RFLAMEAF    FU-RBURST
G592-006-1    +  1  FU-T4AWIRE
G592-009      *  2  FU-GA_NIS    FU-GA_OC
G592-083      +  3  G592-011     G592-012     G592-013
G415          +  4  PP-415-ENG1  PP-415-ENG2  PP-415-ENG3    PP-415-ENG4
G005-1        *  3  G005LEAK1    CF-OX-IGN1   CF-OX-COMB1
G005LEAK1     +  2  G005BRA1     G005OBGRATM
G005BRA1      +  1  G005BLKEGRD
G005BLKEGRD   *  2  G005-ECSGRD  G005-BOSGRD
G226-G1       +  3  G226-G11     G226-G167    G226-G700
G226-G11      *  2  G226DRILL11  G226-EF1-1
G226-G167     *  2  G226DRILL21  G226-EF2-1
```

```
G226-G264    +  3  G226-G265    G226-G284   G226-G838
G226-G265    *  2  G226DRILL12  G226-EF1-2
G226-G284    *  2  G226DRILL22  G226-EF2-2
G226-G369    +  3  G226-G370    G226-G389   G226-G921
G226-G370    *  2  G226DRILL13  G226-EF1-3
G226-G389    *  2  G226DRILL23  G226-EF2-3
G226-G474    +  3  G226-G475    G226-G494   G226-G999
G226-G475    *  2  G226DRILL14  G226-EF1-4
G226-G494    *  2  G226DRILL24  G226-EF2-4
G226-G700    *  2  G226-HF1-1   CF-FP-LK-1
G226-G838    *  2  G226-HF1-2   CF-FP-LK-2
G226-G921    *  2  G226-HF1-3   CF-FP-LK-3
G226-G999    *  2  G226-HF1-4   CF-FP-LK-4

ABANDONS     144

ABANDONS1    +  6  DASS         FMS  MAD    RCS    TCS    STORES
DASS         +  3  DASSDET      DASSEVAL    DASSPROT
DASSDET      +  2  RADWARNR     MISWARNR
DASSEVAL     +  1  DSM
DASSPROT     +  2  TRDS         CFD
RADWARNR     +  3  APR          SUPERHET    SIGNALREC
MISWARNR     +  2  ECU          MISSENS
TRDS         +  3  DEC          TQG  DASSLNCH
CFD          +  3  SDU          DCU  CFDF
SUPERHET     +  2  SUC          SUR
SIGNALREC    +  3  CDBAND       MIDHIBAND   SIGCONV
CDBAND       +  2  CBR          CBA
MIDHIBAND    +  2  DRR          DASSANT
SIGCONV      *  2  DC1          DC2
DASSANT      +  2  DASSFANT     DASSRANT
DASSFANT     *  2  HA1          HA2
DASSRANT     *  2  HA3          HA4
MISSENS      +  4  STP          STB  MISLSENS       MISRSENS
MISLSENS     *  2  SLN          SLB
MISRSENS     *  2  SRN          SRB
DASSLNCH     +  2  LRA          PLC
CFDF        / 6 12 DR1          DR2  DR3    DR4    DR5    DR6    DR7    DR8
                                     DR9    DR10   DR11   DR12
DR1          +  4  FMG1         CMG1 TOP1   BOT1
DR2          +  4  FMG2         CMG2 TOP2   BOT2
DR3          +  4  FMG3         CMG3 TOP3   BOT3
DR4          +  4  FMG4         CMG4 TOP4   BOT4
DR5          +  4  FMG5         CMG5 TOP5   BOT5
DR6          +  4  FMG6         CMG6 TOP6   BOT6
DR7          +  4  FMG7         CMG7 TOP7   BOT7
DR8          +  4  FMG8         CMG8 TOP8   BOT8
DR9          +  4  FMG9         CMG9 TOP9   BOT9
DR10         +  4  FMG10        CMG10TOP10  BOT10
DR11         +  4  FMG11        CMG11TOP11  BOT11
DR12         +  4  FMG12        CMG12TOP12  BOT12
```

```
FMS          +  3  FMCFAIL      MCDU USMS
FMCFAIL      +  2  EGI          FMC
MCDU         *  2  MDCU1        MDCU2
EGI          *  2  EGI1         EGI2
FMC          *  2  FMC1         FMC2
EGI1         +  4  EGI1INPUTS   EGI1UNIT     EGI1POW EGI1TRAY
EGI2         +  4  EGI2INPUTS   EGI2UNIT     EGI2POW EGI2TRAY
EGI1INPUTS   *  2  EGI1GPS      EGI1ANT
EGI2INPUTS   *  2  EGI2ANT      EGI2GPS
USMS         *  4  USMS1        USMS2USMS3   USMS4
USMS1        +  2  USMS1COM     USMS1CB
USMS2        +  2  USMS2COM     USMS2CB
USMS3        +  2  USMS3COM     USMS3CB
USMS4        +  2  USMS4COM     USMS4CB
MAD          +  5  MADCOMPAMP   DETECTHEAD   VECTMAGNET     MADBASE MADCB
RCS          +  2  TDI          COMM
TDI          *  3  TSN          TCSOSSATCOM
COMM         *  2  VOICECOM     TEXTCOM
TSN          *  2  L16          L11
TCSOS        +  3  IOPS         TCSOSMD      TCSOSSCSI
VOICECOM     *  2  RADIOCHAN    ADDITIONAL
TEXTCOM      +  2  TCSOS        TTYMODEM
TTYMODEM     +  3  TTYMOD       TTYKEYGEN    TTYPOWSW
RADIOCHAN    +  2  RADIOF       RADCTRLF
ADDITIONAL   *  2  L16          SATCOM
L16          +  5  DLS          JTT  ANTSYS3 L16SWI  L16BUS
L11          +  6  L11DTS       L11DLP       L11CRYPT       L11PWRSW       L11CB
                                     L11RADIO
SATCOM       +  3  INTERFACE    INTELEC      SIGCOMM
RADIOF       *  2  HF           V/UHF
RADCTRLF     *  2  TCSOS        RMU
RMU          *  2  PILOTRMU     COPILOTRMU
HF           *  2  HF1          HF2
V/UHF        *  2  "V/UHF1,2"   V/UHF3-5
INTERFACE    +  3  MOUNTS       PSW  PNL
INTELEC      +  4  MODEM        RFC  PSU     CTRLUNIT
SIGCOMM      +  3  KEY          DSC  ANTENNAE
MOUNTS       *  2  MT1          MT2
MODEM        +  4  RPA          BAS  MCD     MPS
CTRLUNIT     +  3  SCI          AUI  PRO
ANTENNAE     *  2  ANTSYS1      ANTSYS2
ANTSYS1      +  2  ANT1         MCM1
ANTSYS2      +  2  ANT2         MCM2
DLS          +  2  RAD          BAT
ANTSYS3      *  2  ANTTOP       ANTBOT
L16SWI       +  2  SWA          PWR
L16BUS       *  2  L16BUSA      L16BUSB
ANTTOP       +  2  ATT          RAT
ANTBOT       +  2  ATB          RAB
L16BUSA      +  3  BAC          CBA  TBA
L16BUSB      +  3  BBC          CBB  TBB
```

```
"V/UHF1,2"    *  2  V/UHF1       V/UHF2
V/UHF3-5      *  3  V/UHF3       V/UHF4       V/UHF5
V/UHF1        +  5  V/U1XCVR     V/U1FILT     V/U1ANT V/U1DTOA       V/U1LOGU
V/UHF2        +  5  V/U2XCVR     V/U2FILT     V/U2ANT V/U2DTOA       V/U2LOGU
V/UHF3        +  5  V/U3XCVR     V/U3FILT     V/U3ANT V/U3DTOA       V/U3LOGU
V/UHF4        +  5  V/U4XCVR     V/U4FILT     V/U4ANT V/U4DTOA       V/U4LOGU
V/UHF5        +  5  V/U5XCVR     V/U5FILT     V/U5ANT V/U5DTOA       V/U5LOGU
V/U2FILT      *  2  V/U2VFILT    V/U2UFILT
HF1           +  6  HF1RECEXC    HF1POWAMP    HF1ANTTUN      HF1PPSFIL      HF1DTOA
                                      HF1SHUNT
HF2           +  8  HF2RECEXC    HF2POWAMP    HF2ANTTUN      HF2PPSFIL      HF2DTOA
                                      HF2SHUNT       HF2POWSW       GHF2ANT
GHF2ANT       +  5  HF2FWDANT    HF2CNTANT    HF2AFTANT      HF2ANT  HF2GNDANT
L11RADIO      *  4  HF1          HF2  V/UHF4  V/UHF5
PILOTRMU      +  2  FCRA1        RMUIF1
COPILOTRMU    +  2  FCRA2        RMUIF2
FCRA1         *  2  NRCTCS       FI1
RMUIF1        +  4  PSU1         MC1  FLX1    CA1
FCRA2         *  2  B3I2         FI2
RMUIF2        +  4  PSU2         MC2  FLX2    CA2
NRCTCS        +  2  B3I1         TCSOS
TCS           +  3  TCSPROC      TCSOPTCSREC
TCSPROC       +  2  IOPIFU       TCSOS
TCSOP         *  4  NORM5WKSTN   PILOTWKSTN   ACO2WKSTN      ORDN2WKSTN
TCSREC        *  6  VIDINTU      HDDR MAGDISK1       MAGDISK2       CBS     TXTPRINT
IOPS          / 2   3            IOP1 IOP2    TAC1PWRSW
NORM5WKSTN    *  5  WKSTN1       WKSTN2       WKSTN3  WKSTN4  WKSTN5
PILOTWKSTN    +  4  PILOTTCP     PILOTDP      PILOTTCPIFU    PILOTCHRD
ACO2WKSTN     *  2  ACOWKSTN1    ACOWKSTN2
ORDN2WKSTN    *  2  ORDWKSTN1    ORDWKSTN2
ORDWKSTN1     +  3  ORD1PEP      ORD1PSU      ORD1SONRS
ORDWKSTN2     +  3  ORD2PEP      ORD2PSU      ORD2SONRS
ACOWKSTN1     +  4  ACO1PEPS     ACO1PSU      ACO1CHRDS      ACO1INPUTS
ACOWKSTN2     +  4  ACO2PEPS     ACO2PSU      ACO2CHRDS      ACO2INPUTS
ACO1PEPS      *  2  ACO1PEP1     ACO1PEP2
ACO1CHRDS     *  2  ACO1CHRD     SPARECHRD
ACO1INPUTS    *  4  ACO1KEYP     ACO1KEYB     ACO1KEYPL      ACO1ROLBAL
ACO2PEPS      *  2  ACO2PEP1     ACO2PEP2
ACO2CHRDS     *  2  ACO2CHRD     SPARECHRD
ACO2INPUTS    *  4  ACO2KEYP     ACO2KEYB     ACO2KEYPL      ACO2ROLBAL
WKSTN1        +  5  WK1PEP       WK1DPWK1PSU  WK1CHRD WK1INPUTS
WKSTN2        +  5  WK2PEP       WK2DPWK2PSU  WK2CHRD WK2INPUTS
WKSTN3        +  5  WK3PEP       WK3DPWK3PSU  WK3CHRD WK3INPUTS
WKSTN4        +  5  WK4PEP       WK4DPWK4PSU  WK4CHRD WK4INPUTS
WKSTN5        +  5  WK5PEP       WK5DPWK5PSU  WK5CHRD WK5INPUTS
WK1PEP        *  2  WK1PEP1      WK1PEP2
WK1INPUTS     *  4  WK1KEYP      WK1KEYB      WK1KEYPL       WK1ROLBAL
WK2PEP        *  2  WK2PEP1      WK2PEP2
WK2INPUTS     *  4  WK2KEYP      WK2KEYB      WK2KEYPL       WK2ROLBAL
WK3PEP        *  2  WK3PEP1      WK3PEP2
WK3INPUTS     *  4  WK3KEYP      WK3KEYB      WK3KEYPL       WK3ROLBAL
```

```
WK4PEP        *  2  WK4PEP1      WK4PEP2
WK4INPUTS     *  4  WK4KEYP      WK4KEYB      WK4KEYPL       WK4ROLBAL
WK5PEP        *  2  WK5PEP1      WK5PEP2
WK5INPUTS     *  4  WK5KEYP      WK5KEYB      WK5KEYPL       WK5ROLBAL
STORES        +  2  SMGMTSYS     SONORELSYS
SMGMTSYS      +  4  SMGMTPROC    STNCTRLU     BBAYDR  WEAPRELSW
SONORELSYS    *  2  SINGLAUNCH   TENSHOT
STNCTRLU      *  5  STNCTRLU1    STNCTRLU2    STNCTRLU3      STNCTRLU4      STNCTRLU5
WEAPRELSW     *  4  WRELSW1      WRELSW2      WRELSW3 WRELSW4
SINGLAUNCH    *  2  SINGSHOT1    SINGSHOT2
TENSHOT       *  4  TENSHOT1     TENSHOT2     TENSHOT3       TENSHOT4

PHASE         FlightCheck2       1
AIRCRAFT1A    +  2  1.3-STRCT1   1.4-ON_GND

ABANDONS      33
ABANDONS1A    +  7  DASS         ESM  FMS     RADAR-IFF      RCS     TCS     STORES
ESM           +  2  GSIGPROC     INTTCS
GSIGPROC      +  2  SIGDEL       ESMPRO
INTTCS        *  2  MSU          1553BUS
SIGDEL        +  2  SPINCHAN     MAINCHAN
SPINCHAN      +  2  PROCFAIL     SAU
MAINCHAN      +  2  ANTCLUST     MRX
ANTCLUST      *  4  ACLUST1      ACLUST2      ACLUST3 ACLUST4
PROCFAIL      +  2  SRX          SFE
ACLUST1       *  3  FR1          LBA1 HBA1
ACLUST2       *  3  FR2          LBA2 HBA2
ACLUST3       *  3  FR3          LBA3 HBA3
ACLUST4       / 2 3 FR4          LBA4 HBA4
LBA1          +  2  LSPIANT1     LB1
HBA1          +  2  HSPIANT1     HB1
LBA2          +  2  LSPIANT2     LB2
HBA2          +  2  HSPIANT2     HB2
LBA3          +  2  LSPIANT3     LB3
HBA3          +  2  HSPIANT3     HB3
LBA4          +  2  LSPIANT4     LB4
HBA4          +  2  HSPIANT4     HB4
LSPIANT1      *  2  LS1-1        LS1-2
HSPIANT1      *  2  HS1-1        HS1-2
LSPIANT2      *  2  LS2-1        LS2-2
HSPIANT2      *  2  HS2-1        HS2-2
LSPIANT3      *  2  LS3-1        LS3-2
HSPIANT3      *  2  HS3-1        HS3-2
LSPIANT4      *  2  LS4-1        LS4-2
HSPIANT4      *  2  HS4-1        HS4-2
RADAR-IFF     +  10 DATAPROC     SIGPROC      IFFINTGTR      INS-GPS REC-EXC RFTRANS
                                      RFCTRL  RFSCAN  RADCOOLING     1553BBUS
IFFINTGTR     +  3  IFFUNIT      CRYPTOFILL   IFFCB
RADCOOLING    *  2  DRADTDCR     PRADTDCR
1553BBUS      *  2  1553BUS1     1553BUS2
```

```
PHASE         FlightCheck3       1
AIRCRAFT1B    +  2  1.3-STRCT1   1.4-ON_GND

ABANDONS      3
ABANDONS1B    +  6  DASS         FMS  RADAR-IFF      RCS     TCS     EOSDS
EOSDS         +  7  TUR1         SCU1 PDU1    HGP1   TLU1    PWP1    EOSDSCP
EOSDSCP       +  2  DPS1         RTS1

PHASE         EngStart           13

AIRCRAFT2     +  2  1.3-STRCT2   1.4-ON_GND
1.3-STRCT2    +  2  1.3.1STRC2   1.3.2FIRE2
1.3.1STRC2    +  2  SD_GEN_SYS2  SD_LOW_ZHA
1.3.2FIRE2    +  2  EXPLOSION2   FIRE_DAM1
EXPLOSION2    +  1  G592-2
SD_GEN_SYS2   +  2  SD_ENG       SD_LGS2
SD_LGS2       +  2  SD_LG1       SD_WBT1
SD_WBT1       +  1  G495.
G592-2        +  2  G592-002-1   G592-003-2
G592-003-2    +  6  G592-006-2   G592-007    G592-009       G592-083
                                     FU-PMPFRXP     FU-TMPSNSFR
G592-006-2    *  2  GATE11-1     FU-T4AWIRE
G592-007      *  2  CFSTOREFIRE  FUINADVJETT
GATE11-1      +  1  CFFUEAIRMIX

ABANDONS      1
ABANDONS2     +  1  RCS

PHASE         Take-Off           58

AIRCRAFT3     +  3  1.1-DEV_FP1  1.3-STRCT3  1.4-ON_GND
1.1-DEV_FP1   +  2  1.1.1-AERO1  1.1.2-TRST1
1.3-STRCT3    +  2  1.3.1STRC3   1.3.2FIRE3
1.1.1-AERO1   +  3  LAC_FCS      LAC-ENV1    LAC-PILOT
1.1.2-TRST1   +  2  TRST_FUEL1   G554-1
1.3.1STRC3    +  4  SD_GEN_SYS3  G647-1      SD_LOW_ZHA     SD-FCS
1.3.2FIRE3    +  2  EXPLOSION3   FIRE_DAM3
LAC_FCS       +  7  G394         G568 G569   G570   G696.  G698.  G697.
G554-1        +  4  G174-1       G398-1      G659-1 G173-1
TRST_FUEL1    +  1  G166
EXPLOSION3    +  1  G592-3
SD_GEN_SYS3   +  3  SD_ENG       SD_LGS3     SD_AIR1
SD_AIR1       +  1  G274-1
FIRE_DAM3     +  2  FIRE_FLUID3  FIRE_OTHER3
FIRE_FLUID3   +  3  G694.        G714 G005-3
LAC-ENV1      +  2  G_430_LM     G658
LAC-PILOT     +  2  G312         G635
SD_LGS3       +  2  SD_LG1       SD_WBT2
G394          +  3  LAC_FLAPS    G265 G309
G568          +  3  G670         G671 G672
G569          +  3  G654         G673 G674
```

```
LAC_FLAPS     +  1  G134
G_430_LM      +  1  G430
G570          +  3  G675          G676 G677
G647-1        +  1  SD_OTH_DAM1
SD_OTH_DAM1   +  1  SD_CNTL1
SD_CNTL1      +  3  BBD_OPEN      SD_FLAP1     G683.
BBD_OPEN      +  1  G644
SD_FLAP1      *  2  G265          CFFLAP>ENV.
SD_WBT2       +  2  G396-1        G495.
G396-1        +  3  G629          G695.G530
FIRE_OTHER3   +  3  G481          G009-2       G226
SD-FCS        +  2  G715          GATE12
G274-1        +  2  G274NMVTUPR   G274VTPRNM
G166          +  3  G166-2007AX   G166-2007BX  G166-2007CX
G174-1        +  1  G554TL3ENTR
G398-1        +  4  G554-ADC      G554-CREW8   G554-INDERR    G554-TR-INC
G659-1        +  1  G554-G4821
G173-1        *  2  G554-TL-2EN   CF-G554-C4
G009-2        +  1  G009-FRFLT1
G009-FRFLT1   *  2  G009-G417     G009-TAR1
G009-G417     +  3  G009FRSPFLT   G009-G320    FP-FIRE14
G009-TAR1     +  2  G009-G307     CF-APUOPFLT
G592-3        +  1  G592-003-3
G592-003-3    +  5  G592-006-3    G592-009     G592-083       FU-PMPFRXP
                                       FU-TMPSNSFR
G592-006-3    *  2  GATE11-2      FU-T4AWIRE
GATE11-2      +  2  CFFUEAIRMIX   CF-NEGGMAN
G658          +  2  G658-G100     G658-G110
G635          *  2  HEPEVNAV      NVVNAVER
GATE12        +  2  G312E_AIRSP   G312L_AIRSP
G005-3        *  3  G005-LEAK3    CF-OX-IGN3   CF-OX-COMB3
G005-LEAK3    +  3  G005BRA1      G005OBGRATM  G005PRATMOS
G005PRATMOS   *  2  G005INSFPFL   G005-PBTLFL
G005-PBTLFL   +  3  G005PBTL1FL   G005PBTL2FL  G005PBTL3FL
G005PBTL1FL   +  2  OX-PBTL1LK    OX-PBTL1HED
G005PBTL2FL   +  2  OX-PBTL2LK    OX-PBTL2HED
G005PBTL3FL   +  2  OX-PBTL3LK    OX-PBTL3HED
G430          +  2  G230          G275

ABANDONS      1
ABANDONS3     +  3  RCS           FMS  TCS

PHASE         Climb 64
AIRCRAFT4     +  3  1.1-DEV_FP2   1.2-CFIT1    1.3-STRCT4
1.1-DEV_FP2   +  2  1.1.1-AERO2   1.1.2-TRST2
1.2-CFIT1     +  5  AP_CFITMSA1   CFIT_DISPL   CFIT_SR_NAV    CFIT_PIL1      G656.
1.3-STRCT4    +  2  1.3.1STRC4    1.3.2FIRE4
1.1.1-AERO2   +  3  LAC_FCS       LAC-ENV2     LAC-PILOT
1.1.2-TRST2   +  2  TRST_FUEL2    G554-2
1.3.1STRC4    +  4  SD_GEN_SYS4   G647-1       SD_LOW_ZHA     SD-FCS
1.3.2FIRE4    +  2  EXPLOSION4    FIRE_DAM4
```

```
G554-2        +  4  G174-1        G398-2       G659-2  G173-1
TRST_FUEL2    +  2  L/VNAV_ERR    G166
EXPLOSION4    +  1  G592-4
SD_GEN_SYS4   +  2  SD_ENG        SD_AIR2
SD_AIR2       +  4  G576.         G274-1       G575    G010
FIRE_DAM4     +  2  FIRE_FLUID4   FIRE_OTHER4
FIRE_FLUID4   +  3  G694.         G714 G005-4
AP_CFITMSA1   +  1  G425-1
LAC-ENV2      +  2  G_430_LM      G364-1
CFIT_DISPL    +  1  G312
CFIT_SR_NAV   +  3  G442/443      G635 G625
CFIT_PIL1     +  2  G106          G109-1
FIRE_OTHER4   +  4  G481          G010 G009-2  G226
L/VNAV_ERR    +  2  G625          G635
G575          *  2  G575ECPVNYC   G575ECSVNYC
G398-2        +  3  G554-ADC      G554-CREW8   G554-TR-INC
G659-2        +  2  G554-G4821    G554-G4822
G010          +  2  G010FISLKAG   G010FISLKHP
G592-4        +  1  G592-003-4
G592-003-4    +  6  G592-006-3    G592-008-1   G592-009       G592-083
                                       FU-PMPFRXP     FU-TMPSNSFR
G592-008-1    *  4  CFFLJETTIS    CFSTOREFIRE  FUFLPROCEA     FUFLPROCEB
G364-1        *  1  G364-SVRACC
G442/443      *  2  NVEPITDISP    NVECPDISP
NVEPITDISP    +  3  NVMMRSYSERR   NVEPFD1PROC  NVPILOTPFD1
NVPILOTPFD1   *  2  NVEPFD1DISP   NVEND1DISP
NVEPFD1PROC   +  1  CDEDMC1SYS
NVEPFD1DISP   +  2  CDERRPFD1HW   CD_SW_DU2
NVMMRSYSERR   *  2  NVM1SYSERR    NVM2SYSERR
NVM1SYSERR    +  4  NVERRMMR1HW   NV_SW_MMR    ECSLDCCRTUN    ECSBDCCRUN
NVM2SYSERR    +  2  NVERRMMR2HW   NV_SW_MMR
NVECPDISP     +  3  NVMMRSYSERR   NVEPFD2PROC  NVCPND1
NVCPND1       +  1  NVEPFD2DISP
NVEPFD2PROC   +  1  NVEDMC2SYS
NVEPFD2DISP   +  2  CDERRPFD2HW   CD_SW_DU3
NVEDMC2SYS    +  2  CDD2EFISERR   CDDMC2ECOOL
CDEDMC1SYS    +  2  CDD1EFISERR   NVDMC1ECOOL
NVDMC1ECOOL   *  2  CD_DMC_COOL   SFCDLC
CDDMC2ECOOL   *  2  CD_DMC_COOL   SFCDLC
NVEND1DISP    +  2  CDERRND1HW1   CD_SW_DU2
G109-1        +  4  G109INAR5DC   G109IAR5DEC  G109-PORTFL    G109-ISMK
G005-4        *  3  G005-LEAK4    CF-OX-IGN4   CF-OX-COMB4
G005-LEAK4    +  3  G005-BRA2     G005OBGRATM  G005PRATMOS
G005-BRA2     +  1  G005BLKEFLT
G005BLKEFLT   *  2  G005ECSFFLT   G005BLKFLT
G109IAR5DEC   *  3  OXRDCMPCOND   CFOXAR5COND  G109-GL-AR5
G109INAR5DC   *  2  G109-GLNAR5   OXRDCMPCOND
G109-PORTFL   +  2  G109PFLNAR5   G109PFLAR5
G109-ISMK     *  2  OX-SMK-COND   G109-GL-SMK
G109-GL-AR5   +  2  G109GLAR5S    OXPLTDCFAIL
G425-1        +  6  G593          G594 G596    G597    G598-1  G599-1
```

```
G594          +  5  G594-ALT     G594-APPR   G594-LVLCHG    G594-VNAV      G594-VS
G597          +  2  G597A        G597B
G593          +  2  G593B        G593C
G599-1        +  2  G425APSENG1  G425APSENG2
G598-1        +  1  G425-SMJAM
G596          +  2  G425-TRMLOS  G425-TRMRUN

ABANDONS      1
ABANDONS4     +  4  DASS         FMS  RCS    TCS

PHASE         Transit            23
AIRCRAFT5     +  3  1.1-DEV_FP3  1.2-CFIT2   1.3-STRCT5
1.1-DEV_FP3   +  2  1.1.1-AERO2  1.1.2-TRST3
1.2-CFIT2     +  5  AP_CFITMSA2  CFIT_DISPL  CFIT_SR_NAV    CFIT_PIL2      G656.
1.3-STRCT5    +  2  1.3.1STRC5   1.3.2FIRE5
1.1.2-TRST3   +  3  TRST_FUEL2   G554-3      G452
1.3.1STRC5    +  4  SD_GEN_SYS5  G647-1      SD_LOW_ZHA     SD-FCS
1.3.2FIRE5    +  2  EXPLOSION4   FIRE_DAM5
G554-3        +  3  G174-1       G398-2      G659-2
SD_GEN_SYS5   +  2  SD_ENG       SD_AIR3
SD_AIR3       +  4  G576.        G274-2      G575   G010
FIRE_DAM5     +  2  FIRE_FLUID5  FIRE_OTHER5
FIRE_FLUID5   +  4  G565.        G694.G714   G005-4
AP_CFITMSA2   +  1  G425-2
CFIT_PIL2     +  3  G106         G580-1      G109-1
FIRE_OTHER5   +  4  G481         G010 G009-3 G226
G274-2        +  3  G274-G100    G274NMVTUPR G274VTPRNM
G009-3        +  1  G009-FRFLT2
G009-FRFLT2   +  1  G009-G417
G580-1        *  3  G580CDPR15K  G580NALCBHI ECCALT15KFT
G580CDPR15K   +  2  G580DPR15KL  G580DPR15KA
G580NALCBHI   +  2  G580CHRWF    SFCDLW
G580CHRWF     *  2  G580HIRXPRI  G580HIRXSEC
G425-2        +  5  G593         G594        G596   G597   G598-1

ABANDONS      1
ABANDONS5     +  4  DASS         FMS  RCS    TCS

PHASE         Descent1           12
AIRCRAFT6     +  3  1.1-DEV_FP4  1.2-CFIT3   1.3-STRCT6
1.1-DEV_FP4   +  2  1.1.1-AERO2  1.1.2-TRST4
1.2-CFIT3     +  5  AP_CFITMSA3  CFIT_DISPL  CFIT_SR_NAV    CFIT_PIL1      G656.
1.3-STRCT6    +  2  1.3.1STRC4   1.3.2FIRE6
1.1.2-TRST4   +  3  TRST_FUEL2   G554-4      G452
1.3.2FIRE6    +  2  EXPLOSION3   FIRE_DAM6
G554-4        +  3  G398-2       G659-2      G173-2
FIRE_DAM6     +  2  FIRE_FLUID4  FIRE_OTHER5
AP_CFITMSA3   +  1  G425-3
G173-2        +  1  G554-TL-2EN
G425-3        +  5  G593         G594 G596   G597   G598-2
G598-2        +  2  G425-SMJAM   G598APLDHAZ
```

```
ABANDONS      1
ABANDONS6     +  4  DASS          FMS  RCS     TCS

PHASE         On-Task1            26
AIRCRAFT7     +  3  1.1-DEV_FP5   1.2-CFIT4    1.3-STRCT7
1.1-DEV_FP5   +  2  1.1.1-AERO1   1.1.2-TRST5
1.2-CFIT4     +  5  AP_CFITMSA4   CFIT_DISPL   CFIT_SR_NAV    CFIT_PIL1      G656.
1.3-STRCT7    +  2  1.3.1STRC6    1.3.2FIRE7
1.1.2-TRST5   +  3  TRST_FUEL2    G554-5       G452
1.3.1STRC6    +  4  SD_GEN_SYS4   G647-2       SD_LOW_ZHA     SD-FCS
1.3.2FIRE7    +  2  EXPLOSION5    FIRE_DAM7
G554-5        +  4  G174-2        G398-2       G659-2  G173-2
EXPLOSION5    +  2  EXP_WEP       G592-5
FIRE_DAM7     +  2  FIRE_FLUID4   FIRE_OTHER6
EXP_WEP       +  2  G617.         G607.
AP_CFITMSA4   +  1  G425-4
G647-2        +  2  SDSTOREDAM1   SD_OTH_DAM1
SDSTOREDAM1   +  1  SD_WEP1
SD_WEP1       +  2  SD_WEPS1      SD_NON_WEP
SD_WEPS1      +  3  SD_WEP_PL1    G023         G618.
SD_WEP_PL1    +  1  WEP_LAUNCH
WEP_LAUNCH    +  4  G582.         G583.        G607.   G617.
FIRE_OTHER6   +  5  G621          G481         G010    G009-3  G226
SD_NON_WEP    +  2  G024          G071
G174-2        +  1  G554TL3ENTA
G592-5        +  1  G592-003-5
G592-003-5    +  6  G592-006-3    G592-007     G592-009       G592-083
                                       FU-PMPFRXP     FU-TMPSNSFR
G425-4        +  6  G593          G594 G596    G597    G598-3  G599-2
G599-2        +  1  G425APSENG3
G598-3        +  2  G425-SMJAM    G598APOTHAZ

ABANDONS      1
ABANDONS7     +  5  DASS          FMS  RCS     TCS     MAD

PHASE         On-Task2            1
AIRCRAFT7A    +  3  1.1-DEV_FP5   1.2-CFIT4    1.3-STRCT7

ABANDONS      1
ABANDONS7A    +  6  DASS          ESM  FMS     RADAR-IFF      RCS     TCS

PHASE         On-Task3            1
AIRCRAFT7B    +  3  1.1-DEV_FP5   1.2-CFIT4    1.3-STRCT7

ABANDONS      1
ABANDONS7B    +  6  DASS          FMS  RADAR-IFF      RCS     TCS     EOSDS

PHASE         Attack1             19
AIRCRAFT8     +  3  1.1-DEV_FP6   1.2-CFIT4    1.3-STRCT8
1.1-DEV_FP6   +  2  1.1.1-AERO1   1.1.2-TRST6
1.3-STRCT8    +  2  1.3.1STRC7    1.3.2FIRE8
```

```
1.1.2-TRST6   +  3  TRST_FUEL2   G554-6       G452
1.3.1STRC7    +  5  SD_GEN_SYS4  G647-3       SDHIGHERZHA    SD_LOW_ZHA     SD-FCS
1.3.2FIRE8    +  2  EXPLOSION6   FIRE_DAM7
G554-6        +  3  G174-2       G398-2       G659-2
EXPLOSION6    +  2  EXP_WEP      G592-6
G647-3        +  3  SDSTOREDAM2  SDOTHERDAM2  SD_JETT1
SD_JETT1      *  2  G070         CFJTRELENV.
SDSTOREDAM2   +  1  SD_WEP2
SD_WEP2       +  2  SD_WEPS2     SD_NON_WEP
SDOTHERDAM2   +  1  SD_CNTL2
SD_CNTL2      +  2  BBD_OPEN     G683.
SD_WEPS2      +  3  SD_WEP_PL2   G023 G618.
SD_WEP_PL2    *  2  WEP_LAUNCH   CFWEPLNCH.
SDHIGHERZHA   +  2  G702         G707
G592-6        +  1  G592-003-6
G592-003-6    +  7  G592-006-3   G592-007     G592-008-1     G592-009       G592-083
                                      FU-PMPFRXP     FU-TMPSNSFR

ABANDONS      1
ABANDONS8     +  6  DASS         FMS  RCS     TCS     MAD     STORES

PHASE         Attack2            1
AIRCRAFT8A    +  3  1.1-DEV_FP6  1.2-CFIT4    1.3-STRCT8


ABANDONS      1
ABANDONS8A    +  7  DASS         ESM  FMS     RADAR-IFF      RCS     TCS     STORES

PHASE         Off-Task           2
AIRCRAFT9     +  3  1.1-DEV_FP4  1.2-CFIT5    1.3-STRCT6
1.2-CFIT5     +  5  AP_CFITMSA2  CFIT_DISPL   CFIT_SR_NAV    CFIT_PIL1      G656.

ABANDONS      1
ABANDONS9     +  4  DASS         FMS  RCS     TCS

PHASE         Return             25
AIRCRAFT10    +  3  1.1-DEV_FP7  1.2-CFIT2    1.3-STRCT9
1.1-DEV_FP7   +  2  1.1.1-AERO2  1.1.2-TRST7
1.3-STRCT9    +  2  1.3.1STRC8   1.3.2FIRE5
1.1.2-TRST7   +  3  TRST_FUEL2   G554-7       G452
1.3.1STRC8    +  5  SD_GEN_SYS5  SD_COLL      G647-1  SD_LOW_ZHA     SD-FCS
G554-7        +  3  G174-1       G398-1       G659-2
SD_COLL       +  2  G642         G653COLLDAY
G642          +  2  G642-G176    G642-G177
G642-G176     *  3  SF568UNCP/U  CFPPG642RFL  AWCREWE153
G642-G177     +  2  G642-GATE1   G642-G178
G653COLLDAY   *  3  G6533TOAVD   G653AVDFDAY  CFTMRSKD
G653AVDFDAY   *  3  G653COMFL    G653-SVFLD   AWCREWE125
G653COMFL     *  2  G653CMATCFL  G653CMBSFL
G6533TOAVD    +  2  G6531PLTFRL  G653RLSADHD
G653RLSADHD   +  1  CF3PTHRAD
```

```
G6531PLTFRL   *  2  G653-FLRULE   CCOLL3FLT
G653-FLRULE   +  3  AWCREWE126    AWCREWE127   G312
G653CMATCFL   +  3  CF-ATCFAILS   TCRADFL      AWCREWE174
G653CMBSFL    +  4  CFOPBOUTRNG   CFOPSBASFLS  TCRADFL AWCREWE175
AWCREWE125    *  2  HE3FDETDAY    AWCREW-ERR1
AWCREWE126    *  2  HEPTFVFRIFR   AWCREW-ERR1
AWCREWE127    *  2  HE3FVFRIFR    AWCREW-ERR1
AWCREWE153    *  2  HEPG642TCLS   AWCREW-ERR2
AWCREWE174    *  2  HEPLFATCINS   AWCREW-ERR1
AWCREWE175    *  2  HEPLTFLOPIN   AWCREW-ERR1

ABANDONS      1
ABANDONS10    +  4  DASS          FMS  RCS     TCS

PHASE         Descent2            18
AIRCRAFT11    +  3  1.1-DEV_FP4   1.2-CFIT6    1.3-STRCT10
1.2-CFIT6     +  5  AP_CFITMSA2   CFIT_DISPL   CFIT_SR_NAV    CFIT_PIL3      G656.
1.3-STRCT10   +  2  1.3.1STRC9    1.3.2FIRE9
1.3.1STRC9    +  4  SD_GEN_SYS6   G647-4       SD_LOW_ZHA     SD-FCS
1.3.2FIRE9    +  2  EXPLOSION7    FIRE_DAM6
EXPLOSION7    +  1  G592-7
SD_GEN_SYS6   +  2  SD_ENG        SD_AIR4
SD_AIR4       +  4  G576.         G274-3       G575    G010
G647-4        +  1  SDOTHERDAM3
SDOTHERDAM3   +  1  SD_CNTL3
SD_CNTL3      +  3  BBD_OPEN      SD_FLAP2     G683.
SD_FLAP2      +  1  G265
CFIT_PIL3     +  3  G106          G580-2       G109-1
G274-3        +  2  G274-G100     G274VTPRNM
G580-2        *  2  G580CDPR15K   G580NALCBHI
G592-7        +  1  G592-003-7
G592-003-7    +  7  G592-006-3    G592-007     G592-008-2     G592-009       G592-083
                                       FU-PMPFRXP     FU-TMPSNSFR
G592-008-2    *  3  CFSTOREFIRE   FUFLPROCEA   FUFLPROCEB

ABANDONS      1
ABANDONS11    +  3  FMS           RCS  TCS

PHASE         Approach            13
AIRCRAFT12    +  3  1.1-DEV_FP8   1.2-CFIT7    1.3-STRCT11
1.1-DEV_FP8   +  2  1.1.1-AERO3   1.1.2-TRST4
1.2-CFIT7     +  5  AP_CFITMSA5   CFIT_DISPL   CFIT_SR_NAV    CFIT_PIL1      G656.
1.3-STRCT11   +  2  1.3.1STRC10   1.3.2FIRE9
1.1.1-AERO3   +  3  LAC_FCS       LAC-ENV3     LAC-PILOT
1.3.1STRC10   +  4  SD_GEN_SYS4   G647-5       SD_LOW_ZHA     SD-FCS
AP_CFITMSA5   +  1  G425-5
G647-5        +  2  SDOTHERDAM3   SD_JETT2
SD_JETT2      +  1  G070
LAC-ENV3      +  2  G_430_LM      G364-2
G364-2        +  1  G364-SVRACC
G425-5        +  6  G593          G594 G596    G597    G598-2  G599-3
```

```
G599-3        +  1  G599-LAND

ABANDONS      1
ABANDONS12    +  3  FMS           RCS  TCS

PHASE         Land  26
AIRCRAFT13    +  4  1.1-DEV_FP9   1.2-CFIT8    1.3-STRCT12    1.4-ON_GND
1.1-DEV_FP9   +  2  1.1.1-AERO2   1.1.2-TRST8
1.2-CFIT8     +  5  AP_CFITMSA5   CFIT_DISPL   CFIT_SR_NAV    CFIT_PIL4      G656.
1.3-STRCT12   +  2  1.3.1STRC11   1.3.2FIRE10
1.1.2-TRST8   +  3  TRST_FUEL3    G554-8       G452
1.3.1STRC11   +  4  SD_GEN_SYS7   G647-4       SD_LOW_ZHA     SD-FCS
1.3.2FIRE10   +  2  EXPLOSION8    FIRE_DAM8
G554-8        +  2  G398-2        G659-2
TRST_FUEL3    +  3  L/VNAV_ERR    G069 G166
EXPLOSION8    +  1  G592-8
SD_GEN_SYS7   +  3  SD_ENG        SD_LGS4      SD_AIR2
FIRE_DAM8     +  2  FIRE_FLUID4   FIRE_OTHER7
SD_LGS4       +  2  SD_LG2        SD_WBT3
SD_WBT3       +  2  G396-2        G495.
G396-2        +  5  G629          G695.G058    G064    G530
CFIT_PIL4     +  2  G106          G109-2
SD_LG2        +  3  G589.         G449 G368
FIRE_OTHER7   +  4  G481          G010 G009-4  G226
G069          +  6  G069-IFBBD    G069INSFLFP  G069ARBENRQ    G069ISFFLAI
                                       G069LGENRQ     G069ISFFLAL
G069-IFBBD    +  1  G069INSFBBD
G009-4        +  2  G009-FRFLT2   G009-G316
G592-8        +  2  G592-002-2    G592-003-3
G592-002-2    *  2  G592-004      CF-ONGROUND
G069ISFFLAI   +  1  G069ARAENRQ
G109-2        +  3  G109INAR5DC   G109-PORTFL  G109-ISMK
G069ISFFLAL   *  3  OXRDCMPCOND   CFPLE10000   G069OBOGSLS

ABANDONS      1
ABANDONS13    +  3  FMS           RCS  TCS

PHASE         Taxi  17
AIRCRAFT14    +  2  1.3-STRCT13   1.4-ON_GND
1.3-STRCT13   +  2  1.3.1STRC12   1.3.2FIRE11
1.3.1STRC12   +  3  SD_GEN_SYS8   G647-6       SD_LOW_ZHA
1.3.2FIRE11   +  2  EXPLOSION9    FIRE_DAM9
EXPLOSION9    +  1  G592-9
SD_GEN_SYS8   +  2  SD_ENG        SD_LGS5
FIRE_DAM9     +  2  FIRE_FLUID1   FIRE_OTHER8
SD_LGS5       +  2  SD_LG1        SD_WBT4
SD_WBT4       +  2  G396-3        G495.
G396-3        +  2  G695.         G064
G647-6        +  1  SDOTHERDAM4
SDOTHERDAM4   +  1  SD_CNTL4
SD_CNTL4      +  2  SD_FLAP2      G683.
```

```
FIRE_OTHER8   +  3  G481         G009-5      G226
G009-5        +  2  G009-FRFLT1  G009-G316
G592-9        +  2  G592-002-1   G592-003-8
G592-003-8    +  4  G592-009     G592-083    FU-PMPFRXP     FU-TMPSNSFR

COMPONENTS

G164          U  EXP  0.000000283          G226-EF1-2   U  EXP  0.000001205
G155          U  EXP  0.000000001          G226DRILL22  U  EXP  0.0003177
G660.         U  EXP  9E-10                G226-EF2-2   U  EXP  1.056E-07
G637          U  EXP  0.00000053           G226-HF1-2   U  EXP  5.491E-09
G641          U  EXP  9E-10                G226DRILL13  U  EXP  0.0003139
G694.         U  EXP  9.9E-10              G226-EF1-3   U  EXP  0.000001205
G714          U  EXP  9.9E-10              G226DRILL23  U  EXP  0.0003176
G157NONFBO    U  EXP  1.43E-08             G226-EF2-3   U  EXP  1.056E-07
G481          U  EXP  0.000000075          G226-HF1-3   U  EXP  5.491E-09
LG-ASSY1      U  EXP  3.31E-08             G226DRILL14  U  EXP  0.0003139
FP-FIRE14     U  EXP  0.0000001            G226-EF1-4   U  EXP  0.000001208
CF-EXTFIRE    U  EXP  0.000001             G226DRILL24  U  EXP  0.0003164
FU-BURSTDR    U  EXP  0.00000404           G226-EF2-4   U  EXP  1.056E-07
FU-LBURST     U  EXP  0.000004             G226-HF1-4   U  EXP  5.491E-09
FU-DRAINVLK   U  EXP  0.0000174            G592-011     U  EXP  9.998E-10
FU-LFLAMEAF   U  EXP  0.000000001          G592-012     U  EXP  9.998E-10
FU-T4AWIRE    U  EXP  0.0000001            G592-013     U  EXP  1.285E-09
FU-RFLAMEAF   U  EXP  0.000000001          G009-G12     U  EXP  0.000001498
FU-RBURST     U  EXP  0.000004             G495.        U  EXP  5.68E-08
FU-GA_NIS     U  EXP  0.00000078           CFFUEAIRMIX  U  EXP  0.0000025
FU-GA_OC      U  EXP  0.0000002            CFSTOREFIRE  U  EXP  0.001
FU-TMPSNSFR   U  EXP  0.000000001          FUINADVJETT  U  EXP  6.878E-08
FU-PMPFRXP    U  EXP  0.000000001          G265         U  EXP  2.997E-10
FU-XPOPPLK    U  EXP  0.00000211           G683.        U  EXP  1E-14
G160-G286     U  EXP  7.322E-08            G695.        U  EXP  0.00000001
G160-G287     U  EXP  3.07E-09             G696.        U  EXP  0.000000099
PP-415-ENG1   U  EXP  8.94E-11             G697.        U  EXP  0.000000064
PP-415-ENG2   U  EXP  8.94E-11             G698.        U  EXP  0.00000001
PP-415-ENG3   U  EXP  8.94E-11             G715         U  EXP  0.000000006
PP-415-ENG4   U  EXP  8.94E-11             CF-APUOPFLT  U  EXP  0.019
LG-LHGRSTRC   U  EXP  6.455E-09            CF-NEGGMAN   U  EXP  0.0000277
LG-RHGRSTRC   U  EXP  6.455E-09            OX-PBTL1LK   U  EXP  0.00000011
G449          U  EXP  4.858E-08            OX-PBTL1HED  U  EXP  0.0000012
G005-ECSGRD   U  EXP  0.04732              OX-PBTL2LK   U  EXP  0.00000011
G005-BOSGRD   U  EXP  0.000001319          OX-PBTL2HED  U  EXP  0.0000012
G005OBGRATM   U  EXP  2.903E-08            OX-PBTL3LK   U  EXP  0.00000011
G009-G15      U  EXP  0.003909             OX-PBTL3HED  U  EXP  0.0000012
G009FRSPGND   U  EXP  9.687E-11            CF-G554-C4   U  EXP  0.00718
G226DRILL11   U  EXP  0.0003139            G670         U  EXP  2.882E-09
G226-EF1-1    U  EXP  0.000001205          G134         U  EXP  0.000001
G226DRILL21   U  EXP  0.0003158            G309         U  EXP  0.000001
G226-EF2-1    U  EXP  1.056E-07            G671         U  EXP  5.25E-09
G226-HF1-1    U  EXP  5.491E-09            G672         U  EXP  7.216E-09
G226DRILL12   U  EXP  0.0003139            G654         U  EXP  4.759E-09
```

G673	U	EXP	8.029E-08
G674	U	EXP	5.812E-08
G675	U	EXP	2.17E-09
G676	U	EXP	9.53E-08
G677	U	EXP	7.792E-11
G658-G100	U	EXP	0.000001
G658-G110	U	EXP	0.000001
G230	U	EXP	7.598E-08
G275	U	EXP	4.428E-09
G312E_AIRSP	U	EXP	2.174E-09
G312L_AIRSP	U	EXP	2.015E-09
HEPEVNAV	U	EXP	1.78022E-16
NVVNAVER	U	EXP	0.0003156
G166-2007AX	U	EXP	2.981E-09
G166-2007BX	U	EXP	2.981E-09
G166-2007CX	U	EXP	2.484E-10
G554TL3ENTR	U	EXP	2.048E-09
G554-ADC	U	EXP	9.895E-10
G554-CREW8	U	EXP	1.01197E-16
G554-INDERR	U	EXP	3.095E-09
G554-TR-INC	U	EXP	3.243E-11
G554-G4821	U	EXP	3.72646E-18
G554-TL-2EN	U	EXP	0.000005456
G312	U	EXP	5.885E-09
G274NMVTUPR	U	EXP	0.000001
G274VTPRNM	U	EXP	7.504E-10
G160-G285	U	EXP	1.811E-10
G530	U	EXP	1.544E-08
G629	U	EXP	1.541E-09
G644	U	EXP	1.283E-09
G005INSFPFL	U	EXP	0.004378
G009-G307	U	EXP	2.84E-08
G009-G320	U	EXP	2.074E-09
G009FRSPFLT	U	EXP	1.354E-10
G576.	U	EXP	0.00000009
G656.	U	EXP	0.000000001
G106	U	EXP	9.9E-10
ECSLDCCRTUN	U	EXP	0.000008
ECSBDCCRUN	U	EXP	3.62E-08
CD_DMC_COOL	U	EXP	0.001
CDERRPFD1HW	U	EXP	0.00019
CD_SW_DU2	U	EXP	0.000001
CDD2EFISERR	U	EXP	0.00024
NVERRMMR1HW	U	EXP	0.000162655
NVERRMMR2HW	U	EXP	0.000162655
CDERRPFD2HW	U	EXP	0.00019
CD_SW_DU3	U	EXP	0.000001
CDD1EFISERR	U	EXP	0.00024
CDERRND1HW1	U	EXP	0.00019
NV_SW_MMR	U	EXP	0.000001
OXRDCMPCOND	U	EXP	0.000001

OX-SMK-COND	U	EXP	0.00001
OXPLTDCFAIL	U	EXP	0.00001
G364-SVRACC	U	EXP	4.493E-09
G625	U	EXP	0.000001
G554-G4822	U	EXP	1.01205E-20
G109PFLNAR5	U	EXP	0.000001
G109PFLAR5	U	EXP	2E-18
G109-GL-SMK	U	EXP	0.004024
G593B	U	EXP	0.000001
G593C	U	EXP	5.003E-08
G594-ALT	U	EXP	0.000001
G594-APPR	U	EXP	0.000001
G594-LVLCHG	U	EXP	0.000001
G594-VNAV	U	EXP	0.000001
G594-VS	U	EXP	0.000001
G425-TRMLOS	U	EXP	2.1E-09
G425-TRMRUN	U	EXP	3.084E-09
G597A	U	EXP	9.945E-11
G597B	U	EXP	0.000001
G425-SMJAM	U	EXP	4.114E-11
G425APSENG1	U	EXP	0.000001
G425APSENG2	U	EXP	0.000001
G575ECSVNYC	U	EXP	0.00001316
G575ECPVNYC	U	EXP	0.00001226
G010FISLKAG	U	EXP	2.281E-08
G010FISLKHP	U	EXP	1.405E-08
G005ECSFFLT	U	EXP	0.004378
G005BLKFLT	U	EXP	0.000001365
G109GLAR5S	U	EXP	0.003952
SFCDLC	U	EXP	1.737E-07
G565.	U	EXP	9E-10
G452	U	EXP	9.9E-10
G580HIRXPRI	U	EXP	0.00002651
G580HIRXSEC	U	EXP	0.00002651
G580DPR15KL	U	EXP	0.000001896
G580DPR15KA	U	EXP	0.0001682
G274-G100	U	EXP	7.601E-08
SFCDLW	U	EXP	6.714E-08
G598APLDHAZ	U	EXP	3.628E-09
G621	U	EXP	0.000001
G607.	U	EXP	0.000001
G617.	U	EXP	0.000001
G583.	U	EXP	0.000001
G071	U	EXP	0.000001
G582.	U	EXP	0.000001
G024	U	EXP	0.000001
G023	U	EXP	0.000001
G618.	U	EXP	0.000001
G554TL3ENTA	U	EXP	8.465E-09
G598APOTHAZ	U	EXP	7.675E-09
G425APSENG3	U	EXP	0.000001

G070	U	EXP	0.000001
G702	U	EXP	0.000001
G707	U	EXP	0.000001
CFPPG642RFL	U	EXP	0.0092
CCOLL3FLT	U	EXP	0.001
CF-ATCFAILS	U	EXP	0.001
HEPLTFLOPIN	U	EXP	0.001
CFOPSBASFLS	U	EXP	0.001
CFOPBOUTRNG	U	EXP	0.001
HEPLFATCINS	U	EXP	0.001
TCRADFL	U	EXP	0.001
HEPTFVFRIFR	U	EXP	0.001
CF3PTHRAD	U	EXP	0.000001
AWCREW-ERR1	U	EXP	0.000000009
AWCREW-ERR2	U	EXP	0.000000009
SF568UNCP/U	U	EXP	3.578E-10
G642-GATE1	U	EXP	0.000001
G642-G178	U	EXP	0.000001
G653-SVFLD	U	EXP	0.000001
G599-LAND	U	EXP	0.000001
G589.	U	EXP	0.000001
G069INSFBBD	U	EXP	4.71E-08
G069INSFLFP	U	EXP	3.514E-11
G069ARBENRQ	U	EXP	3.908E-08
G069ARAENRQ	U	EXP	3.427E-10
G069LGENRQ	U	EXP	1.77E-08
G069OBOGSLS	U	EXP	0.002794
G058	U	EXP	1.29E-08
G064	U	EXP	8.762E-08
DSM	U	EXP	0.0004
APR	U	EXP	0.0002535
SUC	U	EXP	0.0001141
SUR	U	EXP	0.0001014
CBR	U	EXP	0.0000551
CBA	U	EXP	0.0000041
DRR	U	EXP	0.0000769
HA1	U	EXP	0.0000053
HA2	U	EXP	0.0000053
HA3	U	EXP	0.0000053
HA4	U	EXP	0.0000053
DC1	U	EXP	0.0003333
DC2	U	EXP	0.0003333
ECU	U	EXP	0.00008365
STP	U	EXP	0.00001281
STB	U	EXP	0.00001281
SLN	U	EXP	0.00001281
SLB	U	EXP	0.00001281
SRN	U	EXP	0.00001281
SRB	U	EXP	0.00001281
DEC	U	EXP	0.0002352
TQG	U	EXP	0.00060467
LRA	U	EXP	0.0005005
PLC	U	EXP	0.000168
SDU	U	EXP	0.000004
DCU	U	EXP	0.0000486
FMG1	U	EXP	0.000001
CMG1	U	EXP	0.000001
TOP1	U	EXP	0.0000718
BOT1	U	EXP	0.0000718
FMG2	U	EXP	0.000001
CMG2	U	EXP	0.000001
TOP2	U	EXP	0.0000718
BOT2	U	EXP	0.0000718
FMG3	U	EXP	0.000001
CMG3	U	EXP	0.000001
TOP3	U	EXP	0.0000718
BOT3	U	EXP	0.0000718
FMG8	U	EXP	0.000001
CMG8	U	EXP	0.000001
TOP8	U	EXP	0.0000718
BOT8	U	EXP	0.0000718
FMG9	U	EXP	0.000001
CMG9	U	EXP	0.000001
TOP9	U	EXP	0.0000718
BOT9	U	EXP	0.0000718
FMG10	U	EXP	0.000001
CMG10	U	EXP	0.000001
TOP10	U	EXP	0.0000718
BOT10	U	EXP	0.0000718
FMG11	U	EXP	0.000001
CMG11	U	EXP	0.000001
TOP11	U	EXP	0.0000718
BOT11	U	EXP	0.0000718
FMG12	U	EXP	0.000001
CMG12	U	EXP	0.000001
TOP12	U	EXP	0.0000718
BOT12	U	EXP	0.0000718
FMG4	U	EXP	0.000001
CMG4	U	EXP	0.000001
TOP4	U	EXP	0.0000718
BOT4	U	EXP	0.0000718
FMG5	U	EXP	0.000001
CMG5	U	EXP	0.000001
TOP5	U	EXP	0.0000718
BOT5	U	EXP	0.0000718
FMG6	U	EXP	0.000001
CMG6	U	EXP	0.000001
TOP6	U	EXP	0.0000718
BOT6	U	EXP	0.0000718
FMG7	U	EXP	0.000001
CMG7	U	EXP	0.000001
TOP7	U	EXP	0.0000718

BOT7	U	EXP	0.0000718
ESMPRO	U	EXP	0.00021851
MSU	U	EXP	0.001104
MRX	U	EXP	0.00023099
SAU	U	EXP	0.0003169
SRX	U	EXP	0.00012306
SFE	U	EXP	0.00010911
FR1	U	EXP	0.00007948
FR2	U	EXP	0.00007948
FR3	U	EXP	0.00007948
FR4	U	EXP	0.00007948
LB1	U	EXP	0.000000024
LS1-1	U	EXP	0.000000026
LS1-2	U	EXP	0.000000026
HB1	U	EXP	0.000000024
HS1-1	U	EXP	0.000000026
HS1-2	U	EXP	0.000000026
LB2	U	EXP	0.000000024
LS2-1	U	EXP	0.000000026
LS2-2	U	EXP	0.000000026
HB2	U	EXP	0.000000024
HS2-1	U	EXP	0.000000026
HS2-2	U	EXP	0.000000026
LB3	U	EXP	0.000000024
LS3-1	U	EXP	0.000000026
LS3-2	U	EXP	0.000000026
HB3	U	EXP	0.000000024
HS3-1	U	EXP	0.000000026
HS3-2	U	EXP	0.000000026
LB4	U	EXP	0.000000024
LS4-1	U	EXP	0.000000026
LS4-2	U	EXP	0.000000026
HB4	U	EXP	0.000000024
HS4-1	U	EXP	0.000000026
HS4-2	U	EXP	0.000000026
MDCU1	U	EXP	0.00021882
MDCU2	U	EXP	0.00021882
EGI1UNIT	U	EXP	0.000333
EGI1POW	U	EXP	0.0000491
EGI1TRAY	U	EXP	0.0000124
EGI1GPS	U	EXP	0.0000375
EGI1ANT	U	EXP	0.0000111
EGI2UNIT	U	EXP	0.000333
EGI2POW	U	EXP	0.0000491
EGI2TRAY	U	EXP	0.0000124
EGI2ANT	U	EXP	0.0000111
EGI2GPS	U	EXP	0.0000375
FMC1	U	EXP	0.000333
FMC2	U	EXP	0.000333
USMS1COM	U	EXP	0.00076923
USMS1CB	U	EXP	0.0000192
USMS2COM	U	EXP	0.00076923
USMS2CB	U	EXP	0.0000192
USMS3COM	U	EXP	0.00076923
USMS3CB	U	EXP	0.0000192
USMS4COM	U	EXP	0.00076923
USMS4CB	U	EXP	0.0000192
MADCOMPAMP	U	EXP	0.0004354
DETECTHEAD	U	EXP	0.00058989
VECTMAGNET	U	EXP	0.00004214
MADBASE	U	EXP	0.000002231
MADCB	U	EXP	0.0000096
DATAPROC	U	EXP	0.00031958
SIGPROC	U	EXP	0.00032909
INS-GPS	U	EXP	0.00015797
REC-EXC	U	EXP	0.00047708
RFTRANS	U	EXP	0.00058922
RFCTRL	U	EXP	0.00015179
RFSCAN	U	EXP	0.0006594
IFFUNIT	U	EXP	0.00037
CRYPTOFILL	U	EXP	0.000017
IFFCB	U	EXP	0.0000048
DRADTDCR	U	EXP	0.000013
PRADTDCR	U	EXP	0.000013
1553BUS1	U	EXP	0.0000046
1553BUS2	U	EXP	0.0000046
PSW	U	EXP	0.0000136
PNL	U	EXP	0.000010824
MT1	U	EXP	0.000001
MT2	U	EXP	0.000001
RFC	U	EXP	0.0002091
PSU	U	EXP	0.0000362
RPA	U	EXP	0.0000036
BAS	U	EXP	0.0000283
MCD	U	EXP	0.0000771
MPS	U	EXP	0.0000317
SCI	U	EXP	0.0000509
AUI	U	EXP	0.0002299
PRO	U	EXP	0.0000895
KEY	U	EXP	0.000367647
DSC	U	EXP	0.00010204
ANT1	U	EXP	0.0001333
MCM1	U	EXP	0.0002199
ANT2	U	EXP	0.0001333
MCM2	U	EXP	0.0002199
JTT	U	EXP	0.000001
RAD	U	EXP	0.0004431
BAT	U	EXP	0.0000525
SWA	U	EXP	0.0000136
PWR	U	EXP	0.0000136
ATT	U	EXP	0.000005
RAT	U	EXP	0.000001

ATB	U	EXP	0.000005
RAB	U	EXP	0.000001
BAC	U	EXP	0.000000032
TBA	U	EXP	0.0000001
BBC	U	EXP	0.000000032
CBB	U	EXP	0.000000141
TBB	U	EXP	0.0000001
V/U1XCVR	U	EXP	0.0003074
V/U1FILT	U	EXP	0.0003908
V/U1ANT	U	EXP	0.0000445
V/U1DTOA	U	EXP	0.00010204
V/U1LOGU	U	EXP	0.0000446
V/U2XCVR	U	EXP	0.0003074
V/U2ANT	U	EXP	0.0000445
V/U2DTOA	U	EXP	0.00010204
V/U2LOGU	U	EXP	0.0000446
V/U2VFILT	U	EXP	0.0003908
V/U2UFILT	U	EXP	0.0002999
V/U3XCVR	U	EXP	0.0003074
V/U3FILT	U	EXP	0.0003908
V/U3ANT	U	EXP	0.0000445
V/U3DTOA	U	EXP	0.00010204
V/U3LOGU	U	EXP	0.0000446
V/U4XCVR	U	EXP	0.0003074
V/U4FILT	U	EXP	0.0003908
V/U4ANT	U	EXP	0.0000445
V/U4DTOA	U	EXP	0.00010204
V/U4LOGU	U	EXP	0.0000446
V/U5XCVR	U	EXP	0.0003074
V/U5FILT	U	EXP	0.0003908
V/U5ANT	U	EXP	0.0000445
V/U5DTOA	U	EXP	0.00010204
V/U5LOGU	U	EXP	0.0000446
HF1RECEXC	U	EXP	0.0001964
HF1POWAMP	U	EXP	0.0001131
HF1ANTTUN	U	EXP	0.000141
HF1PPSFIL	U	EXP	0.0000935
HF1DTOA	U	EXP	0.00010204
HF1SHUNT	U	EXP	0.00003906
HF2RECEXC	U	EXP	0.0001964
HF2POWAMP	U	EXP	0.0001131
HF2ANTTUN	U	EXP	0.000187512
HF2PPSFIL	U	EXP	0.0000935
HF2DTOA	U	EXP	0.00010204
HF2SHUNT	U	EXP	0.00003906
HF2POWSW	U	EXP	0.0000068
HF2FWDANT	U	EXP	0.000001
HF2CNTANT	U	EXP	0.000001
HF2AFTANT	U	EXP	0.000001
HF2ANT	U	EXP	0.000001
HF2GNDANT	U	EXP	0.000001

L11DTS	U	EXP	0.0001272
L11DLP	U	EXP	0.0001012
L11CRYPT	U	EXP	0.0001963
L11PWRSW	U	EXP	0.0000068
L11CB	U	EXP	0.0000105
TTYMOD	U	EXP	0.0003876
TTYKEYGEN	U	EXP	0.000367467
TTYPOWSW	U	EXP	0.0000068
B3I1	U	EXP	0.0000119
FI1	U	EXP	0.0000062
PSU1	U	EXP	0.0000382
MC1	U	EXP	0.0000285
FLX1	U	EXP	0.00000014
CA1	U	EXP	0.00000014
B3I2	U	EXP	0.0000119
FI2	U	EXP	0.0000062
PSU2	U	EXP	0.0000382
MC2	U	EXP	0.0000285
FLX2	U	EXP	0.00000014
CA2	U	EXP	0.00000014
TUR1	U	EXP	0.001045
SCU1	U	EXP	0.00170196
PDU1	U	EXP	0.00021692
HGP1	U	EXP	0.000025
TLU1	U	EXP	0.000021
PWP1	U	EXP	0.00000018
DPS1	U	EXP	0.0000136
RTS1	U	EXP	0.0000136
1553BUS	U	EXP	0.00000001
IOPIFU	U	EXP	0.00013313
IOP1	U	EXP	0.000124
IOP2	U	EXP	0.000124
TAC1PWRSW	U	EXP	0.00000549
TCSOSMD	U	EXP	0.00002
TCSOSSCSI	U	EXP	0.0000001
ORD1PEP	U	EXP	0.00006666
ORD1PSU	U	EXP	0.00002381
ORD1SONRS	U	EXP	0.00007594
ORD2PEP	U	EXP	0.00006666
ORD2PSU	U	EXP	0.00002381
ORD2SONRS	U	EXP	0.00007594
ACO1PEP2	U	EXP	0.00006666
ACO1PSU	U	EXP	0.00002381
ACO1PEP1	U	EXP	0.00006666
SPARECHRD	U	EXP	0.0001
ACO1CHRD	U	EXP	0.0001
ACO1KEYP	U	EXP	0.00003333
ACO1KEYB	U	EXP	0.00002527
ACO1KEYPL	U	EXP	0.00002
ACO1ROLBAL	U	EXP	0.0001
ACO2PSU	U	EXP	0.00002381

ACO2PEP1	U	EXP	0.00006666
ACO2PEP2	U	EXP	0.00006666
ACO2CHRD	U	EXP	0.0001
ACO2KEYP	U	EXP	0.00003333
ACO2KEYB	U	EXP	0.00002527
ACO2KEYPL	U	EXP	0.00002
ACO2ROLBAL	U	EXP	0.0001
PILOTTCP	U	EXP	0.00012189
PILOTDP	U	EXP	0.00013548
PILOTTCPIFU	U	EXP	0.00002495
PILOTCHRD	U	EXP	0.0001
WK1DP	U	EXP	0.00013548
WK1PSU	U	EXP	0.00002381
WK1CHRD	U	EXP	0.0001
WK2DP	U	EXP	0.00013548
WK2PSU	U	EXP	0.00002381
WK2CHRD	U	EXP	0.0001
WK3DP	U	EXP	0.00013548
WK3PSU	U	EXP	0.00002381
WK3CHRD	U	EXP	0.0001
WK4DP	U	EXP	0.00013548
WK4PSU	U	EXP	0.00002381
WK4CHRD	U	EXP	0.0001
WK5DP	U	EXP	0.00013548
WK5PSU	U	EXP	0.00002381
WK5CHRD	U	EXP	0.0001
WK1PEP1	U	EXP	0.00006666
WK1PEP2	U	EXP	0.00006666
WK1KEYP	U	EXP	0.00003333
WK1KEYB	U	EXP	0.00002527
WK1KEYPL	U	EXP	0.00002527
WK1ROLBAL	U	EXP	0.0001
WK2PEP1	U	EXP	0.00006666
WK2PEP2	U	EXP	0.00006666
WK2KEYP	U	EXP	0.00003333
WK2KEYB	U	EXP	0.00002527
WK2KEYPL	U	EXP	0.00002527
WK2ROLBAL	U	EXP	0.0001
WK3PEP1	U	EXP	0.00006666
WK3PEP2	U	EXP	0.00006666
WK3KEYP	U	EXP	0.00003333
WK3KEYB	U	EXP	0.00002527
WK3KEYPL	U	EXP	0.00002527
WK3ROLBAL	U	EXP	0.0001
WK4PEP1	U	EXP	0.00006666
WK4PEP2	U	EXP	0.00006666
WK4KEYP	U	EXP	0.00003333
WK4KEYB	U	EXP	0.00002527
WK4KEYPL	U	EXP	0.00002527
WK4ROLBAL	U	EXP	0.0001
WK5PEP1	U	EXP	0.00006666
WK5PEP2	U	EXP	0.00006666
WK5KEYP	U	EXP	0.00003333
WK5KEYB	U	EXP	0.00002527
WK5KEYPL	U	EXP	0.00002527
WK5ROLBAL	U	EXP	0.0001
VIDINTU	U	EXP	0.00008333
HDDR	U	EXP	0.0007692
MAGDISK1	U	EXP	0.00002
MAGDISK2	U	EXP	0.00002
CBS	U	EXP	0.0000366
TXTPRINT	U	EXP	0.0001
SMGMTPROC	U	EXP	0.0002857
BBAYDR	U	EXP	0.0000011
STNCTRLU1	U	EXP	0.000114285
STNCTRLU2	U	EXP	0.000114285
STNCTRLU3	U	EXP	0.000114285
STNCTRLU4	U	EXP	0.000114285
STNCTRLU5	U	EXP	0.000114285
WRELSW1	U	EXP	0.00000201
WRELSW2	U	EXP	0.00000201
WRELSW3	U	EXP	0.0000491
WRELSW4	U	EXP	0.000013
SINGSHOT1	U	EXP	0.00086207
SINGSHOT2	U	EXP	0.00086207
TENSHOT1	U	EXP	0.00018885
TENSHOT2	U	EXP	0.00018885
TENSHOT3	U	EXP	0.00018885
TENSHOT4	U	EXP	0.00018885

DISCRETE EVENTS 26

CF-APU-TIT	FP-FIRE14	0.18
CF-OX-IGN1	G005LEAK1	0.1
CF-OX-IGN3	G005-LEAK3	0.1
CF-OX-IGN4	G005-LEAK4	0.1
CF-OX-COMB1	G005LEAK1	0.01
CF-OX-COMB3	G005-LEAK3	0.01
CF-OX-COMB4	G005-LEAK4	0.01
CF-FP-LK-2	G226-HF1-2	0.05
CF-FP-LK-3	G226-HF1-3	0.05
CF-FP-LK-4	G226-HF1-4	0.05
CF-FP-LK-1	G226-HF1-1	0.05
CFFLAP>ENV.	G265	0.9
CFFLJETTIS	CFSTOREFIRE	0.005
FUFLPROCEA	CFSTOREFIRE	0.01
FUFLPROCEB	CFSTOREFIRE	0.01
CFOXAR5COND	OXRDCMPCOND	0.01
G109-GLNAR5	OXRDCMPCOND	0.061
ECCALT15KFT	G580CDPR15K	0.224
CFJTRELENV.	G070	0.9
CFWEPLNCH.	WEP_LAUNCH	0.9
HEPG642TCLS	AWCREW-ERR2	0.01

```
CFTMRSKD      G6533TOAVD       0.5             CF-ONGROUND  G592-004         0.05
HE3FDETDAY    AWCREW-ERR1      0.5             CFPLE10000   OXRDCMPCOND      0.109
HE3FVFRIFR    AWCREW-ERR2      0.01
```

B.1.2 MFOP Enablers

There were no MFOP enablers for the first generation of runs.

B.1.3 Platform, MFOP, Mission and Phase Data

```
FLEET 3

PF1 3 MFOP1 MFOP2 MFOP3
PF2 3 MFOP3 MFOP2 MFOP1
PF3 3 MFOP2 MFOP3 MFOP1

MFOPS 3

MFOP1 3 ASUW ASW SAR
MFOP2 3 ASW ASUW SAR
MFOP3 3 SAR ASW ASUW

MISSIONS 3

ASUW 15

FlightCheck1   SET     1       EngStart
EngStart       SET     0.25    Take-Off
Take-Off       SET     0.0167        Climb
               ABANDON Land
Climb          NORM    0.25    0.0003  Transit
               ABANDON Approach
Transit        NORM    3.0     0.15    Descent1
               ABANDON Return
Descent1       SET     0.25    On-Task1
               ABANDON Off-Task
On-Task1       NORM    2.5     0.8
               OUTCOME NoAtt   0.9     On-Task1 [2]
               OUTCOME Attac   0.1     Attack1
               ABANDON Off-Task
On-Task1 [2]   NORM    2.5     0.8     Off-Task
               ABANDON Off-Task
Attack1        SET 0.15        On-Task1 [2]
               ABANDON         Off-Task
Off-Task       REV     Descent1        Return
```

```
                ABANDON Return
Return          REV     Transit Descent2
Descent2        REV     Climb   Approach
Approach        SET     0.167   Land
Land            SET     0.083   Taxi
Taxi            NORM    0.5     0.07 END

ASW 15

FlightCheck2    SET      1              EngStart
EngStart        SET      0.25           Take-Off
Take-Off        SET      0.0167         Climb
                ABANDON                 Land
Climb           NORM    0.25    0.0003  Transit
                ABANDON                 Approach
Transit         NORM    3.0     0.15    Descent1
                ABANDON                 Return
Descent1        SET     0.25            On-Task2
                ABANDON                 Off-Task
On-Task2        NORM    2.5     0.8
                OUTCOME NoAtt   0.9     On-Task2 [2]
                OUTCOME Attac   0.1     Attack2
                ABANDON                 Off-Task
On-Task2 [2]    NORM    2.5     0.8     Off-Task
                ABANDON                 Off-Task
Attack2         SET 0.15                On-Task2
                ABANDON                 Off-Task
Off-Task        REV     Descent1        Return
                ABANDON                 Return
Return          REV     Transit         Descent2
Descent2        REV     Climb           Approach
Approach        SET     0.167           Land
Land            SET     0.083           Taxi
Taxi            NORM    0.5     0.07    END

SAR 13
FlightCheck3    SET      1              EngStart
EngStart        SET      0.25           Take-Off
Take-Off        SET      0.0167         Climb
                ABANDON                 Land
Climb           NORM    0.25    0.0003  Transit
                ABANDON                 Approach
Transit         NORM    3.0     0.15    Descent1
                ABANDON                 Return
Descent1        SET     0.25            On-Task3
                ABANDON                 Off-Task
On-Task3        NORM    2.5     0.8     Off-Task
                ABANDON                 Off-Task
Off-Task        REV     Descent1        Return
                ABANDON                 Return
Return          REV Transit             Descent2
```

```
Descent2        REV Climb               Approach
Approach        SET     0.167           Land
Land            SET     0.083           Taxi
Taxi            NORM    0.5     0.07    END
```

B.2 Second Generation

B.2.1 Fault Tree, Component Data and Discrete Events file

The second generation of the fault tree, component and discrete event data file made some alterations to current gates, and some new additions. The changes from the previous file are shown below.

```
PHASE   Climb   68

...
NVMMRSYSERR     *       2       NVMSEMAIN       NVM3SYSERR
NVMSEMAIN       *       2       NVM1SYSERR      NVM2SYSERR
NVM1SYSERR      +       4       NVERRMMR1HW     NV_SW_MMR       ECSLDCCRTUN
                                ECSBDCCRUN
NVM2SYSERR      +       2       NVERRMMR2HW     NV_SW_MMR
NVM3SYSERR      +       2       NVERRMMR3HW     NV_SW_MMR_R
NVECPDISP       +       3       NVMMRSYSERR     NVEPFD2PROC     NVCPND1
...
G425-1          +       6       G593    G594    G596    G597    G598-1  G599-1
G594            *       2       G594-MAIN       G594-RED
G594-MAIN       +       5       G594-ALT        G594-APPR       G594-LVLCHG     G594-VNAV
                                G594-VS
G594-RED        +       5       G594-ALTR       G594-APPRR      G594LVLCHGR
                                G594-VNAVR      G594-VSR
G597            +       2       G597A   G597B
...

PHASE   On-Task1        29

...
SD_WEPS1        +       3       SD_WEP_PL1      G023    G618.
SD_WEP_PL1      *       2       WEP_LAUNCH      WEP_LAUNCHR
WEP_LAUNCH      +       4       G582.   G583.   G607.   G617.
WEP_LAUNCHR     +       4       G582.R  G583.R  G607.R  G617.R
FIRE_OTHER6     +       5       G621    G481    G010    G009-3  G226
SD_NON_WEP      *       2       SDNW_MAIN       SDNW_RED
SDNW_MAIN       +       2       G024    G071
SDNW_RED        +       2       G024R   G071R
```

```
G174-2          +       1       G554TL3ENTA
...

ABANDONS7       +       7       DASS    FMS     RCS     TCS     MAD     G617.   G607.
...

ABANDONS7A      +       8       DASS    ESM     FMS     RADAR-IFF       RCS     TCS
                                G617.   G607.
...

ABANDONS7B      +       8       DASS    FMS     RADAR-IFF       RCS     TCS     EOSDS
                                G617.   G607.
...

PHASE   Attack1 19
...
SD_WEP_PL2      *       3       WEP_LAUNCH      WEP_LAUNCHR     CFWEPLNCH.
...

ABANDONS8       +       8       DASS    FMS     RCS     TCS     MAD     STORES  G617.
                                G607.
...

ABANDONS8A      +       9       DASS    ESM     FMS     RADAR-IFF       RCS     TCS
                                STORES  G617.   G607.
...

GENERAL FAILURES 1

GENTOP  + 1 EMPTYCOMP

ABANDONS 1

AGENERAL + 1 EMPTYCOMP2

COMPONENTS

EMPTYCOMP       U       EXP     1E-20
EMPTYCOMP2      U       EXP     1E-20
...
NVERRMMR3HW     U       EXP     0.000162655     EXP     1e-6
...
NV_SW_MMR_R     U       EXP     0.000001        EXP 1e-9
...
G625            U       EXP     1e-7
...
G594-ALTR       U       EXP     0.000001        EXP 1e-9
G594-APPRR      U       EXP     0.000001        EXP 1e-9
G594LVLCHGR     U       EXP     0.000001        EXP 1e-9
G594-VNAVR      U       EXP     0.000001        EXP 1e-9
G594-VSR        U       EXP     0.000001        EXP 1e-9
```

```
...
G582.R          U       EXP     0.000001        EXP 1e-9
G583.R          U       EXP     0.000001        EXP 1e-9
G607.R          U       EXP     0.000001        EXP 1e-9
G617.R          U       EXP     0.000001        EXP 1e-9
...
G642-GATE1      U       EXP     1e-7
G642-G178       U       EXP     1e-7
...

SENSORS

SENS_G594       G594            U       EXP 7e-9        EXP 1e-4
SENS_SDNW       SD_NON_WEP      U       EXP 7e-9        EXP 1e-4
SENS_CDD1       CDD1EFISERR     U       EXP 7e-9        EXP 1e-4
SENS_CDD2       CDD2EFISERR     U       EXP 7e-9        EXP 1e-4
SENS_CDDE       CDERRPFD2HW     U       EXP 7e-9        EXP 1e-4
SENS_G274       G274NMVTUPR     U       EXP 7e-9        EXP 1e-4
SENS_NVEPIT     NVEPITDISP      U       EXP 7e-9        EXP 1e-4
SENS_NVECPD     NVECPDISP       U       EXP 7e-9        EXP 1e-4
```

B.2.2 MFOP Enablers

Due to the emptiness of the first generation input file, the second version was entirely new. The update is shown below.

```
LRI 1

LRI_NV 1/2      NVEPITDISP      NVECPDISP

SCHEDULED LIFE 2

LRI_NV  50
G621   70

INSPECTIONS 0

REDUNDANCY 4
G594-MAIN  S SENS_G594 1 G594-RED
NVMSEMAIN  M 0.9       1 NVM3SYSERR
WEP_LAUNCH M 0.95      1 WEP_LAUNCHR
SDNW_MAIN  S SENS_SDNW 1 SDNW_RED

PROGNOSTICS 4

CDD1EFISERR 3 30 40 60 70 85 95
CDD2EFISERR 3 30 40 70 80 92 97
```

```
CDERRPFD2HW 4 10 20 40 50 60 70 80 90
G274NMVTUPR 2 30 40 80 90
```

B.3 Third Generation

B.3.1 Fault Tree, Component Data and Discrete Events file

```
PHASE   FlightCheck1
...
ABANDONS        150
...
RADWARNR        +       3       SYS_APR S_HET_BIG       SIGNALREC
SYS_APR         *       2       APR     APR_RED
MISWARNR        +       2       ECU     MISSENS
TRDS            +       3       DEC     SYS_TQG         DASSLNCH
SYS_TQG         *       2       TQG     TQG_RED
CFD             +       3       SDU     DCU     CFDF
S_HET_BIG       *       2       SUPERHET        S_HET_RED
SUPERHET        +       2       SUC     SUR
S_HET_RED       +       2       SUC_RED SUR_RED
SIGNALREC       +       3       CDBAND  MIDHIBAND       SIGCONV
...
MAD             +       5       MCA     DETECTHEAD      VECTMAGNET      MADBASE
                                MADCB
MCA             *       2       MADCOMPAMP      MCA_RED
...
SMGMTSYS        +       4       SMGP    STNCTRLU        BBAYDR  WEAPRELSW
SMGP            *       2       SMGMTPROC       SMG_RED
...

PHASE   FlightCheck2
...
ABANDONS        38
...
SIGDEL          +       2       SPIN_BIG        MAINCHAN
SPIN_BIG        *       2       SPINCHAN        S_CHAN_RED
SPINCHAN        +       2       PROCFAIL        SAU
S_CHAN_RED      +       2       PROCFAIL_R      SAU_RED
MAINCHAN        +       2       ANTCLUST        MRX
ANTCLUST        *       4       ACLUST1         ACLUST2         ACLUST3         ACLUST4
PROCFAIL        +       2       SRX     SFE
PROCFAIL_R      +       2       SRX_RED SFE_RED
...
RADAR-IFF       +       10      DATAPROC        SIGPROC IFFINTGTR       INS-GPS
                                SYS_REC-EXC     RFT     RFCTRL  RFSCAN  RADCOOLING
                                1553BBUS
```

```
IFFINTGTR       +       3       IFFUNIT CRYPTOFILL      IFFCB
SYS_REC-EXC     *       2       REC-EXC REC-EXC_RED
RFT             *       2       RFTRANS RFT_RED
RADCOOLING      *       2       DRADTDCR        PRADTDCR
...

PHASE   FlightCheck3

ABANDONS        5

ABANDONS1B      +       6       DASS    FMS     RADAR-IFF       RCS     TCS     EOSDS
EOSDS           +       7       TUR1    SCU     PDU     HGP1    TLU1    PWP1    EOSDSCP
PDU             *       2       PDU1    PDU1_RED
SCU             *       2       SCU1    SCU1_RED
EOSDSCP         +       2       DPS1    RTS1

COMPONENTS

APR_RED         U       EXP     0.0002535
SUC_RED         U       EXP     0.0001141
SUR_RED         U       EXP     0.0001014
CBR             U       EXP     0.00000551
DRR             U       EXP     0.00000769
TQG             U       EXP     0.000060467
TQG_RED         U       EXP     0.000060467
LRA             U       EXP     0.00005005
MRX             U       EXP     0.000023099
SAU_RED         U       EXP     0.0003169
SRX_RED         U       EXP     0.00012306
SFE_RED         U       EXP     0.00010911
MCA_RED         U       EXP     0.0004354
DETECTHEAD      U       EXP     0.000058989
REC-EXC_RED     U       EXP     0.00047708
RFTRANS         U       EXP     0.000058922
RFT_RED         U       EXP     0.000058922
RFSCAN          U       EXP     0.00006594
TUR1            U       EXP     0.0001045
SCU1            U       EXP     0.000170196
SCU1_RED        U       EXP     0.000170196
PDU1_RED        U       EXP     0.00021692
SMGMTPROC       U       EXP     0.00002857
SMG_RED         U       EXP     0.00002857

SENSORS

...
SENS_SCU1       SCU1            U       EXP 7e-9        EXP 1e-4
SENS_TQG        TQG             U       EXP 7e-9        EXP 1e-4
SENS_RFT        RFTRANS         U       EXP 7e-9        EXP 1e-4
SENS_REC        REC-EXC         U       EXP 7e-9        EXP 1e-4
SENS_MCA        MADCOMPAMP      U       EXP 7e-9        EXP 1e-4
```

```
SENS_APR        APR             U       EXP 7e-9        EXP 1e-4
SENS_SMG        SMGMTPROC       U       EXP 7e-9        EXP 1e-4
```

B.3.2 MFOP Enablers

```
LRI 4

LRI_NV          1/2     NVEPITDISP      NVECPDISP
LRI_RWR         1/4     APR             SUPERHET        SIGNALREC       S_HET_RED
LRI_TRDS        1/3     DEC             TQG             DASSLNCH
LRI_PDU1        1/2     PDU1            PDU1_RED

SCHEDULED LIFE 5

LRI_NV  50
G621 70
SCU1 50
TUR1 80
RFSCAN 80

...

REDUNDANCY 14
G594-MAIN       S SENS_G594     1 G594-RED
NVMSEMAIN       M 0.9           1 NVM3SYSERR
WEP_LAUNCH      M 0.95          1 WEP_LAUNCHR
SDNW_MAIN       S SENS_SDNW     1 SDNW_RED
SCU1            S SENS_SCU1     1 SCU1_RED
TQG             S SENS_TQG      1 TQG_RED
RFTRANS         S SENS_RFT      1 RFT_RED
REC-EXC         S SENS_REC      1 REC-EXC_RED
MADCOMPAMP      S SENS_MCA      1 MCA_RED
SMGMTPROC       S SENS_SMG      1 SMG_RED
APR             S SENS_APR      1 APR_RED
PDU1            M 0.9           1 PDU1_RED
SUPERHET        M 0.9           1 S_HET_RED
SPINCHAN        M 0.9           1 S_CHAN_RED
```