A method for the architectural design of distributed control systems for large, civil jet engines: a systems engineering approach

A Method for the Architectural Design of Distributed Control Systems for Large, Civil Jet Engines

A Systems Engineering Approach

by

Duncan Bourne

Doctoral Thesis

Submitted in partial fulfilment of the requirements for the award of Engineering Doctorate (EngD) of

Loughborough University

October 2011

Abstract

The design of distributed control systems (DCSs) for large, civil gas turbine engines is a complex architectural challenge. To date, the majority of research into DCSs has focused on the contributing technologies and high temperature electronics rather than the architecture of the system itself. This thesis proposes a method for the architectural design of distributed systems using a genetic algorithm to generate, evaluate and refine designs. The proposed designs are analysed for their architectural quality, lifecycle value and commercial benefit. The method is presented along with results proving the concept. Whilst the method described here is applied exclusively to Distributed Control System (DCS) for jet engines, the principles and methods could be adapted for a broad range of complex systems.

Contents

Aero Engine Controls is a 50-50 joint venture company formed between Rolls-Royce plc and Goodrich Corporation[1]. The company commenced business in January 2009 and produces many of the electronic and hydromechanical sub-systems which constitute the Full Authority Digital Engine Controller (FADEC) used on many Rolls-Royce jet engines[2]. The FADEC is a collection of engine-mounted sub-systems including pumps, actuators and electronic hardware that control, sustain and monitor the engine during flight. Aero Engine Controls designs and manufactures these products at their sites in Birmingham and Derby in the UK. The company operates in both civil and military markets and provides control systems for the engines powering airframes such as the Boeing 787 'Dreamliner' and the Eurofighter. The first major project for the joint venture was the development of the FADEC for the Rolls-Royce Trent XWB engine to power the Airbus XWB (Xtra Wide Body). This engine powers the Airbus XWB airframes due to enter service in 2013.

Prior to the formation of the joint venture, Goodrich Engine Control Systems (GECS) operated as an independent supplier of FADEC components. Their business was principally with Rolls-Royce although the company conducted projects with other engine suppliers. GECS was formally known as TRW Aeronautical Systems (formerly Lucas Aerospace) before being acquired by Goodrich Corporation in 2002. Goodrich Corporation was formed following the transformation of B.F. Goodrich, who after selling their tyre brand to Michelin in 1988, acquired a group of aerospace and chemical businesses. Subsequent acquisitions have given Goodrich Corporation capability in airborne actuation systems, landing gear, nacelles and many other aerospace technologies. Goodrich Corporation has bases in America, Australia, South-East Asia and Europe, as well as service centres at airports around the world.

Rolls-Royce is one of the most familiar names in British industry and the second largest manufacturer of aero-engines behind General Electric (GE). Alongside aero-gas turbines, the company operates in the marine and power generation markets. Following near collapse and nationalisation in 1971, (and subsequent separation of the luxury car business in 1973) the business was re-privatised in 1989 as Rolls-Royce plc. Today, 53% of its sales are in the commercial aerospace sector led by the Trent family of turbofan engines. Trent engines power airframes such as the Boeing 777, 787 and many Airbus airframes such as the A330 and A380. Additionally, Rolls-Royce design and manufacture engines for business jets, regional jets and a broad range of military aircraft. Rolls-Royce are the sole provider of engines for the Airbus XWB series of aircraft. The deal constitutes the company's largest ever contract. The Rolls-Royce headquarters is in Derby (UK) although the company has operations in America, Germany and field service centres at many of the world's major airports.

[1]Aero Engine Controls is formally titled, ''Rolls-Royce Goodrich Engine Control Systems'' but operates under the trade name ''Aero Engine Controls''

[2]This research was conceived by Goodrich Engine Control Systems prior to the establishment of the joint venture company.

Acknowledgments

The research presented in this thesis has been undertaken whilst enrolled as a Research Enginer at Loughborough University and on placement at Aero Engine Controls in Birmingham.

Firstly, I would like to thank my academic and industrial supervisors, Dr. Roger Dixon and Alex Horne for their dedication and encouragement. Their input and knowledge have been invaluable to this work. Roger's tireless commitment has provided the fillip allowing this research to evolve and produce the given outcomes. Despite increasing responsibility in his own job, Alex has supported and promoted my work within Aero Engine Controls. Alex has also dedicated time to the Systems Engineering Doctorate Centre (SEDC) management committee. I would like to thank Professor Charles Dickerson for introducing me to the formalisms of system architecture and the principles of graph theory which underpin so much of this work.

Furthermore, I am indebted to the staff of Aero Engine Controls for their time and willingness to share their work and experience with me. Special thanks to members of the Electronics Technology Team who have accommodated me and where possible, involved me in their professional and social activities. Members of the team have provided a wealth of information and understanding which has contributed greatly to the project.

Background information regarding the aerospace industry and the company circumstances could not have been acquired without the generously afforded time of many Aero Engine Controls staff. Whilst many staff have contributed their expertise, special thanks are due to Peter Smout, Gary Chandler and Ian James.

Thanks are due to the staff who manage and administer the SEDC at Loughborough University. Particular thanks go to Sharon Henson, Karen Holmes, Bob Malcolm and again to Roger Dixon - their tolerance of my non-cooperation in administrative and procedural matters has been admirable.

My thanks extend to the members of the Control Systems Research Group at Loughborough University and fellow students enrolled on the Engineering Doctorate programme. Their humour, understanding and own personal experiences have provided a priceless antidote to the tribulations of academic research.

I acknowledge the role of the Engineering and Physical Sciences Research Council (EPSRC) and Aero Engine Controls in providing the funding, infrastructure and support that enabled this research.

Finally, I would like to thank my friends and colleagues with whom I have shared so many experiences over the past three years: Luke Bowman, Nathaneal Rice, Craig Shelley, Jeffery Escosio, Richard Smart and David Hodgkinson. As fellow house mates, climbers, mountain bikers, cyclists and pub-goers, their friendship has been invaluable.

Acronyms

3D	3-dimensional
ACO	Ant Colony Optimisation
AFDX	Avionics Full Duplex Switched Ethernet
AIDA	Automatic Control in Distributed Applications
ANTLE	Affordable Near-Term Low Emissions
API	Advanced Peripheral Interface
ASTOVL	Advanced Short Take-off Vertical Landing
CAD	Computer Aided Design
CAN	Controller Area Network *(Bosch)*
CC	Combustion Chamber
CCB	Common Circuit Block
CCS	Centralised Control System
CPU	Central Processing Unit
D&C	Delay & Cancelation
DCS	Distributed Control System
DECWG	Distributed Engine Control Working Group
DFADEC	Distributed Full Authority Digital Engine Controller
DND	Do Not Despatch
DOP	Data Over Power
EA	Evolutionary Algorithm
EEC	Electronic Engine Controller
EHM	Engine Health Monitoring
EIS	Entry Into Service
EMI	Electromagnetic Interference
EPSRC	Engineering and Physical Sciences Research Council
FADEC	Full Authority Digital Engine Controller
FANI	Farthest Neighbour Nearest Insertion
FH	Flight-Hour
FMU	Fuel Metering Unit
FMV	Fuel Metering Valve
FOHE	Fuel Oil Heat Exchanger
FUD	Full Up Despatch
GA	Genetic Algorithm
GE	General Electric
GECS	Goodrich Engine Control Systems
HiPECS	High Performance Engine Control System
HiPECS-A	High Performance Engine Control System 'A'
HiPECS-A	High Performance Engine Control System 'B'
HiTEAM	High Temperature Electronics for Aerospace Manufacture
HiTED	High Temperature Electronic Device
HMU	hydromechanical unit
HP	High Pressure
HPC	High Pressure Compressor
IATA	International Air Transport Association
ICB	Inter-Channel Bus
IO	input/output
IRR	Internal Rate of Return
IPC	Intermediate Pressure Compressor

LP	Low Pressure
LTD	Long Term Despatch
LOTC	Loss of Thrust Control
LVDT	Linear Variable Displacement Transducer
MCS	Monte Carlo simulation
MDD	Manufacturer Defined Despatch
MEL	Minimum Equipment List
MIRR	Modified Internal Rate of Return
MIT	Massachusetts Institute of Technology
MLO	Mid-life Overhaul
MoD	Ministry of Defence
MOEA	Multi-Objective Evolutionary Algorithm
MOGA	Multi-Objective Genetic Algorithm
MTTF	Mean Time To Failure
MTTUR	Mean Time to Unscheduled Removal
NFF	No Fault Found
NSGA-II	Non-dominated Sorting Genetic Algorithm
NPD	New Product Development
NPV	Net Present Value
PCB	Printed Circuit Board
PIR	Periodic Inspection/Repair
PMA	Permanent Magnet Alternator
POA	Power Optimised Aircraft
PSO	Particle Swarm Optimisation
R&O	Repair & Overhaul
RR	Rolls-Royce
RTDCS	Real-Time Distributed Control System
SAT	Surrounding Air Temperature
SEDC	Systems Engineering Doctorate Centre
SiC	Silicon Carbide *(electronic devices)*
SOI	Silicon on Insulator
SQL	Structured Query Language
SPEA2	Strength Pareto Evolutionary Algorithm
STD	Short Term Despatch
STF	System Test Facility
SysML	Systems Modelling Language
TDMA	Time Division Multiple Access *(network)*
TGT	Turbine Gas Temperature
TLD	Time Limited Despatch
TRL	Technology Readiness Level
TTCAN	Time Triggered CAN
TTP	Time Triggered Protocol *(TTTech time-triggered network)*
UAV	Unmanned Aerial Vehicle
UTC	University Technology Centre
UML	Unified Modelling Language
USM	Unscheduled Maintenance
VSV	Variable Stator Vane
VSVA	Variable Stator Vane Actuator
XWB	Xtra Wide Body

Executive Summary

Introduction

Distributed Control Systems (DCSs) are widely seen as an inevitable advance in gas turbine control technology. The touted gains are numerous and appeal to subsystem suppliers, engine manufacturers and airlines alike. Notional benefits include weight reduction, increased reliability, improved fault isolation and an increase in processing power to support advanced health monitoring and diagnostic algorithms.

DCSs involve the division of a centralised control system into individual sub-systems comprising of smart sensors, actuators and controllers; each individual sub-system is known as a node and contains processing capability. The nodes are physically separated and communicate via a shared data network. Sensors and actuators may exist as independent network nodes or be connected directly to a controller node. A typical DCS architecture is shown in figure 1. The degree of distribution is open to the designer. Each sensor and actuator may be afforded it's own node or groups of hardware interfaced to a control node or data concentrator.

Despite the plausible advantages, DCSs have yet to replace Centralised Control Systems (CCSs) on large, civil gas turbine engines. The unavailability of High Temperature Electronic Devices (HiTEDs) capable of surviving in the engine environment for the duration of a 25-35 year product life-cycle is an insuperable barrier to highly distributed systems. Broader concerns include system safety and the response of control system dynamics in the event of network delays or failure. These concerns are justified and tangible yet not without remedy. Better understanding of these issues and techniques for overcoming them are addressed by a wealth of academic, commercial and popular literature (see Kopetz (1995); Chen *et al.* (2007); Baillieul & Antsaklis (2007) for example). However, cardinal barriers remain - it is very difficult to conceive or propose control system architectures which provably realise the potential benefits.

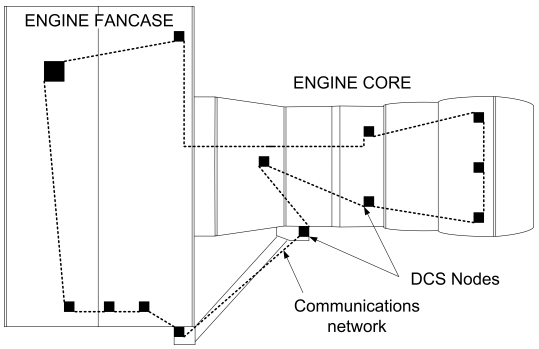


Figure 1: A distributed control system architecture. The black boxes represent the control system nodes and the interconnecting lines the data network.

It is often difficult for system designers to demonstrate that introducing new or emerging technology will add sufficient value to warrant the effort and cost of development.

DCSs are neither a technology nor a method, but a conglomeration of technologies and methods which realise an architectural form. Consequentially, industry lacks the tools and techniques necessary to design, simulate, evaluate and compare candidate distributed architectures. This research is concerned with developing such a method and using it to propose and evaluate DCS.

Method

The chosen method uses of a Genetic Algorithm (GA) to optimise distributed system architectures. A GA is a meta-heuristic optimisation method which aims to refine solutions to complex problems by mimicking the processes of evolutionary biology. The algorithm starts with a set of randomly generated solutions to a particular problem encoded as a set of digital 'chromosomes'. As the algorithm progresses, the fitness of each solution is evaluated and recorded. Solutions are ranked according to fitness with the best being duplicated and the poorest discarded. New solutions are generated by mating and mutating the best solutions in the hope that improved offspring result. The algorithm is iterative and finishes once a pre-defined period of time has elapsed or certain quality criteria are met.

In this implementation, the GA's chromosomes contain a coded representation of a DCS architecture. The chromosome dictates the location of each node on the engine, the tasks each node will perform and the electronic hardware contained within the node. This coded information is used in conjunction with a database of components and a functional model of the engine to build a meta-physical representation of the DCS in question. The meta-physical model exists entirely in software but reflects the structure, layout and properties of the control system as it would appear on engine. The model includes harnessing and has properties such as weight, size and cost.

Once built, the fitness of each DCS is assessed using three evaluation functions. Each reflects a fundamental concern of commercial DCS designers: the Architecture Evaluation Function, the Life-cycle Evaluation Function and the Business Evaluation Function.

- The architecture evaluation function considers the weight and size of the system, the length of harnessing, the modularity of the components and the quantity of data passed between them.
- The Life-cycle evaluation function uses a Monte Carlo simulation to model the performance of the DCS over a typical thirty year life-cycle. As the components fail so the cost and consequences of maintenance actions are recorded.
- The Business Evaluation Function uses the information generated by the architecture and life-cycle evaluation functions to determine the costs associated with a particular DCS. The cost of building and supporting the DCS are calculated. Using these costs, it is possible to determine how quickly an initial investment could be recouped based on the fleet size and sales pattern.

The entire scheme is shown in figure 2 below:

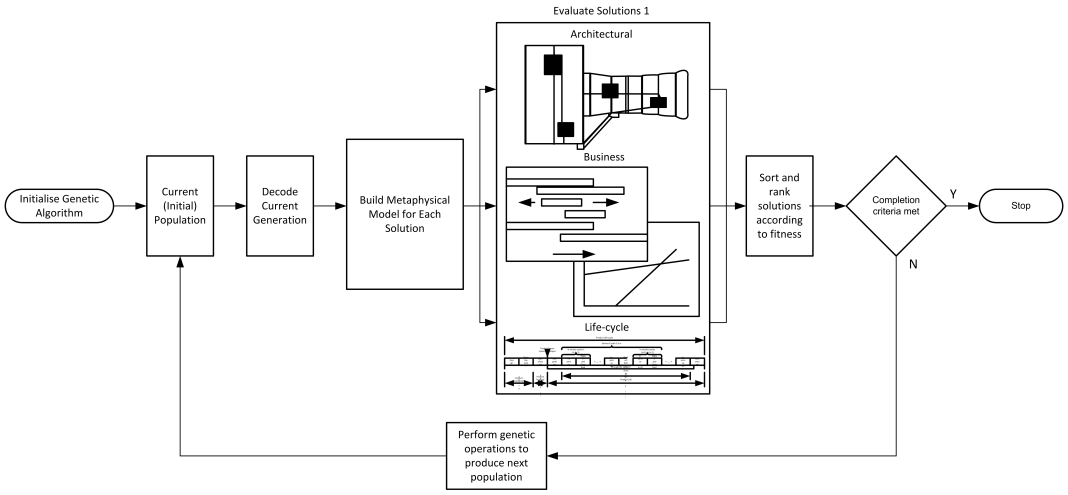


Figure 2: Schematic diagram of optimisation scheme

The research outcomes have been constrained by a lack of computational power. This thesis presents results for the optimisation scheme using the architectural evaluation alone. The lifecycle and business evaluation functions are demonstrated on a standalone test case. Despite the full implementation being implausible, it is shown in appendix B that the execution time could be reduced from 15 weeks to around 10 hours given the appropriate software implementation and hardware platform.

Results

The GA was run for 1000 iterations using the architectural optimisation alone. The system selected as the most optimal comprised two nodes: one mounted on the fancase and the other on the core. Two views of the optimal solution are shown in figures 3 and 4 below:

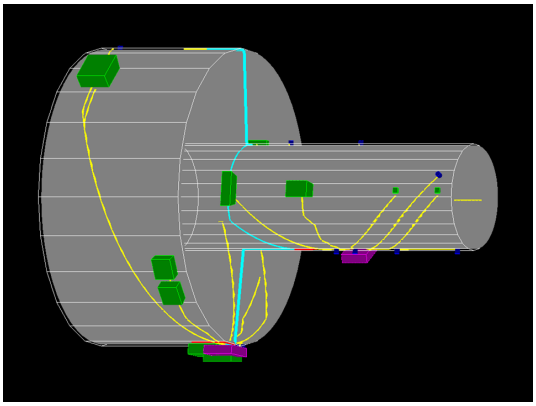


Figure 3: The two node optimal architecture

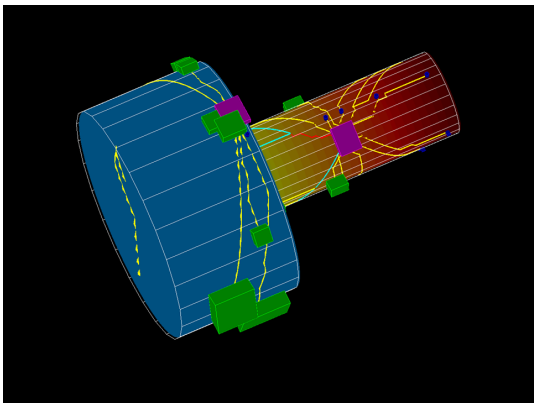


Figure 4: Two node optimal solution from underside of engine and showing temperature profile

Whilst it is impossible to prove that the architecture is truly optimal, a qualititative assessment highlights that the nodes are sensibly placed and that the components (indicated by the green boxes and blue circles) on the core are connected to the core node and the fancase components to the fancase node.

Having only been subject to the architectural evaluation, this architecture cannot be considered holistically optimal. It is likely that the influence of the lifecycle and business evaluation would move the core mounted node towards the fancase in order to reduce the surrounding air temperature and increase

reliability.

The lifecycle and business evaluation functions were exercised on a standalone test system with 14 components. Each component in the system fails according to a failure distribution incorporating both burn in and wear out failures. As the monte carlo simulation progresses, so components fail. The failures influence the operability of the system and hence the level of service disription.

The lifecycle simulation was tested on three different systems. Each system has a set of components with differing levels of reliability. The test cases are refereed to as 'R' which has the greatest reliability, 'R/3' in which the Mean Time To Failure (MTTF) of the components is one third of 'R' and 'R/5' where the MTTF is one fifth of 'R'. Each set of components is assumed to have a 30 year service life. The graphs below show the number of components replaced and in-flight shutdowns experienced during each service year.

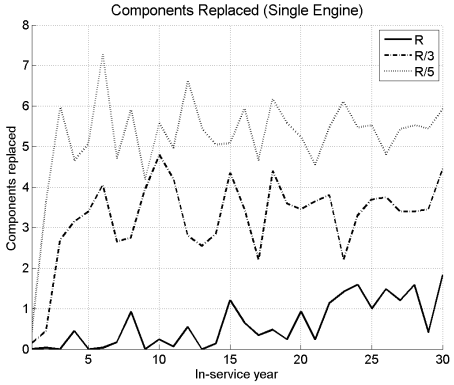


Figure 5: Average number of components replaced in a given service year

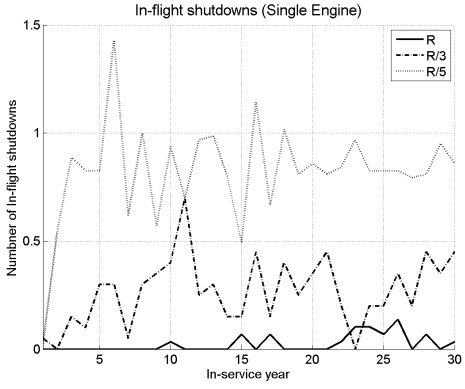


Figure 6: Average number of in-flight shutdowns due to control system unreliability

It is clear to see that the most reliable set of components, 'R' produces the most reliable and least disruptive system.

The business evaluation function takes the performance metrics from the lifecycle results, scales them to be representative of an entire fleet and attributes costs to each of the metrics. By summing these costs and the income generated from flying hours, the evaluation function is able to calculate the return on investment for the system developer. Using data taken from the three systems described above, the Net Present Value (NPV) for the three candidate system is calculated.

The graph shows how the most reliable set of components provides the greatest financial return.

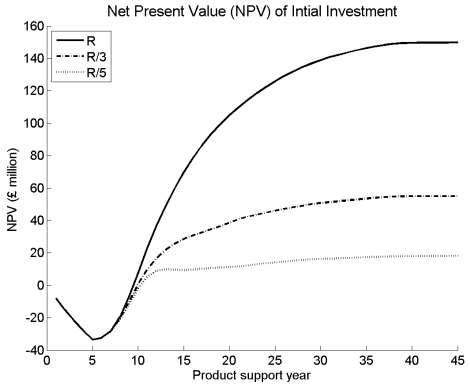


Figure 7: Annualised present values of initial investment

Conclusion

Whilst it has not been possible to run the optimisation in its entirety, the results presented above show that the constituent parts of the algorithm perform as required. The tool enables system architects to ascertain the wider consequences of archtiectural decisions. The optimisation scheme itself, the monte carlo simulation and the process of constructing the DCS architectures, provide both industrial and academic novelty.

Part I

Introduction & Background

Chapter 1

Introduction

1.1 Rationale: The Role of Product Architecture

Product architecture plays a subtle yet significant role in defining a system's properties and performance. Despite this, the drive towards a specific system architecture is often made automatically during the early stages of product development. In many instances, the natural tendency is to assume a system architecture based on a previous product or one which confines the proposed system to a single physical entity. Reasons for these approaches include the retention of experience, conceptual simplicity, retention of capabilities gained during the development of legacy systems and the difficulty of quantifying and proving the value of novel architectures. However, being that 90% (Smith & Reinertsen, 1997) of a product's financial cost and performance may be attributed to its initial design, decisions regarding product architecture are irremissibly significant. Effective use of architecture can shorten development times, help reduce or control complexity and significantly change the operation and efficiency of a business.

Modular architectures provide an alternative to centralised architectures although centralised systems may in themselves be modularised. Modules may be physically disparate or collocated and be designed to inhabit similar or vastly different physical environments. Logically, modular products are designed to maximise component reuse and promote ‘‘plug-and-play’’ composition.

1.2 Introduction: Distributed Control Systems for Aero Jet Engines

Present day jet engines are controlled and monitored by a series of hydraulic, fuel-draulic, electrical and pneumatic sub-systems known as the Full Authority Digital Engine Controller (FADEC). At the heart of the FADEC is the centralised Electronic Engine Controller (EEC) responsible for control and safety of the wider engine system.

The EEC is a dual redundant computing system hosting engine control, monitoring and safety functions. Each channel has an independent power supply, computing platform and interface circuitry to connect the unit to transducers on the engine chassis. Both channels reside in a single physical unit. The EEC receives the pilot thrust demand and controls the engine subsystems to achieve the desired performance. Furthermore, the EEC is responsible for the prompt shutdown of the engine in the event of an over-speed or over-temperature condition that would otherwise compromise the engine or airframe. The sensors and actuators associated with the engine sub-systems are located across the engine chassis and connected to the EEC by wiring harnesses as shown in figure 1.2.1. It is commonplace that the EEC is mounted on the engine fancase to protect the electronics from the extreme heat of the compressor, combustor and turbine stages.

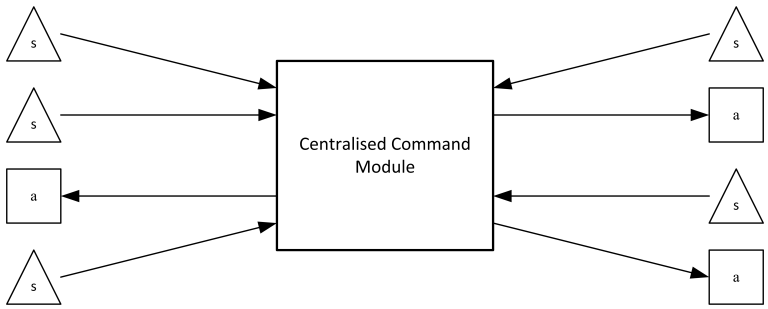


Figure 1.2.1: High-level overview of a centralised control system. All sensors (s) and actuators (a) are connected to a single command module containing all the processing power

It has long been postulated that a distributed EEC would contribute to improvements in control system performance and lifecycle costs. Distributed Control Systems (DCSs) involve the division of a control system into individual sub-systems comprising of smart sensors, actuators and controllers; each separate sub-system is known as a node and may contain processing capability. Nodes are physically separated and communicate via a shared data network as outlined in figure 1.2.2:

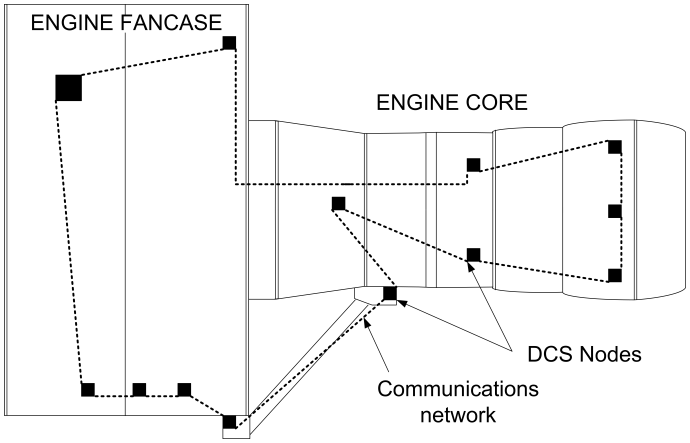


Figure 1.2.2: Outline architecture of a DCS showing nodes (black boxes) connected to a data network (dashed lines). The architecture shown is arbitrary

The sensors and actuators may exist as independent nodes or be directly coupled to a controller node. The topology of such systems varies considerably depending on application and may be flat or hierarchical as shown by figures 1.2.3 and 1.2.4.

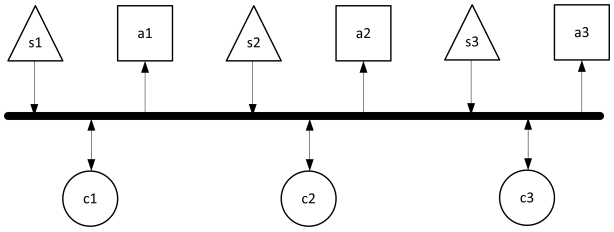


Figure 1.2.3: Flat distributed system architecture without a central command module. The sensors and actuators reside on the network as independent nodes and communicate between themselves to achieve global system control.

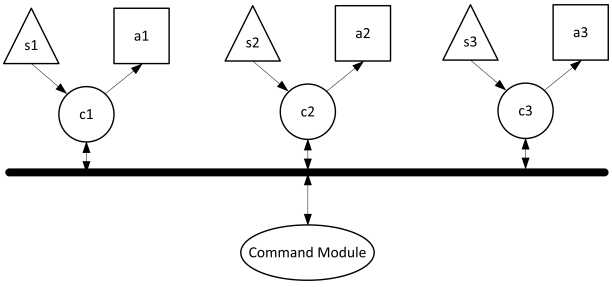


Figure 1.2.4: Hierarchical distributed control system - the distributed controllers receive set point information from a central command module and control local actuators based on local sensor measurements to achieve global control. The controller and associated sensors and actuators constitute a node

This research considers the architectural design of DCSs for controlling large, civil gas turbine engines. The following section describes the motivation for this work.

1.3 Motivation for the Study of DCSs

Intuitively, Distributed Control Systems offer a raft of commercial and technical gains across the system lifecycle. These benefits include (in no particular order):

- An overall reduction in control system weight
- Reduction in harnessing and harness complexity
- The ability to extend or modify a system once in service and protect against obsolescence using targeted upgrades

- Reuse of modular components during system design (hence savings in development time and certification costs)
- Improved fault containment and isolation
- The ability to partition technical risk and new engine technologies
- Simplified testing (at node level)
- Decreased maintenance times
- Increased control system processing power which could be used to:
  - Implement more sophisticated engine control laws
  - Increase diagnostic and prognostic capability
  - Increase sampling rates and measurement accuracy
- Reduced Electromagnetic Interference (EMI)
- Increased number of despatch configurations *(see section 7.5)*
- Provide a platform for the introduction of new and emerging technologies

Despite the extensile list of benefits and the belief that the obtrusion of distributed control systems is inevitable, such systems have yet to become commonplace beyond demonstrator engines. Previous demonstrator programmes have failed to convince system developers that distributed systems are commercially viable or intrinsically capable of achieving the widely advertised benefits. Whilst many of the tools, techniques and technologies required to realise these systems are in themselves mature, the process of integrating them to design and evaluate distributed architectures has yet to be formalised.

The practice of architecting complex systems such as the DCS is much practiced yet poorly understood. There are few proven techniques or mathematical formalisms that permit architectural design decisions to be related to in-service performance. Bringing such formalisms to the discipline of architectural design is a key research topic in systems engineering.

1.3.1 The Wider Motivation

Airlines require more efficient engines and increased reliability to decrease fuel and operating costs. These demands will only be met through the introduction of new technology and an increased focus on innovation.

Reducing fuel consumption is not only an environmental concern, but perhaps more pertinently, a commercial necessity. Rising fuel costs have hit the industry hard and contributed to the collapse of several carriers such as Silverlink and xL. Fuel efficiency

has become pivotal to the survival of airlines. This need has translated into pressure on engine manufacturers to improve engine design and accelerate the pace of new technology integration. (Source: International Air Transport Association (IATA) Jet Fuel Price Monitor)

Fuel technology alone is unlikely to see radical change in the short term. Bio-fuels have been demonstrated and are expected to make an impact in the next 10-15 years. Despite these emerging technologies, it seems conclusive that conventional fuels will remain in use for the foreseeable future - be it exclusively or as components of bio-fuel blends. It is unlikely that the FADEC will undergo significant changes in either structure or premises of operation whilst hydrocarbon-based fuels predominate. Many of the technologies proposed for reducing fuel burn such as lean-burn and turbine tip-clearance systems require components to be mounted on the engine core. Furthermore, the functionality of these systems is readily decoupled from the existing control laws; their integration requires an increasingly agile platform capable of adaptation and extension. New and emerging technologies driven by performance, commercial and legislative demands strengthen the case for DCSs.

1.4 Aims and Objectives

The aim of this research is to: **develop and analyse a set of candidate distributed control system architectures for controlling a large, civil turbofan engine using a systems engineering approach**. Given the industrial context of the research, the architectures shall be designed to achieve both technical performance and commercial viability.

The industrial context of this research engenders a further intangible aim - to stimulate and maintain an industrial awareness of distributed control systems, challenge the precedents and norms of legacy systems and promote mindfulness of alternative control system architectures.

In order to achieve these aims, this research uses a Genetic Algorithm (GA) as a tool to propose, evaluate and optimise DCS architectures. The research considers the context of the problem and the difficulties of designing distributed systems in order to realise the benefits listed previously in section 1.3.

The GA's current population contains coded specifications for a set of DCS architectures. The specification includes the location of each node, the control system tasks that each node will perform and the electronic interfaces contained within the node. This information is used in conjunction with a database of components and a functional model of the engine control system to construct a meta-physical model of the DCS in question. The meta-physical model exists entirely in software but reflects the structure, layout and properties of the control system as it would appear on engine. The model includes harnessing and has properties such as weight, size and cost.

Each of the architectures is evaluated for its architectural quality, (weight and size *etc*), and system performance whilst in service. Data gleaned through these evaluations is used to assess the system from a business perspective by considering the costs and return on investment. The scores given by the various evaluation functions reflect the holistic quality of the proposed architecture from the perspective of the system architect.

The following objectives allow this aim to be realised. This research will:

- Gain an understanding of the design problem, trade space and design parameters.
- Select and implement an appropriate GA for optimising system architectures.
- Develop a set of architecture frameworks upon which metaphysical models of the engine and DCS may be built
- Devise the process of building the metaphysical models from the design variables
- Develop algorithms to evaluate each of the models for architectural quality, lifecycle value and business cost.
- Realise all of the above in an appropriate software environment

The objectives listed were defined following a detailed analysis of the relevant academic literature and current industrial practice. The industrial background is discussed in chapter 2 and the literature in chapter 3. Section 3.5 of the literature review combines the findings of these chapters and the concerns of Aero Engine Controls to position the study and engender the objectives stated above.

1.4.1 Scope

No system level study could adequately address all facets and concerns of control system designers and business administrators; the breadth of applications, platforms and

implementation possibilities is vast. Consequentially, this research focuses on distributed control systems for large civil turbofan aero-engines. Examples include the Rolls-Royce Trent 1000 which powers the Boeing 787 Dreamliner and the General Electric (GE) GP7000 for the Airbus A380.

The data used in this research is guided by industrial axioms and practices. Where necessary, the data has been distorted to protect Aero Engine Control's intellectual property. This research only considers the architecture of the EEC and not the wider engine or FADEC.

1.5 Computational Constraint and Results

The optimisation framework developed in this thesis was conceived to operate as a complete system for designing and evaluating DCS architectures. However, the monte-carlo simulation used for the life-cycle analysis proved to be very processor intensive and made a full implementation infeasible. Therefore, this thesis presents results for the optimisation scheme using the architectural analysis alone. The lifecycle and business evaluation functions are demonstrated on a small-scale test case. The lifecycle simulation operates exactly as it would for a full scale implementation and all elements of the implementation have been written with scalability in mind.

The computational limitation is a product of the resources afforded to this research, rather than any inherent flaw or in the structure of the optimisation or the associated software. It is shown in Appendix B how the estimated execution time of 15 weeks could be cut to around 10 hours using only a modest quantity of relatively inexpensive computing hardware.

1.6 Thesis Structure

This thesis is divided into three parts: Introduction & Background, Method and Conclusions. Each is described presently.

1.6.1 Part I - Introduction & Background

Chapter 2 provides the background information necessary to understand the basic operation of the jet engine, FADEC and EEC. The industrial context is set by considering Aero Engine Control's previous experience with DCSs, the difficulties of distributed system design and a high-level overview of the wider commercial context. A literature review is given in chapter 3. The review highlights areas where previous academic literature has contributed to DCS design and identifies those topics which have received little attention. The final background chapter (chapter 4) describes the basic functionality of the GA, the process of selecting and developing the algorithm and the results from verification testing. Further background information is presented throughout the thesis where necessary.

1.6.2 Part II - Method

Chapter 5 discusses the architecture frameworks and the process of building the meta-physical DCS architectures. The chapter describes the structures used to capture the architectural specification and the methods for building the software models. The techniques used for harness routing and reliability assessment are discussed. Furthermore, a technique of generating and analysing systems using binary relations is presented. A brief overview of the evaluation functions concludes the chapter. The following three chapters (6, 7 & 8) explain the processes used by the evaluation functions. Chapter 6 considers the functions required to ascertain the architectural quality of the system using the meta-physical model. The lifecycle evaluation chapter (chapter 7) describes how a Monte Carlo Simulation (MCS) is used to 'fly' the engine through a 30 year service life. The performance data gained is used to determine the likely disruption that a given design will have on airline operations. Information gleaned from the lifecycle evaluation is used by the business evaluation function (chapter 8) to examine the costs associated with a particular architecture. Given that the algorithm could not be executed to its full extent, each of the evaluation chapters contains a set of results from verification testing. The results for the architectural evacuation present an optimal DCS architecture based on this evaluation alone. These results demonstrate the ability of the GA to optimise DCS architectures.

1.6.3 Conclusions

The third part of this thesis contains the conclusions. The conclusion summarises the strengths of the method and the associated contributions made by this research. Weaknesses of the method and potential improvements are discussed along with an alternative method. Subjects for further research are considered.

1.7 Contributions

The methods and techniques used in this research provide the following contributions to the field:

- A framework for the design of Distributed Control Systems for large civil jet engines using a Genetic Algorithm
- A better understanding of the technical and commercial difficulties of DCS design and techniques which may help overcome them
- A system of binary relations to describe and analyse system architecture
- Use of a Monte Carlo Simulation for ascertaining system performance over an extended lifecycle operating under Time Limited Despatch (TLD)

Chapter 2

Background Information and Industrial Context

2.1 Introduction

The introductory chapter discussed the motivation and aims for this research. This chapter provides a technical and commercial context for the research by providing background information on the jet engine, present day control systems and the business environment. Over the past 20 years, Aero Engine Controls has undertaken an number of development projects aimed at demonstrating distributed control systems for both civil and military engines. These projects are described and accompanied by a brief analysis of the lessons learned and capabilities gained.

Ultimately, the DCS proposed by the Genetic Algorithm provides a hardware platform capable of implementing the engine control laws. The information contained in this chapter is intended to highlight the design issues and considerations faced by control system architects. The basic descriptions grant familiarity with the FADEC system, basic redundancy schemes, basic control system tasks, engine parameters and the challenges of architectural design. Nearly all the information presented here is either implicitly or explicitly incorporated into the models and evaluation functions used during the optimisation. Readers familiar with the FADEC and EEC systems may wish to move directly to sections 2.4.2 and 2.5.

The explanations herein are introductory; they provide the unfamiliar reader with the

background information necessary to understand the premises of the research. In-depth information is provided throughout the thesis where necessary.

Section 2.2 gives an overview of the working principles of a large gas turbine engine, the FADEC and its subsystems. Following a brief look at the present day EEC, various options for partial distribution of the centralised EEC are presented and evaluated. Aero Engine Control's previous work on DCSs is presented.

2.2 The Gas Turbine System

The basic structure and operation of a gas turbine engine has not changed significantly since its inception in the 1930's. The working principle is as follows: The engine is a tube open at both ends - air enters the engine where a compressor raises the pressure before being mixed with fuel and expanded by combustion. The exhaust gasses from the combustion process are expelled from the engine to produce thrust. Before leaving the engine, the heated gasses are used to power a turbine which drives the input compressor via an interconnecting shaft. This increases the compressor speed and thus, the mass of air entering the combustion chamber. The processes is colloquially known as the, ''suck, squeeze, bang, blow'' cycle or more formally as the Brayton cycle. The effort required to design and manufacture gas turbine engines is disproportionate to their conceptual simplicity. A basic diagram of a turbofan engine is shown in figure 2.2.1 below.

It is unlikely that the basic functional design of a turbofan engine will ever change significantly. Open rotor engines proposed for short-haul aircraft will see the removal of the fancase but the fundamental core engine design will remain. Literature suggests that engines will become more efficient with the introduction of refined designs and lighter, more temperature tolerant materials. In contrast, the methods of actuation and control are likely to change significantly with the desire to reduce weight, maintenance times, control system complexity and the use of flammable fluids in hydraulic systems.

Ultimately, the only control input is the mass flow of fuel into the burner manifold. The control laws are complex and beyond the scope of this thesis. The modulation of fuel dictates the speed and acceleration of the engine spools and consequentially, thrust. The system responsible for controlling and monitoring fuel flow is known as the Full Authority Digital Engine Controller (FADEC). The FADEC comprises a system of both active and

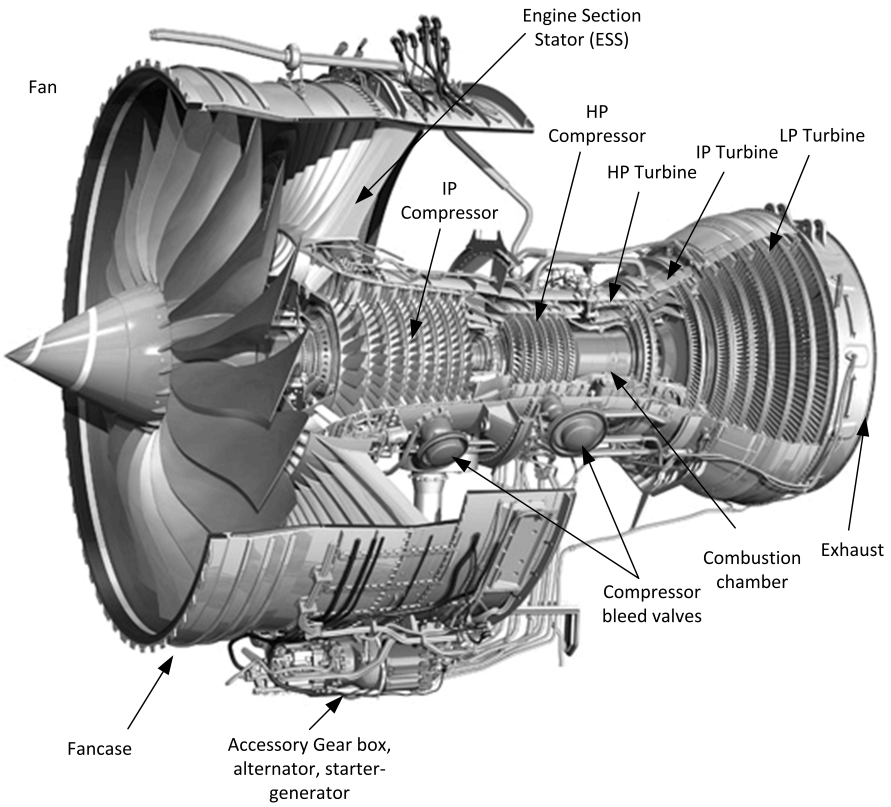


Figure 2.2.1: Three shaft turbofan engine (Trent 1000) showing the bypass fan, two core compressor stages and three turbine stages. Figure adapted from www.rolls-royce.com

passive hydromechanical subsystems under the control of the EEC.

2.3 The FADEC

Present day jet engines are controlled and monitored by a series of hydraulic, fueldraulic, electrical and pneumatic sub-systems known as the FADEC. All FADEC subsystems are mounted on the engine and permit the engine to be operated as an isolated entity providing a fuel supply, power supply and input commands are available. In normal operation, the FADEC receives a thrust demand from the cockpit and adjusts the fuel flow to maintain the desired thrust whilst removing the complexities of control from the pilot. Whilst the pilot has the overriding ability to shut the engine down, the FADEC monitors the engine for serious malfunctions and can shut the engine down automatically in response to events that could otherwise cause a hazardous or catastrophic failure or the engine or airframe [1]. Prior to the advent of electronic control systems, engine control was performed using hydromechanical apparatus incorporating valves and bellows. Without the FADEC, the pilot would have to maintain a constant thrust by calculating and adjusting the fuel flow to accommodate changes in airspeed and altitude. To do this whilst maintaining safe and efficient operation would be nigh-on impossible.

As well as controlling fuel flow, the FADEC controls the aerodynamic geometry of the engine by means of bleed valves and guide vanes to prevent stall and ensure efficient operation. As engine technologies progress and the number of control variables increases, so the FADEC will become increasingly important and complex.

The primary function of the FADEC is to control fuel flow to the burner manifold. The FADEC fuel control system comprises of three main hydromechanical and electronic elements - The Fuel Metering Unit (FMU)/hydromechanical unit (HMU) , Variable Stator Vane Actuators (VSVAs) and the EEC as shown in figure 2.3.1 below...

2.3.1 Subsidiary FADEC Systems

A brief description of the FADEC components and their functionality is given below. With reference to figure 2.3.1 the list follows the fuel path from the tank feed (l.h.s) to

[1] 'Hazardous' and 'Catastrophic' are the two most severe designations of four aircraft failure categories. Hazardous and catastrophic failures are those deemed to endanger either the lives of passengers and/or the survivability of the airframe

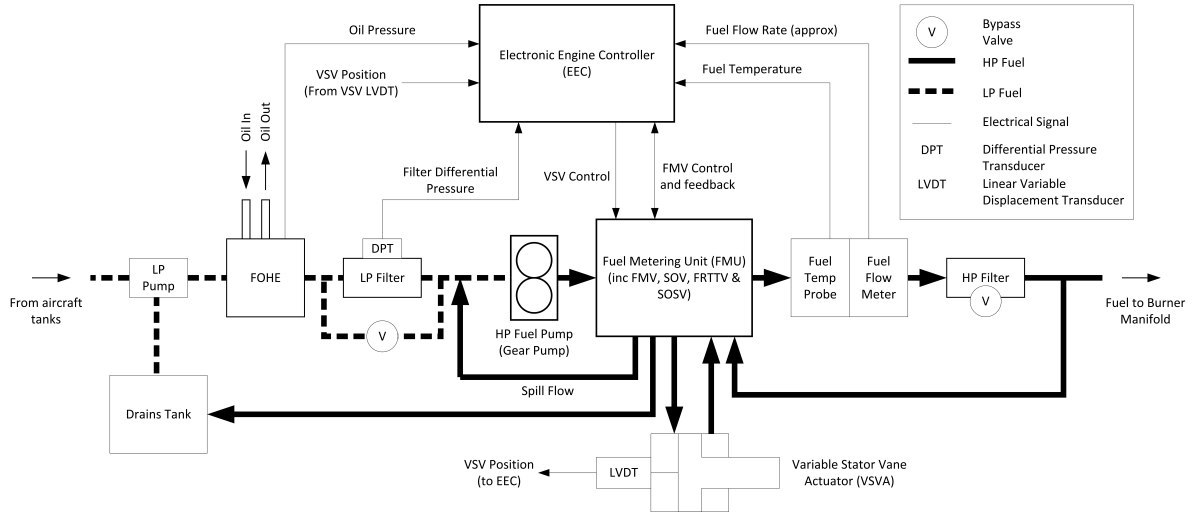


Figure 2.3.1: A typical FADEC fuel control system for a large civil turbo fan jet engine.

the burner manifold (r.h.s) but ignores the feedback and spill flows.

2.3.2 Fuel Control and the FMU/HMU

Low Pressure (LP) and High Pressure (HP) pump The LP pump increases the pressure of fuel fed from the aircraft tanks to remove modulated flows caused by dissolved gasses leaving solution.

Fuel Oil Heat Exchanger (FOHE) The FOHE is an intricate matrix of pipes designed to cool the engine oil using the cooler fuel from the aircraft tanks. It is usually located after the LP pump. This action has the combined benefit of reducing the oil temperature and removing ice particles from the fuel.

LP and HP Filter The fuel filters are designed to prevent contaminants and fuel borne ice particles from entering the fuel control system and blocking or damaging the injector nozzles.

Filter Bypass Valves Both the HP and LP filters have passive bypass valves. If the filter becomes blocked and the differential pressure across the filter rises sufficiently, the valve opens and the fuel bypasses the filter. This mechanism ensures that operation is preserved.

FMU/HMU The Fuel Metering Unit (FMU) or hydromechanical unit (HMU) unit comprises of a series of solenoid valves which regulate the fuel flow into the burner

manifold and allow for automated or pilot-initiated shutdown of the engine. At the heart on the FMU/HMU is the Fuel Metering Valve (FMV). The position of the FMV controls the fuel flow to the burners. The position of the FMV is measured using an Linear Variable Displacement Transducer (LVDT) stimulated and monitored by the EEC. The position of the FMV is controlled fueldraulically - the fuel flow which moves the valve is controlled by an electromechanical servo valve.

Fuel Flow Meter The fuel flow meter is used only for pilot information and control system verification, not to control fuel flow. Present day fuel flow metres suitable for flight applications have insufficient bandwidth and accuracy to monitor the fuel to the accuracy demanded by the control laws.

Drains Tank Valves within the FMU/HMU allow fuel to drain from the burner manifold once the engine is shut down. In addition to preventing the build-up of lacquer in the manifold and fuel nozzles, this facility helps the engine to conform to environmental legislation precluding the release of vaporised fuel into the atmosphere.

2.3.3 The Electronic Engine Controller (EEC)

The EEC is a dual redundant computing system hosting engine control and safety functions. Each channel has an independent power supply, computing platform and interface circuitry which reside in a single physical unit. The EEC interprets the pilot thrust demand and controls the engine subsystems to achieve the desired performance. Furthermore, the EEC is responsible for the prompt shutdown of the engine in the event of an over-speed, shaft-break or Loss of Thrust Control (LOTC) condition which could otherwise compromise the engine or airframe. The sensors and actuators associated with the engine sub-systems are located across the engine chassis and connected to the EEC by wiring harnesses.

Architecturally, the EEC is divided into two dual redundant channels (nominally A and B) although the system is manufactured as a single unit. Both channels are physically and electrically isolated but able to communicate via an isolated data link known as the Inter-Channel Bus (ICB). The hardware and software in both channels is identical, so the redundancy acts to accommodate sensor, actuator, harness and power supply

Figure 2.3.2: A typical EEC for a large civil engine. The two redundant channel are clearly marked on the case and physically isolated within the unit. *(figure courtesy of Aero Engine Controls)*

malfunctions alone. The ICB allows the sharing of data between channels so as inputs and outputs from one channel may be controlled by the processing element of the other. This adds a further level of redundancy and increases the number of despatch configurations.

Each channel is further divided into control and protection elements. The control element implements the control laws and commands the FMV position whilst the protection functionality monitors the engine for over-speed, shaft break and loss of thrust control events. The protection element has the authority to shut the engine down without pilot involvement. This action is permissable in instances where a slow human response may compromise safety. By preference, the control and protection elements are powered by separate isolated, supplies. Furthermore, the control and protection circuitry reside on different circuit boards and interface to sensors via separate connectors. Both channels have connections to the engine sensors and actuators. Most of the engine sensors are dual redundant whilst many of the actuators have redundant elements such as dual windings. An ideal high level architecture for an EEC is shown in figure 2.3.3:

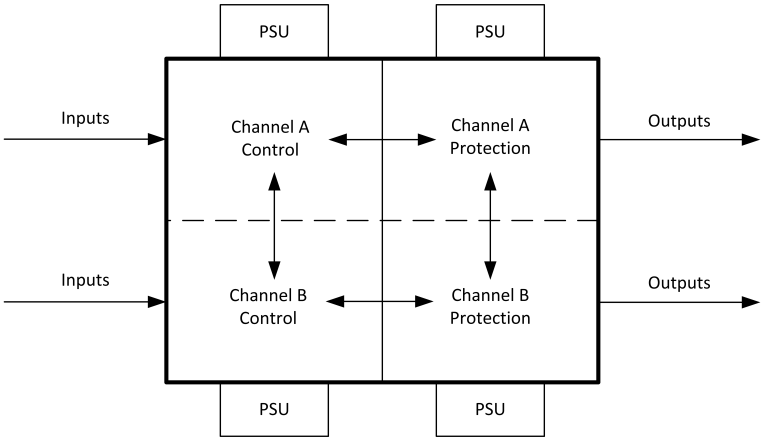


Figure 2.3.3: The ideal centralised EEC architecture. The double headed arrows show the paths of communication between the control and protection elements as well as inter-channel communication. The dotted line represents the physical and electrical isolation between channels A and B.

Each of the four elements has its own input signal conditioning, processing capability and output signal conditioning. The control elements interface to nearly all engine sensors whilst the protection elements interface to only those sensors required to detect serious engine malfunctions. In some instances, a separate sensor is provided for control and safety giving a total of four engine sensors monitoring a single parameter. The signal conditioning includes analogue buffers, amplifiers, lightning strike protection, anti-aliasing filters, multiplexors and analogue to digital converters. The computation is performed on a microprocessor which controls output circuitry such as torque motor drives, current drives, logic level drivers and data communications. Whilst the specific hardware configuration varies from application to application, the basic elements remain largely unchanged.

2.3.4 Common Circuit Blocks (CCBs)

Aero Engine Controls builds its EECs from modular components known as Common Circuit Blocks (CCBs). The company hold a library of CCBs associated with the electronic functions of the EEC. For example, there are amplifier and integrator circuit blocks as well as one for the processing and signal conditioning elements. Every transducer interface has an associated circuit block. If an EEC requires four LVDTs interfaces, four instances of the LVDT circuit block are instantiated in the design. All the circuit blocks are optimised for PCB layout and have been extensively tested. The concept allows complete EECs to be composed from primitive modules, thus decreasing development, test and certification time. The notion of CCBs is a fundamental part of the models used

in the DCS optimisation.

2.3.5 Power Sources and the Permanent Magnet Alternator (PMA)

Airframe power from a 115VAC supply is used to power the starter system. Once started, the FADEC is required to draw electrical power from an on-engine source. The generator used is known as the Permanent Magnet Alternator (PMA). The PMA harnesses rotational power form the accessory gearbox. Therefore, the output frequency (and thus available power), varies with the engine speed. The variable frequency supply is rectified and converted to a 55VDC supply which powers the EEC. The PMA is dual wound to provide redundancy - nominally, one winding powers each channel. The 55VDC supply is regulated down to voltage levels suitable for low level circuitry by independent power supplies within the EEC. Ideally, there are 4 independent supplies to provide independent power to control and protection functionality on both channels.

2.3.6 FADEC Mounting

The high-temperatures and vibration experienced during operation are a major constraint in component design and location. The engine and its environment may be broadly categorised into three distinct zones - the fancase (zone 1), a pre-combustion core zone known as ‘‘core zone 2’’ (zone 2) and a post combustion core known as ‘‘core zone 3’’ (zone 3) *(see figure 2.3.4)*. There are two temperatures of principle interest to component designers - the surface temperature of the engine chassis and the temperature of the bypass steam known as the Surrounding Air Temperature (SAT). Furthermore, the engine environment may be affected by ingestion of foreign objects and debris from within the engine or other engines on the airframe. Whilst the majority of an engine's operational life is spent in the cold surroundings of the mid or upper atmosphere, the engine must be capable of operating from the world's hottest airports.

The zones are shown in figure 2.3.4. The location of FADEC components across the engine chassis is largely dictated by the harsh environment. Conventional electronics are incapable of surviving in zones 2 and 3 so nearly all electronic devices are located on the fancase. Consequentially, the location of nearly all FADEC components is non-ideal. The FMU/HMU and the PMA require a mechanical drive which is taken from the HP

shaft to the accessory gearbox. These components are mounted together on the underside of the fancase where they are easily accessible to maintenance staff. The VSVAs are mounted directly to the unison ring which is mechanically coupled to the Variable Stator Vanes (VSVs) in the compressor. Fuel is transferred from the pumps and valves on the fancase to zones 2 and 3 by means of pipes mounted within the bifurcation duct.

2.4 The Engine Environment

Zone 1 is the least hostile, partly because the fancase is cooled by air entering the engine during flight. At cruise altitude, the temperature of the inlet air may be -50°C. The operating temperature of the fancase is typically 90--110°C and is the most uniform environment of the three engine zones. Despite providing favourable temperatures, the fancase is exposed to threats from debris, bird strike and ''fanblade-off'' events. Fanblade-off is the fracture and release of a bypass fan blade during flight. The fancase must be capable of containing a loose fanblade which may impact with a force of 200G if released at maximum engine speed. A loose fanblade deforms the fancase with an oscillatory ripple as it circulates during its deceleration. Components mounted on the fancase must be capable of withstanding this deformation and retain their mountings during a fanblade-off event.

Zone 2 provides an intermediate environment. Surface metals may reach 150°C and the SATs 250°C. Zone 2 is not exposed to as many mechanical threats as zone 1.

Zone 3 includes the combustion chamber and turbine stages and provides the most hostile environment. The gas stream may reach temperatures of 1500--2000°C whilst typical metal temperatures may be 350°C and SATs 600°C.

Engine components are designed to be unaffected by lightning strikes. On average, an airframe will be struck once a year during flight. Whilst the strike may not compromise the safety of the airframe, electronic subsystems must be designed to withstand the surge.

Zones 2 and 3 tend not to have a uniform temperature profile. Temperatures may vary considerably depending on the location of fuel and oil system elements and the way the engine is dressed. Therefore, profiles vary from engine to engine. Whilst the designation of zones conveys a broad understanding of environmental conditions, only zone 1 may be treated as a uniform and well-defined environment.

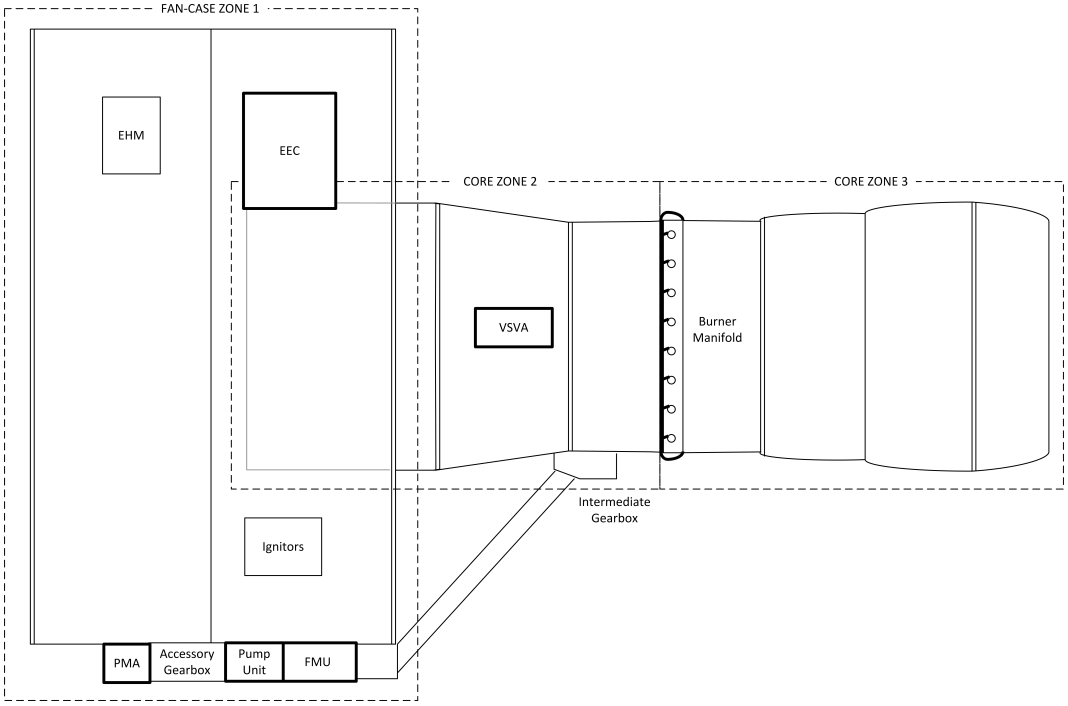


Figure 2.3.4: Location of the FADEC components. The items marked with bold outlines are those designed and manufactured by aero engine controls. Each of these components relates to the fuel delivery system or requires fueldraulic power for actuation.

The high temperatures and flammable liquids give an inherent risk of fire in the event of a leak or malfunction. Engine components are mounted in locations which prevent the spread of fire and permit the engine to be shutdown quickly and safely.

Engine malfunctions such as over-speed or oscillatory thrust may inflict mechanical damage and result in engine debris. Where possible, engines are designed to contain engine debris, although this is not always achievable. In the event of an over-speed condition, turbine blades could be ripped from their axial mountings. At full speed, each blade has the weight of a London bus and aided by a small, sharp profile, would easily rip through the engine casing. The debris could leave the engine, rip through the airframe and lodge in the other engine(s) causing wider destruction. This failure mode is known as cross-engine debris and is a considerable threat to the airframe.

2.4.1 Airframe Communications

The EEC communicates with the airframe via a dual-redundant ethernet network known as Avionics Full Duplex Switched Ethernet (AFDX). The AFDX bus communicates the speeds, pressures and temperatures that the pilot requires to verify engine function. The

EEC also communicates maintenance messages and status messages which require the pilot's or maintenance technician's attention. The messages are sent as raw data and appear on displays in the cockpit. Such messages may alert the pilot to unusually high Turbine Gas Temperature (TGT) or low oil pressure. The maintenance messages are stored on the airframe and read by maintenance staff post flight.

2.4.2 Distribution of the Present Day EEC

The following section outlines the potential for 'distributing' or 'disbursing' elements of the centralised EEC. The descriptions and evaluations introduce some of the drivers for distribution and the various architectural considerations.

The architecture of the centralised EEC was designed to allow partial distribution or 'disbursal' of the control and protection modules. The control and protection functions communicate via a serial data buses which may be extended, thus allowing the four main modules of the EEC to be spatially distributed. This approach is more consistent with 'remote electronics' than 'distributed control' but nevertheless may permit future systems to be more space efficient and architecturally robust.

The strongest drivers for separating EEC modules are reducing unit size and the likelihood of damage from fire and debris. On smaller engines where space is constrained, two or four smaller units may be easier to mount than an equivalent centralised EEC. There is potential for control units to be mounted in the cooler areas of zone 2 which may engender shorter, lighter harnesses - this benefit may compensate for the weight gain arising form separation.

The possibility of EEC destruction and loss of control during fire and debris events is an important consideration. Spatially distributed units are less likely to be damaged or destroyed by the effects of fire and cross-engine debris. Accordingly, the availability of the EEC functions to shut the engine down or gracefully degrade performance may be increased. This gain is set against the increased risk of disruption to inter-module communication in the event that data harnesses are severed or burned. Smaller units are easier to mount and less likely to be dislodged during a ''fanblade-off'' event.

The three basic schemes for EEC separation are shown in the diagrams below. Figure 2.4.1 shows the present day EEC contained in a single unit. Figure 2.4.2 shows all four

modules separated. The arrow-headed lines connecting the blocks imply the presence of a serial data link conveying information between the modules.

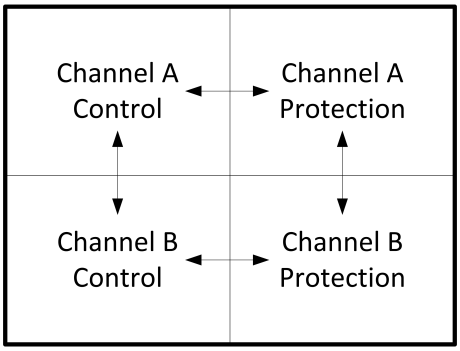


Figure 2.4.1: Centralised EEC Architecture

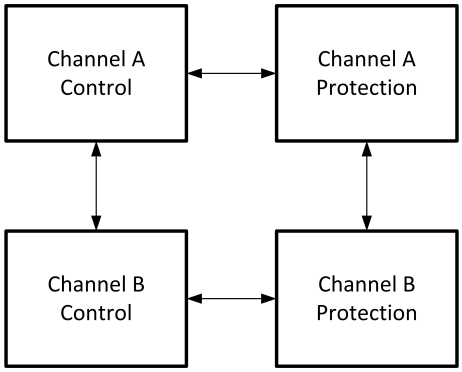


Figure 2.4.2: Architecture with both control and protection portions separated by extension of serial busses

For cost reasons alone, the arrangement shown in figure 2.4.2 is unlikely see implementation unless mounting space is severely constrained. As the most distributable of the architectures, the design is less prone to damage from fire and debris and therefore carries merit.

Other alternatives reduce the number of units from four to two by hosting a single channel or both control and protection modules in a single unit. These units would be individually smaller than a present day EEC, allow for good physical separation and be more economical to develop and produce than the four module option.

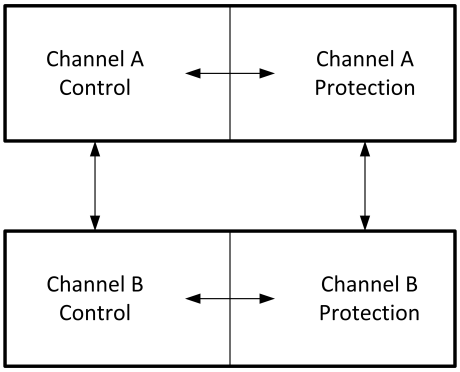


Figure 2.4.3: Channel A and B physically separated

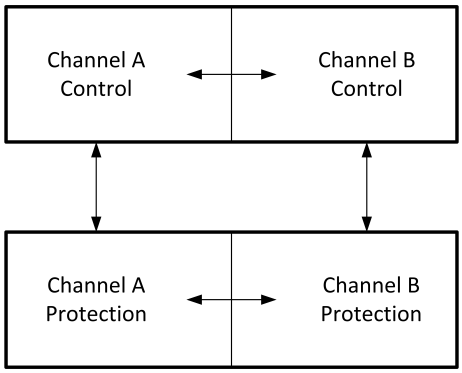


Figure 2.4.4: Control and protection functions physically separated

The arrangement shown in figure 2.4.3 is perhaps the most vaunted of the three alternative architectures. Indeed, similar schemes were successfully used in some of Aero Engine Control's legacy products. By co-hosting the the control and protection functions, the risk to inter-module communications is decreased and the necessary spares holding reduced. However, the size of the individual units and the conterminous nature of the

control and protection functions, make the units less suited to core mounting.

The final architecture figure 2.4.3 is similar in physical form to that in figure 2.4.4 but hosts both channel's control and protection modules within the same physical unit. The desirability of this architecture relies substantially on the routing of data harnesses between modules and the likelihood of both serial links being severed simultaneously.

When compared to those used in demonstrator programmes such as High Performance Engine Control System (HiPECS) and Affordable Near-Term Low Emissions (ANTLE) (section 2.5) these architecture bear overt simplicity. However, they represent potential solutions to design scenarios where space is a fundamental constraint as with the open rotor engine (see figure 2.4.5). Each offers low level dispersion without modification to the fundamental electronic or software design. Their conceptual simplicity makes them low risk and hence commercially attractive. From the perspective of the DCS designer, these configurations swash as commercially tangible alternatives to more advanced architectures. The value of each alterative is defined using uncomplicated benchmarks against which all DCSs will be compared. Adversely, such configurations are unlikely to yield systems which deliver many of the benefits associated with more elaborate DCSs.

2.5 AEC's Experience with Distributed Control Systems

THE FOLLOWING SECTION WAS WRITTEN BY THE AUTHOR AND EDITED BY EMPLOYEES OF AERO ENGINE CONTROLS. MINOR MODIFICATIONS WERE MADE TO THE ORIGINAL TEXT.

DCSs are widely seen as an inevitable advance in gas turbine control technology. The perceived gains are numerous and appeal to subsystem suppliers, engine manufacturers and airlines alike. Potential benefits include weight reduction, increased reliability, improved fault isolation, an increase in processing power necessary to support advanced performance, health monitoring, diagnostic and prognostic algorithms and an open architecture which enables cost-effective up-grades for engine control and monitoring systems.

Since the early 1990's, Aero Engine Controls has undertaken several technology demonstrator programs aimed at advancing the understanding and design of distributed control systems. Affordable high temperature electronics, or a means to enable electronics to survive in the harsh engine environment were recognised as key enablers; as a result, a

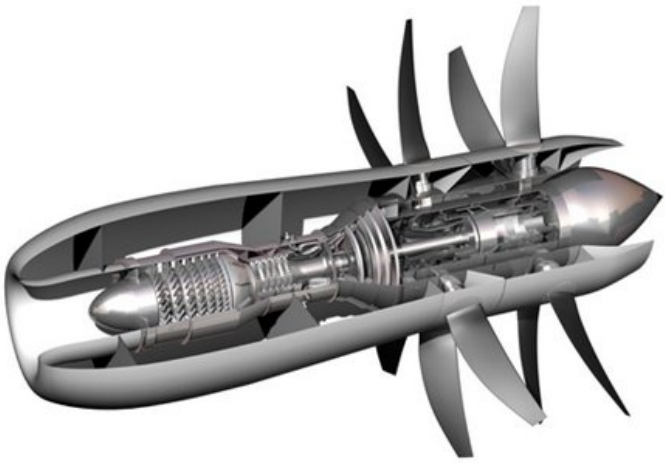


Figure 2.4.5: The lack of a fancase and small diameter core of the open rotor engine makes it unlikely that a centralised dual-channel EEC could be mounted on engine. Source:www.rolls-royce.com

high temperature electronics technology acquisition programme was launched *(see section 2.5.4)*.

The following sections summarise the key development programs undertaken by Aero Engine Controls, and the capabilities gained:

2.5.1 High Performance Engine Control System (HiPECS-A) c. 1997 - 2000

High Performance Engine Control System 'B' (HiPECS-A) was a Ministry of Defence (MoD) funded programme to develop distributed components for military transport aircraft. The chosen architecture included a core-mounted data concentrator, a 'smart' FMU and a fan-case mounted data concentrator/controller. Inter-node communication was achieved using a Controller Area Network (CAN) augmented with a proprietary Time Division Multiple Access (TDMA) layer to achieve the necessary determinism. Loop closure for the fuel control and metering functions were performed using electronics in the smart FMU.

The system was demonstrated on an in-house System Test Facility (STF) using rack mounted nodes and a wholly integrated smart FMU *(see figure 2.5.1)*. Despite not being run on engine, the HiPECS demonstrator was a fully functional, dual redundant control system and included all the fault detection and safety features of a contemporary commercial system.

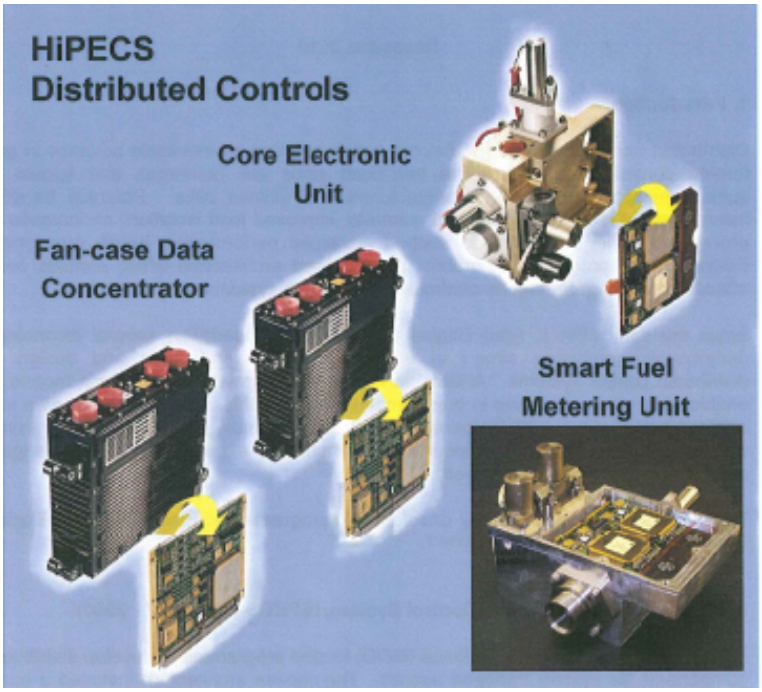


Figure 2.5.1: Elements of the HiPECS-A demonstrator *(figure courtesy of Aero Engine Controls)*

2.5.2 High Performance Engine Control System (HiPECS-B) c. 2000 - 2002

Like its predecessor, HiPECS-B considered the application of DCSs to military engines. The MoD funded project was undertaken in collaboration with Rolls-Royce. The project comprised a systems study of both centralised and distributed architectures for controlling a hypothetical military engine. In order that the proposed designs could be evaluated against likely future requirements, the engine was assumed to include advanced features such as staged combustion (for both main flow and reheat), thrust vectoring and advanced health monitoring.

A 13-node architecture was proposed and evaluated. A subset of these nodes was realised as an off-engine demonstrator exercised on a Rolls-Royce STF.

The systems study concluded that the distributed architecture offered weight savings, reliability improvements and reduced likelihood of lost missions when compared to its centralised counterpart. Furthermore, the technology demonstrator proved the integrity of the CAN communications protocol and the potential for DCSs as a future technology.

2.5.3 Advanced Near-Term Low Emissions Engine (ANTLE) c. 2000 - 2005

The European funded ANTLE program used the same basic DCS architecture as the HiPECS-A programme, with two nodes and a smart FMU. The system was based on the functionality of the Trent 500 controller and was realised using electronics technology developed during the HiTEAM 2 project *(see section 2.5.4)*. In March 2005, the ANTLE control system became the first DCS to control a Rolls-Royce demonstrator engine.

The ANTLE system included a smart actuator in the form of an electrically driven Variable Inlet Guide Vane Actuator (VIGVA). The actuator incorporated both control and power electronics. As in previous DCSs, a CAN network was used for inter-node communication.

The DCS designed for ANTLE was carried forward into the Power Optimised Aircraft (POA) programme which followed in 2008 *(see figure 2.5.2)*. Whilst Aero Engine Controls was not involved with its development, the system was expanded to include additional interfaces.

2.5.4 High Temperature Electronics Projects

HiTEAM 1 c. 1994 - 1998

High Temperature Electronics for Aerospace Manufacture (HiTEAM) 1 was a government supported project aimed at developing the art of high temperature electronic design using bulk-silicon components. The project was undertaken in collaboration with a mining industry partner and investigated the design and manufacture of electronic circuitry for high temperature environments. Whilst no high-temperature parts were produced or tested, the work provided an opportunity to ascertain the availability of High Temperature Electronic Devices (HiTEDs) and the suitability of Printed Circuit Board (PCB) manufacturing techniques.

HiTEAM 2 c. 1999 - 2002

HiTEAM 2 continued the work of HiTEAM 1 into a demonstration and evaluation phase. The work saw the introduction of hybrid circuits *(see figure 2.5.3)* and a number of Silicon on Insulator (SOI) components. Demonstrator printed circuit boards were manufactured using conventional materials, bulk silicon and high-temperature solder. To meet the

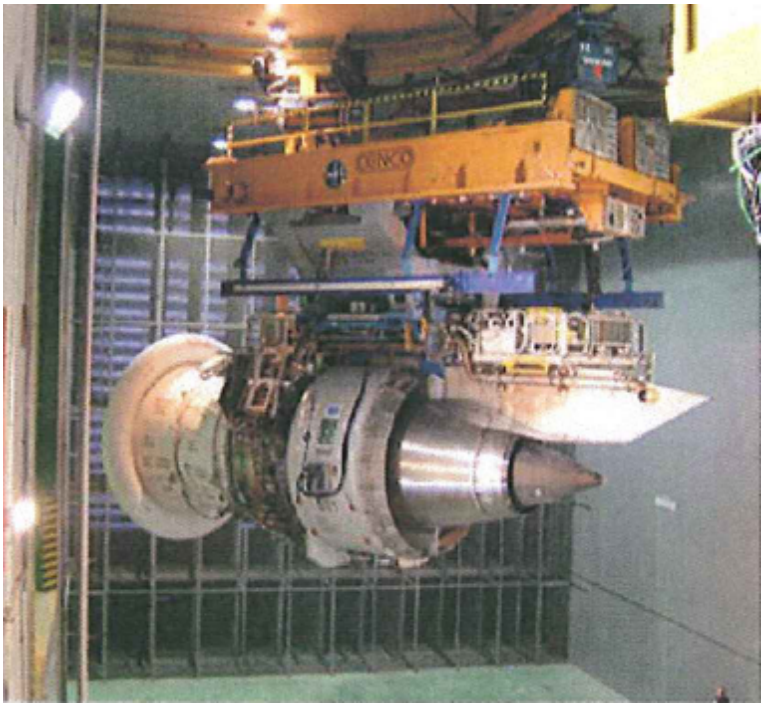


Figure 2.5.2: The Aero Engine Controls ANTLE demonstrator was successfully adapted to control the POA demonstrator in 2008 *(figure courtesy of Aero Engine Controls)*

requirements of the engine environment, the need for chip-and-wire was confirmed *(see figure 2.5.4)*.

The smart FMU incorporated a computer, 40V power supply unit, drives for torque motor, solenoid and a LVDT. This unit successfully demonstrated functional operation at 200°C.

Environmentally Friendly Engine (EFE) 2006 - 2011

Aero Engine Controls has produced a fully functional 170°C electronic demonstrator, and continues to invest in demonstrating 250°C packaging and thermal management technologies. For example, Aero Engine Controls is developing a high temperature lightning protection solution based on Silicon Carbide (SiC) diode technology.

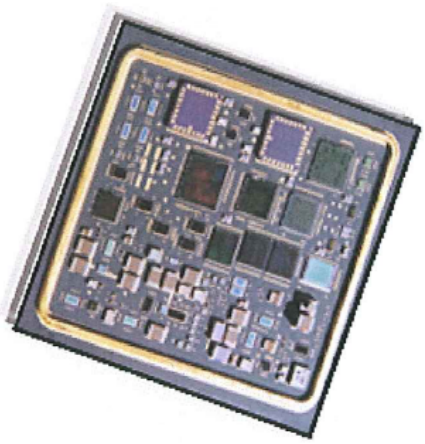


Figure 2.5.3: Hybrid circuit such as that used in HiTEAM 2 *(figure courtesy of Aero Engine Controls)*

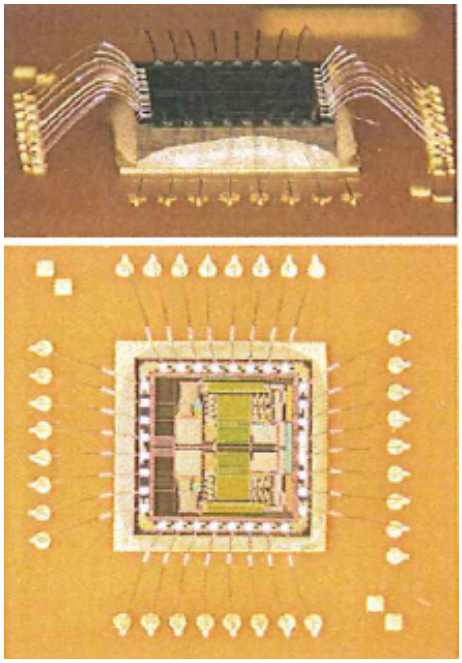


Figure 2.5.4: Wire bonding *(figure courtesy of Aero Engine Controls)*

2.5.5 Summary

As the functionality demanded from engine control systems increases, so the complexity, computational demands and physical dimensions of systems will grow. These development programs have demonstrated that DCSs offer a plausible alternative to the centralised control system. DCSs have the potential to reduce weight, size and increase reliability whilst accommodating new functionality and offering an expandable platform capable of accommodating low cost upgrades.

Aero Engine Controls has successfully operated a distributed architecture on demonstrator engines, showing how such a system is plausible and realisable. In doing so, the capability to design distributed systems has been gained; the challenge remains to devise architectures and component technologies which most readily realise the benefits.

Chapter 3

Literature Review

Chapters 1 and 2 introduced the notion of DCSs, their industrial context and the lessons learned from previous demonstrator programmes. This literature review considers academic contributions to both the art of DCS design and the component technologies necessary to realise them. Literature reviews specific to GAs, the method and evaluation functions are presented in their respective chapters.

3.1 Introduction

There is a surprising lack of academic and popular literature relating to the architectural design and implementation of Distributed Control Systems (DCSs). This is particularly true of DCSs for real-time safety critical applications and consequentially, their application to jet engine control. This may be due to the industrial nature of distributed system design whereby the specifics of application determine so many of the system's properties. Commercial secrecy may also have thwarted potential publications. Indeed, this research is naturally set apart from conventional research having stemmed from an occurring industrial need with a predefined context, rather than the academic process of reviewing and furthering the work of others. Törngren & Wikander (1996), propose that the lack of engineering methodology and tools is due to the multi-disciplinary nature of distributed control system design. Real-time control system design requires an understanding which spans the fields of electronic engineering, computer science, control theory, mechanical engineering and systems engineering - it is a truly architectural problem. Historically, research concerning distributed control systems has pertained to the communications

technologies, network scheduling, high-temperature electronics and software development rather than the system in its entirety. The following discussion indicates that this trend is beginning to change, yet a significant amount of system level research into distributed control system design and analysis remains outstanding. Törngren & Wikander's belief further underpins the need for a systems engineering approach to DCS design and development. Distributed systems are neither a technology nor a methodology, but a conglomeration of technologies which realise an architectural form. Distributed control systems as a generic entity are difficult to define and therefore, study. DCSs may have any number of nodes, be physically separated over any area and utilise an enumerable combination of network topologies, communications technologies, power supply systems and software applications. The term ''distributed system'' is broad and unless applied with context becomes ambiguous - it may be used to describe any number of systems from the internet, to group control of Unmanned Aerial Vehicles (UAVs), multi-media systems, industrial process control and remote electricity metering to robot and jet engine control. This research is concerned with real-time DCSs with fixed architecture, static-scheduling and pre-determined functional allocation.

3.2 Historical and technical perspective

When compared to technical disciplines such as power generation or industrial process engineering, distributed control system design is an immature and unexplored art. Perhaps the earliest published work of direct relevance to this research was a MIT PhD thesis (Mok, 1983) considering the problems of software and communication design in real time DCSs. The work proposed advancements to an abstract model for software design and scheduling for distributed control applications from textual requirements but did not consider the wider concerns of network or control system architecture. Mok's references contain no papers relating directly to distributed control system design.

Hermann Kopetz, Professor of Real-Time Systems at the Vienna University of Technology, is one of the few academics to have produced a substantial body of research concerning real-time distributed systems. Work from over 100 of Kopetz's papers cover many contrasting aspects of real-time DCS design from best practice at a systems level (Kopetz *et al.*, 1991a,b), to fault tolerance (Kopetz *et al.*, 2000) and the low-level design

considerations of clock synchronisation and chip design (Kopetz, 2007, 2008). His earlier work culminated in the Time Triggered Protocol (TTP) (Kopetz & Grunsteidl, 1993) of which he is the chief architect. TTP is a time-triggered communications protocol for embedded real time systems which has found application in automotive (TTTech, 2007a), civil aerospace (TTTech, 2004) and its ethernet based equivalent in manned space systems (TTTech, 2007b).

In 1997, Kopetz published a monograph of works related to the design of real-time distributed systems. Kopetz draws on his own work (Kopetz & Ochsenreiter, 1987; Kopetz, 1995) and the work of other researchers from software and control system design disciplines (see Locke, 1992; Lin & Herkert, 1996, for example). The resulting book (Kopetz, 1997) entitled, ''Real Time Systems: Design Principles for Embedded Applications'' has become a seminal text and despite its heritage, has not been superseded (2009). A later text (Pop *et al.*, 2004) considers distributed real-time embedded systems in a wider context and includes more detailed discussion on event-driven and multi-cluster systems yet adds little to the understanding of real-time DCS design.

Whilst Kopetz's work is technically comprehensive, the majority has been undertaken from the perspective of a real-time system's designer rather than a system architect. His work presents a multitude of design considerations and technologies that will greatly influence DCS architectures, yet he does not directly address the practice of architecting or designing value into such systems. Whilst his treatment of these technologies stems from an academic perspective, the problems addressed and techniques proposed were familiar to designers of legacy demonstrator systems such as ANTLE and HiPECS.

From the late 1980s onwards, a body of academic work relating to the technologies and control system techniques required for distributed control systems was published. Several academics investigated the effect of packet loss and communications jitter on the stability of closed loop control. Contributions include Chan & Ozguner (1995), Chow & Tipsuwan (2001), Chen *et al.* (2007) and Yedavalli *et al.* (2008). An insight into the the future of DCSs, the difficulties of implementation and application of wireless control of wide-area control problems is given by Lian *et al.* (2002). Whilst this work highlights many important design considerations and is valuable for the wider application of distributed control systems, relevance to this research is negligible. The techniques discussed are either outdated or unsuitable for the applications where absolute determinism is prerequisite.

Perhaps the most familiar application of distributed control systems originates from the automotive industry, where the CAN has been used extensively to permit communication between distributed processing elements located throughout the car. Work in this area has focused on increasing the determinism of the CAN network through the application of Time Triggered CAN (TTCAN) to ''by-wire'' systems. Papers from Short & Pont (2007) and Führer *et al.* (2000) address the issue of determinism and fault tolerance of TTCAN and the wider application to distributed control. However, the environment and nature of distribution in the jet engine is very different from automotive applications. The functionality of the jet engine control system is very tightly coupled and almost wholly dependant on real-time communications. Therefore, lessons learned in the automotive sector provide a stimulus for discussion rather than a direct application to jet engine control.

The internet, wide area networks and multi-cluster computing systems have spawned a new generation of distributed computing systems that evolved into their present form rather than definition through systematic design. A substantial body of academic literature and research programmes consider such networks for use in supercomputers and industrial control (see Tsai *et al.*, 1996, for example). Whilst many of these systems have hard real-time constraints, the level of determinism and the technologies which host them are inappropriate for this application.

3.3 Application to Jet Engine Control

It is noteworthy that both the Rolls-Royce HiPECS (1997) and ANTLE (2000) programmes preceded much of the systems level research focusing on DCS for gas turbine engines; both programmes were technically successful but failed deter Aero Engine Controls from commercial production of centralised systems. Despite their apparent novelty, the military and commercial sensitivity of these projects meant that very few academic papers were published. One paper, Watkinson (2003) presents the architecture used in the ANTLE programme and basic operation of the communications bus. HiPECS and ANTLE preceded the formation of the NASA Distributed Engine Control Working Group (DECWG). The DECWG was formed between the NASA Glenn Research Centre and North American industrial partners to examine current and future requirements of

propulsion systems. The group is primarily concerned with the development of distributed systems for military purposes; this is reflected in the group's industrial composition. The scope of their study includes an assessment of the paradigm shift from centralised engine control architecture to distributed systems using open-system standards (SAE, 2007). Since 2002, the group have published a number of technical papers which are discussed presently.

Culley *et al.* (2007b) present an overview of the concepts and problems involved in developing distributed architectures for gas turbine engines. The paper is written with a slant towards engines for military fighter-attack aircraft, but the vast majority of the content is directly applicable to distributed FADEC design for large civil platforms. The authors acknowledge that DCS design is a complex problem and relate that complexity to the ambiguous nature and number of architectural possibilities. They highlight the lack of concrete definition to describe DCSs. Having presented the benefits of distributed FADEC systems, the environmental constraints and costs are discussed. The paper is a preliminary attempt at summarising the basic technical and commercial issues involved in distributed system design. A further paper (Culley *et al.*, 2007a) authored by a broader contingent of the DECWG discusses the wider constraints to DCS and proposes that such systems will only become viable once industrial standards for the communications protocol and technology are realised. The group's paper recognises the complexity of the business case and the reluctance of commercial manufacturers to conform to industrial standards. Neither of these papers present the outcome of 'research' - their contribution is to formalise and substantiate the many industrial opportunities and concerns which surround distributed systems. Furthermore, whilst not proposing solutions, the authors recognise the complexity of the DCS design problem. That is not to undermine their importance to the field. In a subject area where the complexities are vast and the perspectives diverse, better understanding may be required before meaningful research directions are established.

Papers from the DECWG reference many earlier NASA papers concerned with the development of specific engine technologies related to DCSs. This is inevitable given the impact that system functionality and communications technology has on architectural design. Such references include Garg (2002) who discusses NASA's research into Engine Health Monitoring (EHM) systems. Subsequent papers include Volponi *et al.* (2004) and

Litt *et al.* (2005) which describe data fusion and engine intelligence respectively. EHM is likely to have a substantial impact on DCS design. Engine manufacturers have a thirst for diagnostic data to aid fault diagnosis and therefore, reduce lifecycle costs. As the algorithms and data required for EHM increase in complexity and size, distributed control systems offer the potential for the increase in processing power required to support them. Simon *et al.* (2004) discuss the requirements for novel sensors required to improve engine control and health monitoring capability. Their paper does not directly address distributed control systems but does present a list of future sensor needs and their specifications. Novel sensors will operate at bandwidths many orders of magnitude higher than present day systems. The need for higher-bandwidths and more localised control will inevitably require some form of distributed architecture for implementation. Distributed control systems must be designed with architectures that are sympathetic to future engine technology and expansion.

A later report from the DECWG (Culley & Behbahani, 2008) presents an overview of the communications technologies required to implement DCSs. The authors discuss the wider implications of the communications technology chosen - the effect on weight, acquisition cost and lifecycle cost are highlighted. Without making direct recommendation, many design considerations are presented. The authors attempt to quantify the amount of communicated data necessary to implement distributed control on a typical turbofan engine. Their estimates seem improbably low and account only for the most basic of system functions, yet such estimates hold indicative value. The paper references a comprehensive comparison of various communications technologies which many be used in future NASA aerospace applications. This study (Gwaltney & Briscoe, 2006) was conducted at the NASA Glenn Research Center and compares many different communications technologies suitable for Real-Time Distributed Control System (RTDCS) application.

A further overview of distributed control systems for gas turbines (Huang & Xu, 2003) entitled, ‘‘Distributed control systems for aeroengines: A survey’’ was published in Chinese and an English translation is currently unavailable.

Thompson *et al.* (1999b) present the first any only instance of academic literature considering architectural design of DCS for gas turbine control. The work was undertaken at the Rolls-Royce University Technology Centre (UTC) in Sheffield and uses a genetic algorithm to optimise the design of a distributed FADEC system. The authors consider

Distributed Full Authority Digital Engine Controller (DFADEC) design for a generic Advanced Short Take-off Vertical Landing (ASTOVL) aircraft such as the F-35 Joint Strike Fighter (in practice, the principles demonstrated are independent of application). The authors use the Multi-Objective Genetic Algorithm (MOGA) to locate smart units (nominally sensors and actuators) across the engine platform. The evaluation used is seemingly comprehensive - Each architecture is assessed for reliability, cost, component cost, failure rate and maintenance costs. The analysis permits different network topologies to be considered. Whilst the principles are demonstrated, the system does not consider many of the commercial nuances of DCS design and is insufficiently comprehensive for use in an industrial evaluation. Additionally, the research neglects the allocation of functionality to nodes or communications throughput. More fundamentally, the method does not constrain node allocation based on the engine environment - it highlights where improvements in technology may be required to realise the system but falls short of proposing distributed systems which meet present day technology constraints. It is noteworthy that despite the work's heritage, subsequent citations have been made by researchers interested in the application of MOGAs rather than distributed gas turbine control. This notion extends to all the academic texts covered in this section - these papers are rarely cited and very few academic authors have shown an interest in defining the design of DCS for jet engines or other comparable systems.

3.4 Research Challenges in Distributed System Design

Discussion with industrial engineers and wider academic reading have highlighted five areas where significant progress is required before DCSs become commercially viable. The literature reviewed in the following five areas is intended to provide the technical perspective required of a system architect:

- Architectural Design of Distributed Systems (including fault tolerance)
- Provision and Integration of HiTEDs and other device considerations
- Whole system simulation and evaluation of DCSs
- Performance and cost effectiveness of communications technology
- Predicting, measuring and evaluating the lifecycle value of DCSs

These considerations are not an exclusive set - there are many commercial forces and precedents which must be overcome, yet they cover areas where academic research

may contribute to furthering the understanding of DCS technology. Relevant academic contributions to these research challenges are discussed presently. Architectural design and system simulation are discussed together.

3.4.1 Architectural Design and Whole System Simulation of Distributed Control Systems

The multidisciplinary nature of DCS designs makes them very difficult to architect and simulate in their entirety. The cost savings, performance and risk analysis offered by modern software packages has become a fundamental part of modern industrial engineering practice. Software packages are available to support the simulation of many different technical systems, but not distributed systems in their entirety.

Törngren & Wikander (1996) recognise the design of distributed systems as an optimisation problem and present a method for control system decentralisation derived from functional analysis. Their method involves determining the input/output (IO) bounds (effectively functional coupling) of system functions to isolate the elementary functions which are subsequently allocated to hardware based a number of optimisation criteria. Whilst Törngren & Wikander's method is based on a sound systems approach, it relies upon there being only a small number of permissable hardware architectures and does not take into account environmental constraints or lifecycle value.

Few authors have addressed design frameworks for RTDCS. Törngren & Torin (1998) discuss the conceptual design of distributed systems and the need for an enhanced focus during the design phase. They identify six key phases in the design of RTDCSs including structuring and allocation of functionality to the underlying hardware platform. Törngren & Torin discuss how the multidisciplinary nature of DCSs has resulted in a deficit of tools able to model and simulate the behaviour of real-time distributed systems. They describe the technical design considerations and the difficulties of formulating network schedules and modelling internode communication. This discussional paper was a precursor to developing a toolset (later known as Automatic Control in Distributed Applications (AIDA)) intended to support modeling and analysis of real time distributed systems. In a later more detailed paper (Törngren & Redell, 2000) the authors propose a framework for AIDA, which considers behavioural, structural and temporal models

of the distributed system. This is perhaps the first published application of a broader approach to distributed control system design, although the inspiration stems from best practice in embedded system design rather than systems engineering. Approaches and tools for modelling the control application, computing and mechanical aspects of DCSs are discussed and compared. El-Khoury & Törngren (2001) presents some of the basic MATLAB/SIMULINK® models used within the AIDA toolset to the analysis of control system performance in the presence of temporal uncertainty. The completed toolset was published by Redell *et al.* (2004).

Many of the design considerations identified by Törngren *et al.* whilst developing AIDA reflect genuine concerns of engineers considering distributed systems for aero-engine control. However, the approaches proposed assume that system architectures are derived from the underlying control laws and system functionality rather than non-technical requirements. Whilst such an approach may offer technical superiority, the realities and constraints of the commercial aerospace market mean architectures derived in this way are unlikely to produce the necessary financial rewards. In a commercial context, architectural changes are required to lower lifecycle costs and increase design efficiency in preference to technical superiority. The authors do not consider value or the ability to produce and sell the system.

Fault Tolerance in DCSs

An area of distributed control system design that has received wider research interest is that of fault tolerant nodes. Brasileiro *et al.* (1996) present a method for replica synchronisation using a software based comparison of system outputs. Authors have given particular credence to the development and use of distributed recovery blocks in node design. Recovery blocks use both hardware and software techniques to permit the failed element of a dual redundant node to resume proper operation using information from the fully functioning element (see Kim & Welch, 1989, for detailed explanation). As described by Randell & Xu (1994), the recovery block concept has evolved over many years and despite the obvious benefits has failed to gain industrial acceptance. The author believes that this may be due to the lack of an industrial strength proof of concept rather than technical inferiority. Hecht *et al.* (1991) gives an example of a recovery block for a nuclear reactor application.

3.4.2 High-Temperature Electronics and Other Device Considerations

The present unavailability of HiTEDs is one of the few hard barriers to DCS implementation. Conventional military specification semiconductor and passive devices cannot survive in hostile environment of the compressor, combustion and turbine stages where SATs may reach 800°C. Present day engine electronics are located on the engine fancase where SATs are considerably lower. The importance of HiTEDs is perhaps overemphasised as the solitary barrier to DCS implementation, yet such systems will not be realised without electronic devices capable of withstanding the engine environment for the duration of a twenty-five to thirty year service life. The potential uses for HiTEDs in aerospace applications are well understood by industry but highlighted in academic literature by Johnston *et al.* (2000). The aerospace industry is far from the only sector interested in HiTEDs - papers such as Johnson *et al.* (2004) highlight increasing demand in the automotive industry where engineers are investigating the possibility of embedding electronics within the engine chassis and exhaust stream. Johnson *et al.* suggests that like the aerospace industry, the integration of HiTEDs has been stifled by a conservatism which promotes reliability and cost reduction over novelty and performance. Additionally, the Oil and Gas industries have a great interest in furthering HiTED technology for both resource discovery and drilling devices (see Ohme *et al.*, 2006, for example). Here the business case is substantially different as drilling components may be considered expendable.

Lande (1999) discusses the potential applications of HiTEDs and their use in the aerospace, automotive and oil and gas industries. His paper briefly discusses the supply and demand problem which is keenly felt by industry. Although there are many interested parties, the volume of sales and likely uptake are insufficient to lure device manufacturers into self-funded research and the mass production necessary to drive down costs. The market situation differs little from that seen in high-temperature polymer production (discussed by Hergenrother (2003)). High temperature polymers which have potential for plastic casing and mounting items for DCSs are a considerable way from commercial viability.

According to Hornberger *et al.* (2004) SiC devices hold the greatest potential for both high-temperature processing and power electronic devices. Obviously, the work required to develop these devices has its origins in theoretical physics and electronic engineering

design. The specifics of these disciplines lie outside the scope of a systems study. Papers such as Flandre (1995) and Flandre *et al.* (2001) discuss some of the more technical aspects of HiTED design and implementation. Alongside research into new devices, manufactures are keen to improve thermal management within their devices to increase component operating life and allow a greater number of devices to be co-located. A discussion into this research area is provided by McGlen *et al.* (2004). Devices used in engine control must not only withstand extremely high temperatures, but neutron bombardment. Discussion regarding 'latch-up' (whereby a logic gate's status becomes changes or fixed following a neutron collision) and the status of suitably qualified integrated circuits are provided by Layton *et al.* (2003) and Page Jr *et al.* (2005) respectively.

3.4.3 Communications Technologies

Developing the real-time communications technology to the necessary specification and cost has presented a surprising technical challenge. Absolute determinism is required for safety-critical real-time systems. Such a standard of communication is not achieved without computational complexity or financial cost. Communication technologies and the effect of non-determinism on control system performance have received wide coverage from academic researchers. Two overview papers on networked control systems (Antsaklis & Baillieul, 2004; Baillieul & Antsaklis, 2007) dedicate much of their content to discussing communication constraints and the problems which must be overcome to achieve desirable control system performance. Many of the papers discussed previously in section 3.2 are concerned with the development and analysis of communication technologies.

Until the advent of TTP, CAN was favoured for real-time distributed networks. Torngren (1995) presents a review of CAN technology and the considerations which influence distributed system design. A later paper from the Rolls-Royce UTC in Sheffield (Thompson *et al.*, 1999a) considers the use of CAN networks for distributed jet engine control. Many of the technologies and techniques considered are now obsolete. CAN was used as the communications bus for ANTLE and HiPECS, but failed to gain favour with Rolls-Royce.

The Time Triggered Protocol was specifically designed for use in RTDCS and stemmed from the aforementioned work of Hermann Kopetz. The protocol and associated hardware

is specially designed to overcome the weakness of communication technologies such as CAN and provide the very highest levels of determinism. The protocol has found academic favour and various facets of the technology such as clique avoidance, fault handling and TDMA synchronisation are considered in papers such as Lonn (1999) and Bauer & Paulitsch (2000). Papers and articles such as Jakovljevic (2006), Jakovljevic *et al.* (2006) and Samuel *et al.* (2007) as well as cases previously mentioned in section 3.2 show that TTP has been used in a number of aerospace applications. Despite it's technical merits, the cost of licences, hardware and lack of implementation experience makes the industry reticent of the technology.

3.4.4 Lifecycle Modelling

Developing a business case for new and emerging technology is notoriously difficult. The costs of developing technology are often well understood - indeed, the cost to the system developer is often the cost of undertaking a process driven development rather than the cost of a specific technology. However, the most substantial costs and ultimately those which dictate a system's viability, are those which arise over the duration of the product lifecycle. For commercial realisation, the lifecycle value of a system must not only prove viable, but provide sufficient value above and beyond existing systems, that the risk and costs of development are commercially palatable.

A small but significant body of work from the systems engineering community has attempted to realise an academic perspective on product lifecycle analysis. Fricke & Schulz (2005) takes an academic look at design and a later paper (Browning & Honour, 2008) at measurement of lifecycle value for products with long service lives. Both papers describe principles well understood by industry. Engel & Browning (2008) relate lifecycle value to the architecture of a system and discuss methods of measuring lifecycle value based on a product's composition. Their work is very high-level and deals principally with large-scale socio-technical systems rather than products akin to engine control systems.

3.5 Positioning the Study

This section combines the knowledge gained in the background chapters to position the study and substantiate the research objectives stated in section 1.4. Various constraints

and the concerns of Aero Engine Controls are considered in the process of selecting the specific areas for the research and the ensuing methods.

3.5.1 Factors influencing the research method

There are five important notions drawn from the background information and literature which significantly influence the choice of research method. Broadly, they are concerned with the aims of the research and the gaps in both industrial and academic understanding:

- Aero Engine Controls has demonstrated a distributed control systems with three nodes. Despite the system's technical success and effectiveness on engine, the implementation did not yield a system considered to be commercially viable nor convincingly realise the touted benefits.
- Aero Engine Controls has stipulated that this research have a broad focus and approach distributed control system design from a systems perspective. A declared aim of the research is to produce candidate architectures with the potential to realise the benefits associated with DCSs.
- To date, academic research into DCSs has focused primarily on the constituent technologies rather than architectural techniques and design guidance.
- Most academic literature on DCSs for gas turbine engines neglects or substantially underestimates the commercial constraints that ultimately define a system's viability.
- In accordance with the degree regulations, the research should make both an academic and industrial contribution.

3.5.2 Research Objectives

Further to the stated research aims, the research should provide:

- A platform independent solution to the DCS design problem permitting application to a variety of technology platforms. Traits of portability include scalability, flexibility and technology independence. Accordingly, the solution should be relevant beyond Aero Engine Controls, large, civil jet engines and hold applicability once present day electronics and control technology has been superseded.

- Proactive, not reactive solutions: the research seeks to deliver optimal distributed architectures by design rather than by disbursal of centralised systems or analysis of existing or postulated distributed designs.
- A solution which addresses the broadest range of systems engineering concerns possible - or could be extended to do so with further research or development.

Further to the notions stated above, there are a number of stabile constraints which limited the variety of tennable approaches:

- DCS designs could not be built or functionally tested. No demonstrator platform is available and it is inconceivable that control system hardware would be fabricated to meet the needs of this research.
- Information and raw data relating to the target engines and operation of the companies involved is commercially sensitive and therefore, difficult to obtain or publish.
- Given the lack of tools and techniques to analyse distributed architectures, validation of any DCS would prove extremely challenging or impossible.
- There are a lack of established heuristics and guidance concerned with distributed architecture. Demonstrator programmes such as ANTLE acquired their architectures by ‘‘engineering judgement’’ rather than theoretical concern for optimality.
- This research is the work of an individual researcher and is not part of a wider collaboration. Whilst Aero Engine Controls employees and supporting academics provide stimuli, information and analysis, their role in the research is non-participatory. Funding for practical aspects of the research is constrained.

The qualifications given in section 1.4 define the objectives and boundaries which limit the choice of hypothesis and method. Several approaches to fulfilling the research aims were proposed. A mind-map of the alternatives is given below in figure 3.5.1. It was believed that a research hypothesis could be devised around each alternative. Ultimately, the use of computational techniques to design optimal distributed systems was chosen.

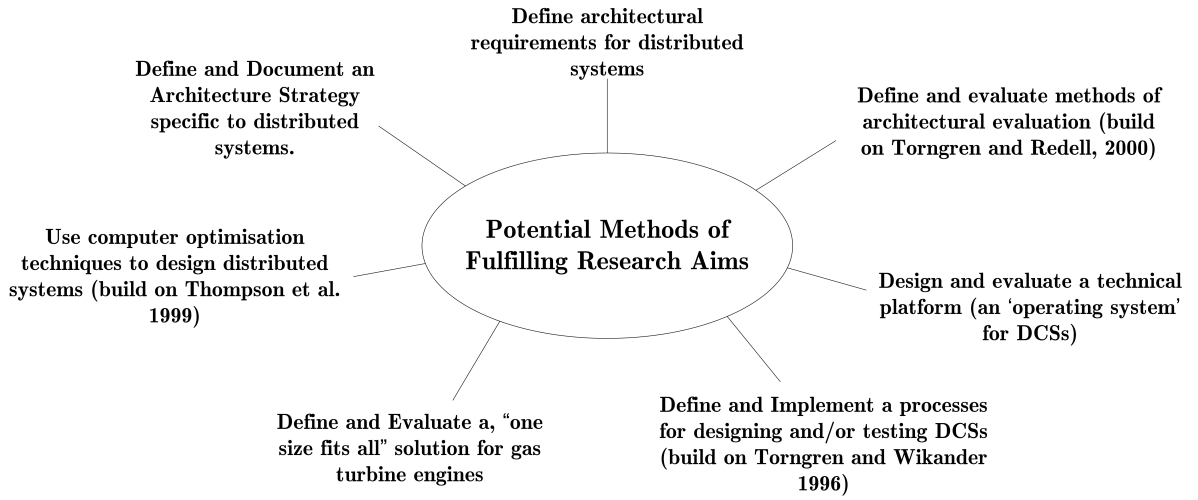


Figure 3.5.1: Potential methods for fulfilling research aims

The use of MOGAs by Thompson et al. (1999b) to generate DCS designs acknowledges a general consensus that architectural design is a complex optimisation problem. This research will aim to build on this work and introduce a clearer systems engineering perspective to DCS design - Thompson et al's work has several (previously discussed) limitations which this research will aim to overcome. Most notably, by acknowledging the wider business case and the present day limitations on devices capable of withstanding the harsh engine environment. The research will aim to provide a structure for the architectural optimisation of complex systems which could equally be applied to products beyond jet engine control systems.

3.6 Summary

Having considered the research challenges, the foundations of this research project and the commercial needs of Aero Engine Controls, this research endeavours to further the understanding of the architectural design of DCSs from a systems engineering perspective. In doing so, it will address the concerns of Aero Engine Controls as a potential DCS developer.

Chapter 4

Optimisation Algorithms

4.1 Introduction

The vast number of architectural choices and decisions lend this optimisation problem to resolution by a multi-objective optimisation algorithm, or more specifically, a Genetic Algorithm (GA). This chapter considers the nature of the optimisation task and justifies the use of a GA and the specific algorithm chosen. The chapter considers the design decisions necessary to select the algorithm and shows how the structure of the algorithm is used in this research. It is shown that the choice of algorithm is dictated by both the nature of the problem and the specific GA's effectiveness.

Part II presents a more holistic perspective on the research method and details the structure of the optimisation scheme and the techniques by which the Distributed Control Systems (DCSs) are built and evaluated. In the context of this research, the GA is considered a tool - the chosen GA is a widely established, well researched and heavily documented; there is no intention to contribute novelty to the to the academic field of GA design (beyond increasing the wealth of application examples). Accordingly, the discourse and evaluations presented herein provide the reader with a high-level summary. Readers seeking more exhaustive explanations may refer to Haupt (2004) and Deb (2001).

4.2 Selecting an Optimisation Technique

Mathematics provides a multitude of methods for determining solutions to optimisation problems. The most widely recognised techniques involve finding maxima and minima

using calculus and attendant mathematical programming techniques such as gradient descent. However, calculus based methods rely on bounded, continuous functions to describe the given search space. Furthermore, they cannot be used to solve multi-objective problems. System architecture and hence the design of DCS do not lend themselves to description in these terms and cannot be afforded the resources required by inherently inefficient greedy algorithms. From hereon-in, we will consider only multi-objective approaches. The term GA implies a multi-objective generic algorithm. Important facets of the DCS optimisation problem are listed below:

Non-Linear Complex system architectures are usually non-linear - the components which constitute the system itself are rarely linear in their structure, operation or physical arrangement.

Discontinuous System architectures are discrete in form. Architectures are composed of discrete instances of specific objects and are likely to be evaluated by discrete means. For example, an electronic system may have two power supplies and accordingly be recognised as dual redundant (true or false). Here, both the composition of the system and its evaluation are in discrete terms.

Multi-objective Systems engineers are often concerned with trade-offs influencing many stakeholders with vastly different viewpoints - common trade-offs include weight, reliability and cost. Where the tradeoffs between different objectives are not understood, a multi-objective algorithm is necessary. The optimisation must operate in many dimensions in both the design and objective spaces. An optimisation routine would be overtly limited if it were able to optimise only a single parameter.

Many Permutations The number of permutations by which hardware elements may be arranged and mounted on the engine is seemingly infinite.

When considered in combination, the properties listed above engender a further consideration.

Computationally Expensive Whatever the method of optimisation chosen, intuition suggests that it would impose a considerable computational burden.

The characteristics listed above form a basic specification for the optimisation algorithm; it must be capable of handling non-linear, discontinuous, multi-dimensional problems and analysing vast numbers of permutations in a computationally efficient manner.

Papers comparing multi-objective optimisation techniques and their applications (Marler & Arora (2004); Coello (1999); Coello *et al.* (2006) for example) highlight two classes of algorithm capable of addressing such problems - Evolutionary Algorithms (EAs) and Particle Swarm Optimisation (PSO) - computational optimisation algorithms stemming from the field of artificial intelligence and machine learning. Whilst both algorithms are a product of the computational, meta-heuristic age, the advent of PSO is comparatively recent having first been demonstrated in 1995 (Eberhart & Kennedy, 2002).

''Evolutionary Algorithms are important and established tools for engineers and scientists approaching complex optimisation problems'' (Deb, 2001). EAs and hence Multi-Objective Evolutionary Algorithms (MOEAs), use computational mechanisms based on or inspired by natural evolutionary processes such as inheritance, mutation and natural selection. Their fundamental operation applies these processes to generate solutions, disregard those considered inferior and use those considered favourable as the basis for generating increasingly favourable solutions. Four different approaches constitute the family of EAs (Fleming & Purshouse, 2001):

1. Genetic Algorithms
2. Genetic Programming
3. Evolutionary Programming
4. Evolutionary Strategy

Genetic algorithms (first proposed by Holland (1975)) are perhaps the most widely known of the approaches and considered in more detail presently. Genetic Programming and Evolutionary Programming (Walker (2001) and Fogel & Fogel (1996) provide introductory papers covering the respective subjects) are related techniques designed to evolve computer programmes to fulfil pre-determined requirements and thus automate the coding process; neither technique lends itself to this optimisation when applied in their canonical form. Evolutionary strategy is another related technique more similar to the GA, but best suited to problems where solutions are represented by real numbers. Fleming & Purshouse (2001) note that the boundaries between the four approaches have been blurred

as scientists and engineers combine the most desirable traits of each algorithm to suit their particular problem. More recently, practitioners have combined evolutionary algorithms with game theory when tackling non-corporative optimisation problems. Périaux *et al.* (2001) gives one such example.

PSO may be considered as a member of the EAs family although the method's comparative novelty and differences in operating principle help maintain the distinction. As the name suggests, PSO mimics the flight of insects (particles) working as a swarm to achieve a common goal; the swarm members 'fly' within and 'explore' the search space. Rather than using genetic operators, the individuals adjust their, ''flight path'' according to their own experience and that bought to bear by other swarm members (Eberhart & Shi, 1998). Like the GA, PSO is an iterative, meta-heuristic method capable of working within n-dimensional decision spaces. In PSO, the 'velocity' of each particle is adjusted by changing its 'weight inertia'. This parameter determines how far a swarm member can 'fly' around the search space and thus, how far the swarm member can deviate from a given position. The particle velocities of the present generation are influenced by the experiences of the previous generations. This mechanism provides a means of controlling the balance between further exploration of local optima and permitting particles to explore the global space.

Choosing between the two methods is challenging. Various authors (Eberhart & Shi (1998); Hassan *et al.* (2005)) have published papers comparing the success of the two approaches on widely established test problems. Whilst such work provides useful guidance, both authors acknowledge that the disparities in performance are dependent on the nature of the problem in hand. Both papers promote PSO but do not dismiss GAs, nor argue that PSO is ubiquitously superior. Poli (2008) provides a clear and concise categorisation of over 700 papers citing the application of PSO to real world problems - the review gives useful context, yet the applications identified are broadly similar to those practiced on GAs (Gen & Cheng (1997); Fleming & Purshouse (2001)). A review of GAs and PSO performance across a breadth of applications would prove a consuming task and is beyond the scope of this research. By considering a subset of papers comparing the application of both approaches (Panda & Padhy (2008); Lee *et al.* (2004); Ou & Lin (2006); Gao *et al.* (n.d.)) and those papers referenced above, the author concludes the following:

- Both GAs and PSO have proven effective when applied to test problems and real world applications. Both are capable of repeatedly finding optimal solutions (where optimal solutions are known) to complex, non-linear problems.
- Both algorithms boast a similar breadth of applications
- Both algorithms operate effectively on n-dimensional search spaces
- Applications of GAs are more numerous than for PSO but this is likely due to heritage
- PSO algorithms are generally less complex than for GAs (especially for the more sophisticated GAs)
- GAs are more computationally expensive than PSO (contradicted by Panda & Padhy (2008))
- For the single objective case, PSO often returns a greater breadth of candidate solutions and does not converge to a single solution. GAs are more likely to locate an optimal solution and guide the entire population to converge towards it
- GAs are equally or more effective when applied to problems that cannot be described using real numbers.
- The conceptual complexity of the algorithms is broadly similar although the GAs are more familiar to both industrial and academic audiences.
- The suitability of each algorithm is very dependant on the nature of the problem. It may not be known which is more effective until both have been trailled.

In essence, there is very little to chose between the two approaches and consequentially the selection is influenced by factors beyond technical merit alone. Given that the optimisation technique is not itself a matter of this research, comparing multiple optimisation approaches is not a concern and a single technique must be chosen - The strengths of the GA, the number of variants available, the breadth of academic literature covering both theory, application and test problems, make the GA the chosen, if not undisputed optimisation solution for this task. Although not used within Aero Engine Controls, the identity of the GA is familiar to industrial practitioners and the fundamental concepts easy to convey to an industrial audience. Figure 4.2.1 shows the various options and decisions taken during method selection.

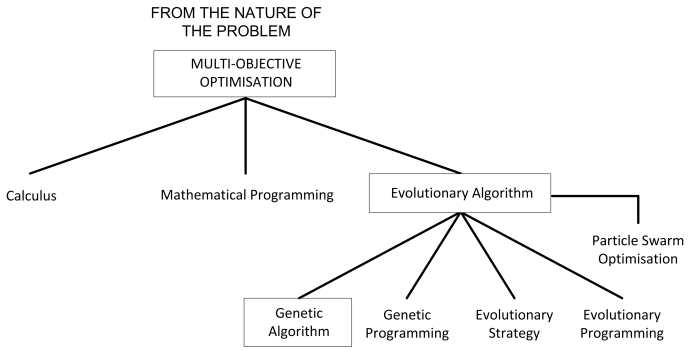


Figure 4.2.1: Options and decisions taken whilst selecting an appropriate multi-objective optimisation method

It is noteworthy that the choice of algorithm was made before the implications of the computational demands were fully understood. Whilst there is no compelling evidence that PSO would have provided better solutions or that the original decision was misguided, hindsight suggests that the generally reported reduced computational demand of PSO may have granted the algorithm favour. A further or improved implementation using PSO is therefore a matter for future research. Several authors including Settles & Soule (2005) and Esmin *et al.* (2006) have proposed and demonstrated hybrid algorithms that combine the swarm behaviour of PSO with the evolutionary concepts of GAs. This hybrid approach seems well suited to this optimisation problem, as the possible number of permutation is vast, but the number of good quality solutions likely to be small. It is conceivable that a PSO algorithm could narrow the search space and identify salutary regions before using the elitist GA to refine a smaller set of solutions.

4.3 General Form of the Genetic Algorithm

The diagram below shows a functional model of the GA as used in this research. The 'decode', 'construct' and 'evaluate' blocks will be referred to in the method (chapter 5). In essence, the method chapter and subsequent chapters discuss how these basic functions have been implemented to realise the overarching method.

The decode function translates the chromosome from a series of integer values into a blueprint for a DCS architecture. The construct function uses that blueprint to build a metaphysical model of the system. The evaluate function analyses that model from three perspectives: for architectural quality, lifecycle value and the quality of investment. In essence, each evaluation function provides a view of the architecture frameworks discussed

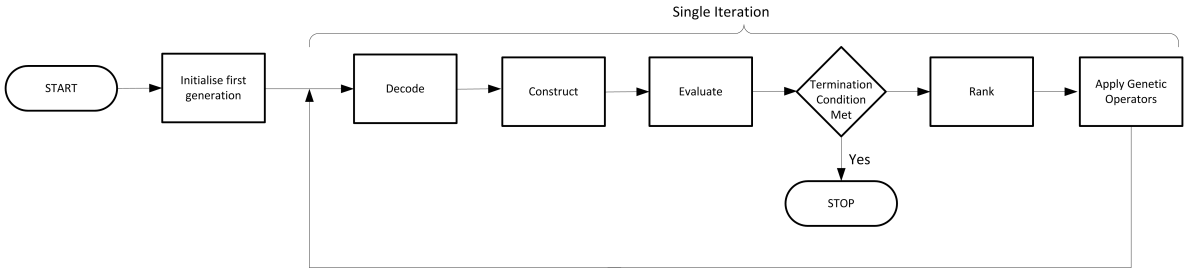


Figure 4.3.1: High Level block diagram of the GA as used in this research

in the next chapter. The genetic operators use the results from the evaluation functions to rank the solutions and produce the next population of chromosomes. The process is iterated until an optimality criterion is met, or a predetermined number of iterations has passed.

4.3.1 Selecting the appropriate GA

Following a high-level literature survey based on (Deb, 2001; Haupt, 2004), four candidate GAs were selected: Non-dominated Sorting Genetic Algorithm (NSGA-II), Strength Pareto Evolutionary Algorithm (SPEA2) and Multi-Objective Genetic Algorithm (MOGA). Each is eminently suited to this task yet differentiated by characteristic strengths and weaknesses. This section, discusses those characteristics and justifies the Strength Pareto Evolutionary Algorithm (SPEA2)'s application to this task.

The nature of the optimisation problem itself excludes many other available GAs; the algorithm must be capable of handling multi-objective problems, the problem may only be represented by discrete values and the number of possible permutations makes a random search untenable. It is preferable to use an elitist evolutionary algorithm to ensure that the best solutions are never lost and decrease the time to convergence. The effects of elitism both help and hinder the optimisation process. The preservation of elite solutions means that the best solution is never lost and that future offspring are based entirely on better solutions. Maintaining the evolutionary advantage means the algorithm should converge faster. The downside is the potential for solution-space niching and a higher probability of convergence to a non-optimal solution. A further advantage of an elitist algorithm is the ability to isolate a small subset of solutions rather than provide a large pareto optimal front. Choosing an elitist algorithm meant that MOGA was no longer a candidate. Figure 4.3.2 shows the decisions taken in choosing the appropriate GA.

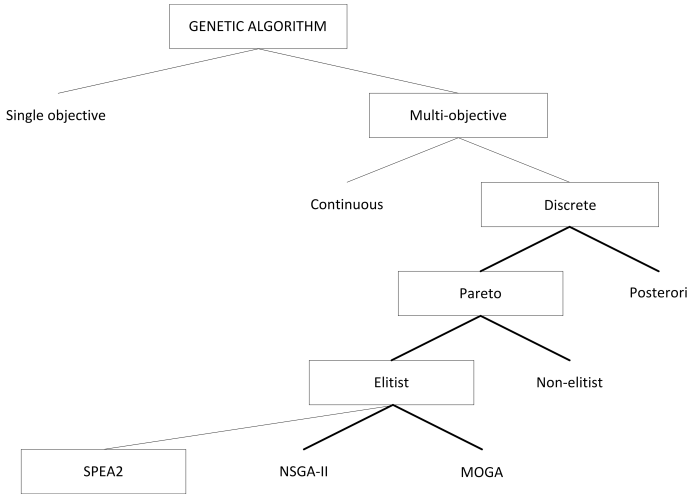


Figure 4.3.2: Process of choosing the appropriate GA

NSGA-II is perhaps the most documented of the approaches and various studies have shown the algorithm to outperform the other candidates on many different test problems.

The number of successful application examples and the algorithm's low computational overhead made NSGA-II the preliminary choice for this work. Only through testing and discussion with other researchers, were the algorithm's weakness in multidimensional objective spaces realised. NSGA-II performs exceptionally well on 2-dimensional problems - optimal solutions are obtained quickly and almost uniformly spaced across the pareto optimal front. SPEA2 found optimal solutions but was slower to execute and more prone to clustering. However, NSGA-II performed poorly on 3-dimensional problems with multiple decision variables - although optimal solutions were found, the diversity amongst solutions was very poor. Despite the speed deficit, SPEA2 provided both optimal and diverse solutions when tested on 3-dimensional objective spaces. Therefore, SPEA2 became the favoured algorithm.

The SPEA2 GA is an improved implementation of the original SPEA algorithm which better selects and retains diverse solutions. The original SPEA algorithm operates in a similar manner to MOGA.

4.4 Implementation

The operation of the SPEA2 is presented by Zitzler *et al.* (2001) and not covered in depth here. In short, the algorithm maintains two populations: an elite, 'Archive' (A) population

and a current population (P) of new solutions created by the genetic operators. The initial set of chromosomes is generated randomly and constitute random set of DCS architectures. The fitness of a solution is determined by the number of solutions it dominates (in the combined population $R = A \cup P$ and density function based on the k-th nearest neighbour algorithm. The density function aims to avoid objective space niching.

The algorithm has been coded specifically for this project and realised as three classes: a GA class, a solution class and a "GA_association" class. The GA class contains functions for non-dominated sorting, selection, crossover, density calculation and fitness assignment. The GA class could be used to apply the SPEA2 algorithm to any problem without modification. The solution class contains the chromosome for each solution and the functions (or function calls to) construct, decode and evaluate the solutions. The solution class is almost entirely application specific and holds the relational matrices used to compose and analyse the systems *(see section 5.5)*. An instantiation of the solution object exists for every solution in both the current and archive population. The GA_association class permits the GA and solution to interact. The class instantiates containers for the objective space values and provides functions for ranking solutions. The scheme is shown below:

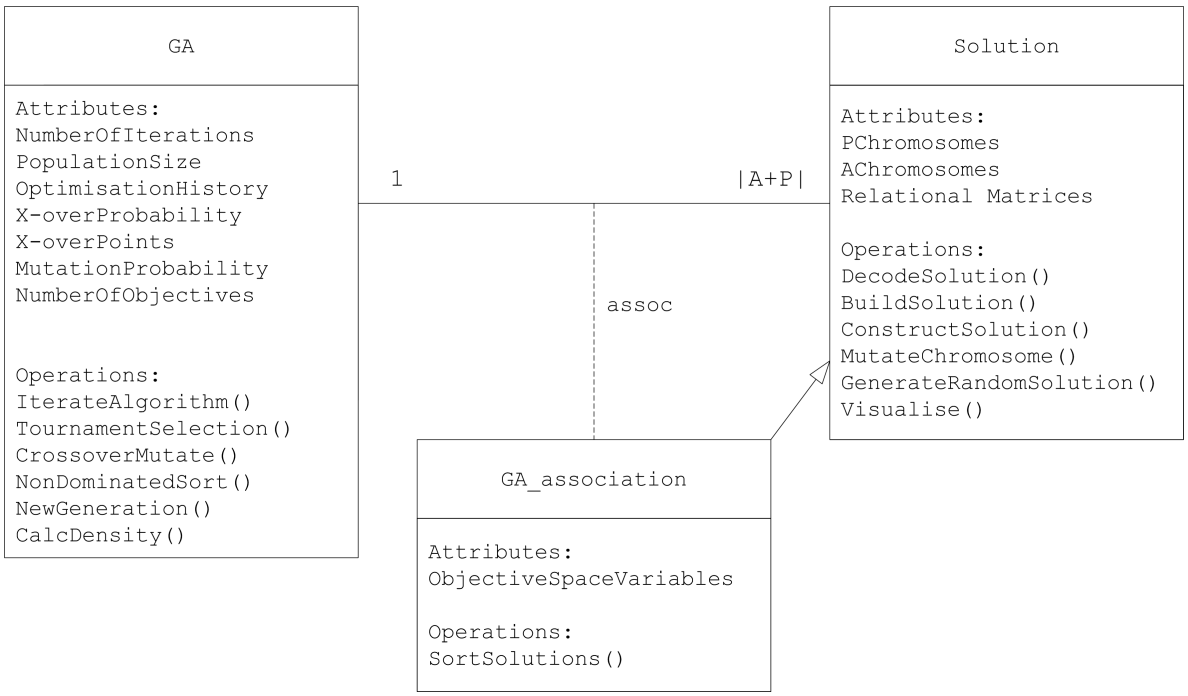


Figure 4.4.1: Object-oriented implementation of the SPEA2 algorithm

The entire algorithm is coded in object-oriented matlab. The `Visualise()` operation

of the solution class prepares the solution for viewing using the visualisation tool (section 5.13).

The GA uses multi-point, random permutation crossover to produce two offspring from any given number of crossover points. This technique permits small portions of chromosomes to be interchanged and brings about subtle mutations in long chromosomes. The chromosomes and construction process have been designed to avoid the generation of infeasible solutions and so additional functionality for constraint handling and objective-space violations are unnecessary.

4.4.1 Testing

It is practically impossible to verify the performance of the SPEA2 code on the DCS optimisation itself. Therefore, the algorithm was tested a number of standard test problems proposed by Deb *et al.* (2002). The problems are devised to test algorithms with non-linear functions exhibiting high sensitivity to parameter variation. The problems have multi-dimensional decisions spaces and 3-dimensional objective spaces to facilitate visualisation of the results. The implementation of the GA used for this optimisation was tested on two of Deb *et al.*'s test problems, DTLZ1 and DTLZ2. The test problems are designed to be scalable in both the decision and objective spaces. The equations given below are stated for 5 decision space variables and 3 objective space variables.

In each instance, $\boldsymbol{x} = [\ x_1, \ldots, x_k\ ]$ is a vector of decision variables of length k, where k is the number of decision variables. The tests used here have 5 decision-space variables and 3 objective space variables. Each decision variable takes a value 0 to 1. The decision values are calculated as the recipricol of a 14-bit binary coded integer. The vector $\boldsymbol{x}_M$ is a subset of $\boldsymbol{x}$ containing the final 3 values of $\boldsymbol{x}$. The SPEA2 algorithm is run for 350 iterations with a 99% crossover probability, 3 randomly generated crossover points and 2% mutation probability. The formula for the two test problems are given below:

DTLZ1

$$f_1(\boldsymbol{x}) = \frac{1}{2}x_1x_2(1+g(\boldsymbol{x}_M)) \tag{4.1}$$

$$f_2(\boldsymbol{x}) = \frac{1}{2}x_1(1-g(\boldsymbol{x}_2)(1+g(\boldsymbol{x}_M)) \tag{4.2}$$

$$f_3(\boldsymbol{x}) = \frac{1}{2}(1-g(\boldsymbol{x}_1))(1+g(\boldsymbol{x}_M)) \tag{4.3}$$

$$g(\boldsymbol{x}_M) = 10\left[|\boldsymbol{x}_M| + \sum_{i=1}^{|\boldsymbol{M}|}(x_i-0.5)^2 - \cos(20\pi(x_i-0.5))\right] \tag{4.4}$$

DTLZ2

$$f_1(\boldsymbol{x}) = [1+g(\boldsymbol{x}_M)]\cos\left(\frac{x_1\pi}{2}\right)\cos\left(\frac{x_2\pi}{2}\right) \tag{4.5}$$

$$f_2(\boldsymbol{x}) = [1+g(\boldsymbol{x}_M)]\cos\left(\frac{x_1\pi}{2}\right)\sin\left(\frac{x_2\pi}{2}\right) \tag{4.6}$$

$$f_3(\boldsymbol{x}) = [1+g(\boldsymbol{x}_M)]\sin\left(\frac{x_1\pi}{2}\right) \tag{4.7}$$

$$g(\boldsymbol{x}_M) = \sum_{i=1}^{|\boldsymbol{M}|}(x_i-0.5)^2 \tag{4.8}$$

In both cases, optimal solutions are found where all vales of $\boldsymbol{x}_M$ are 0.5. As shown by the plots below, the algorithm converges to both pareto optimal fronts indicated by the solid line in figure 4.4.2 and the spherical mesh in figure 4.4.3. The diversity amongst solutions is comparable with results from the original paper by Zitzler *et al.* (2001) although some niching has occurred.

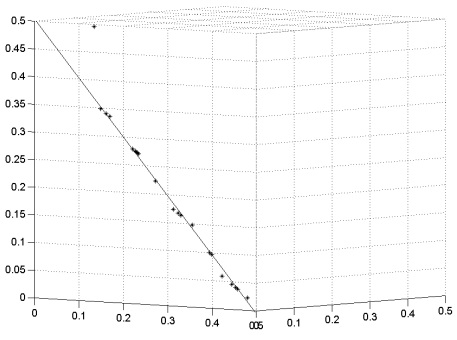


Figure 4.4.2: The SPEA2 algorithm tested on DTLZ1

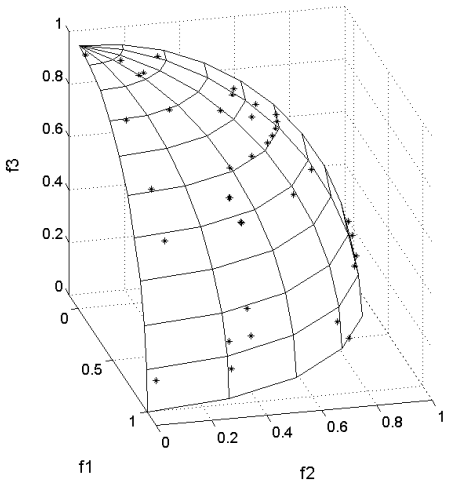


Figure 4.4.3: The SPEA2 algorithm tested on DTLZ2

Whilst these results do not provide a comprehensive analysis of the algorithm's performance, they demonstrate its proper function and give confidence that the algorithm executes as required.

4.5 Conclusion

The preceding discussion has justified the use of a GA and in particular, the SPEA2 variant. The algorithm itself is more computationally expensive than other similar algorithms but proven to provide a diverse set of solutions from multi-dimensional decision and objective spaces. Particle Swarm Optimisation is another technique worthy of consideration for future implementations. The SPEA2 algorithm has been written and implemented as three MATLAB® classes. The algorithm has been tested and proven to operate effectively on canonical test problems with known pareto-optimal fronts.

Part II

Method

Chapter 5

Research Method

5.1 Introduction

This research uses the Strength Pareto Evolutionary Algorithm (SPEA2) to automatically design and optimise Distributed Control System (DCS) architectures. The previous chapter detailed the development of the Genetic Algorithm (GA) and demonstrated its suitability for tackling complex, non-linear, multi-dimensional problems. This chapter focuses on the processes required to construct the candidate Distributed Control Systems - it examines the implementation of the the, 'decode' and 'construct' functions as applied to this optimisation. The structure of the evaluation functions is presented at the end of this chapter. The subsequent chapters 6, 7 and 8 describe the three evaluation functions in greater detail and present results which verify their function.

5.2 Overview of the Method

The optimisation process commences by generating a set of random chromosomes representing DCS architectures *(see figure 5.2.1)*. Each solution becomes a member of the GA's current population and constitutes a coded string of integer values. Each solution in the current population is decoded and subsequently 'built' to realise a metaphysical model of a DCS. The model includes nodes, harnesses and electronic hardware. This metaphysical control system is 'mounted' on a metaphysical engine with both physical and functional architecture. The engine model remains unchanged for the duration of the optimisation process.

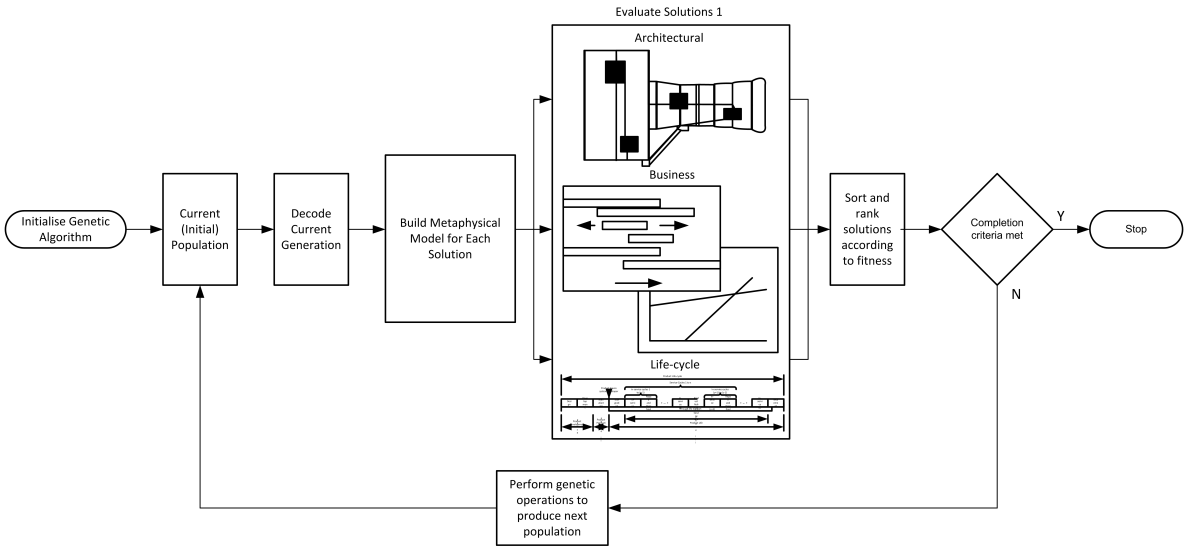


Figure 5.2.1: Synoptic diagram of the optimisation scheme

Once built, each DCS is evaluated using three separate evaluation functions. The first examines the architectural quality of the control system by considering facets such as weight, size and the length of harnessing required. The second uses a Monte Carlo Simulation (MCS) to calculate the lifecycle costs and disruption associated with operating the DCS over a thirty year service life. Finally, a business evaluation function uses the measures from the architectural and lifecycle evaluations, to determine the costs of production and ownership.

Depending on the configuration, each evaluation function produces either a single score or number of scores that the GA uses to rank the solutions. The GA applies genetic operators to produce the next generation of solutions and ensures that elite solutions are maintained in an archive. The optimisation iterates until either optimality criteria are met or a maximum number of iterations has passed.

5.3 The Optimisation Process

Not all the information required to build the architectures is coded within the chromosome - the chromosome provides a basic structure from which the remaining constructs are derived. The ancillary information required to build the complete models is read from a relational database prior to the optimisation starting.

The chromosome encodes the following architectural decisions:

- Node locations
- Hardware (Common Circuit Blocks (CCBs)) contained in each node
- Tasks preformed by each node
- Data network topology
- Power system topology

The metaphysical model is a complete (w.r.t the scope of this research), tangible model of a DCS. This research considers a 'DCS architecture' to include the following physical elements and attributes:

- Physical elements
  - Set of nodes
  - Physical dimensions and weight of each node
  - Location of nodes on the engine chassis
  - Electronic hardware (CCBs) contained within each node
  - Connectors on each node, their size and weight
  - Harnesses sizes and routes for connections between engine components and nodes
  - Harnesses for data and power networks
- Non-physical elements
  - Control systems tasks performed by each node
  - Signals sent between nodes

A formal logical definition of this architecture is given in section 5.4.7. The model may be viewed using a basic visualisation tool designed for this research *(see section 5.13)*.

The algorithm starts by reading the information stored in the relational database. The database holds data which defines the structure of the engine, the CCB specifications, a list of control system tasks and the parameters required to action them.

As the data is read from the database, so the metaphysical model of the engine is constructed. Details of the database structure and implementation are given in Appendix A The engine model includes the engine dimensions and environment, in addition to the locations and size of sensors, actuators and subsystems on the engine chassis. All elements of the engine model are considered independent of the distributed Electronic Engine Controller (EEC). The engine and its structure are unaffected by the decisions of the GA. Once the process of building the engine model and reading the configuration information is complete, the database connection is closed and the optimisation process started. The database can hold model data for many different engines, although the

optimiser considers only one engine at a time. The engine environment is recorded in matrices holding temperature and vibration data. Models relating connector size and weight, and reliability and temperature are required during construction.

The user must specifies which engine the optimisation is for and which configuration of the GA is required.

The number of nodes, η, remains constant for the duration of a single optimisation pass. Fixing the number of nodes is necessary to allow the algorithm to converge quickly and avoid the complication of dynamically sized chromosomes (section 5.7). The optimal number of nodes is established by executing the algorithm iteratively with the number of nodes η increasing from a lower value, η_{min} to a upper value, η_{max} on every pass (where $\eta_{min}, \eta_{max} \in (\mathbb{Z} \geq 1)$). The progression in node number may simply increment from η_{min} to η_{max} or be a pre-determined vector of node numbers *eg.* $\eta = [3\ 7\ 10]$. The Pareto-optimal set of solutions for each number of nodes J_η is combined to form a set, J. On completion of all optimisation passes, the combined set is subjected to non-dominated sorting and ranking to reveal the set of global pareto-optimal solutions, J^*. The results pertain only to a specific engine. The pseudocode for a complete optimisation is given in algorithm 5.3.1 and diagrammed in figure 5.3.2. Figure figure 5.3.2 is an expansion of the, ‘‘OPTIMISATION’’ block in figure 5.3.1.

Algorithm 5.3.1: SystemOptimisation(η_{min}, η_{max})

$J = \emptyset$

for $\eta \leftarrow \eta_{min}$ **to** η_{max}

$\begin{cases} J_\eta = \text{NodeOptimisationGA}(\eta) \\ J = J \cup J_\eta \end{cases}$

$J^* = \text{Front}(J)$

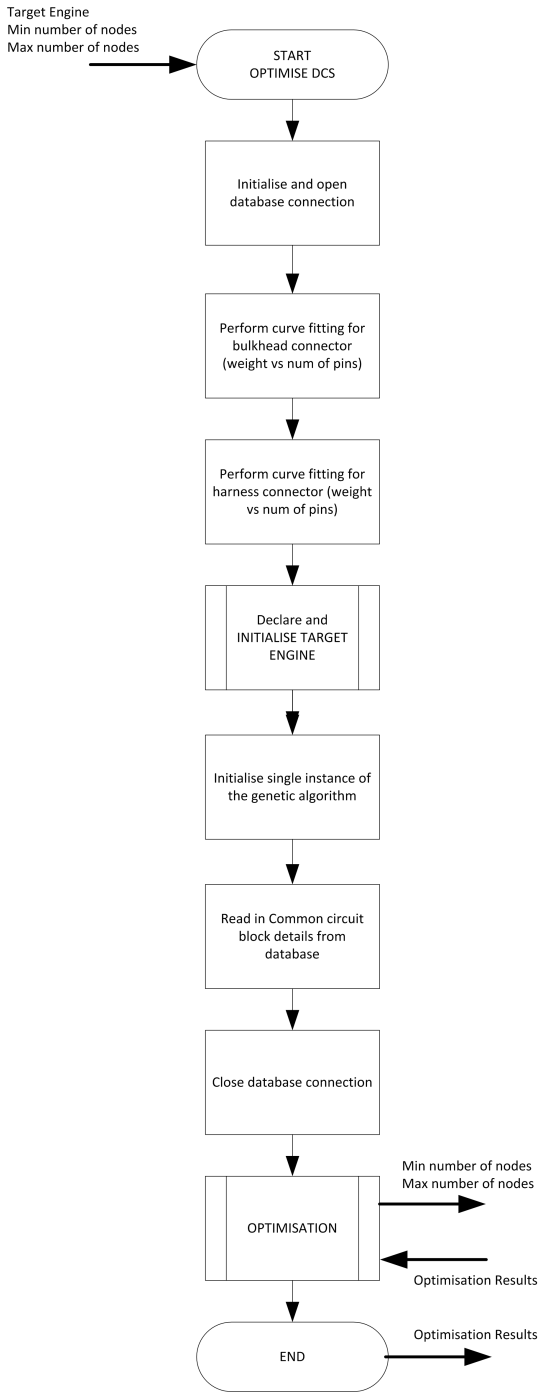


Figure 5.3.1: Flow diagram showing the high level initialisation and model building process

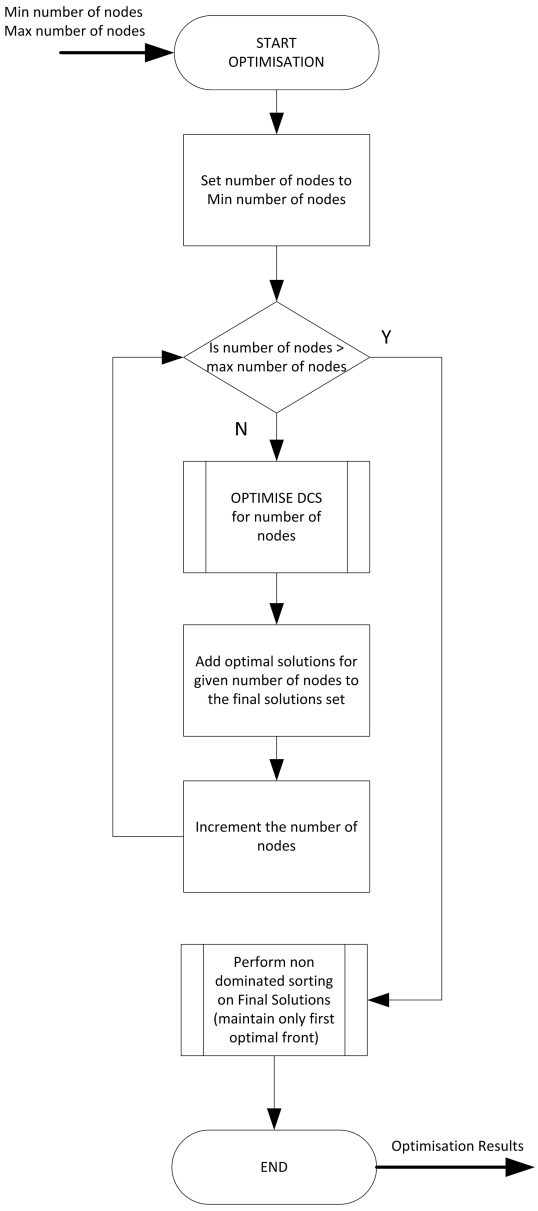


Figure 5.3.2: The global optimal set of DCS solutions is built by considering a single number of nodes in each iteration

5.4 Metaphysical Modelling Approach for the Engine and DCS

The metaphysical models of the engine and DCSs are based on a set of primitive architecture frameworks. The frameworks were devised for this research and draw inspiration from (but do not wholly conform to) ISO1471 (IEEE, 2000). The framework defines the model entities and the relationships between them. The frameworks define a single 'structural' view assumed from the viewpoint of the system architect. Whilst no other views are formerly defined, it could be argued that the three evaluation functions offer a technical, lifecycle and business view of the framework. Both the engine and DCS frameworks are static; there are no behaviours or methods associated with the final designs. All engines and DCS architectures considered during the optimisation process adhere to the architecture framework.

The architecture of the engine remains unchanged for the duration of the optimisation process, whereas the architecture of the DCSs is determined by the GA. It will be shown in section 5.5 how a system of binary relations is used to construct and evaluate each architecture - this system is based on the structure of the architecture frameworks. Although only one engine is used in this research, the optimisation process has been designed to work on many different engines and could easily be extended to work across product family or range. All engines share the same framework but could be differentiated by their dimensions and environments.

Set Notation

The entities and relationships in the frameworks are recorded in a series of relational sets and matrices. In this document the notation for the sets is introduced after the definition of the framework. The set notation is continued throughout the thesis.

Some examples given in this chapter employ nomenclature and abbreviations highly particular to aero-engine control systems. Examples include 'PRSOV_POS' and 'FMV_DRIVE'. This nomenclature serves to align examples to the target system. Readers unfamiliar with the meaning of these terms should be aware that they are chosen arbitrarily and do not influence the method presented. Intuitive abbreviations are used where possible. The architecture frameworks are presented using Unified Modelling Language (UML)/Systems Modelling Language (SysML) diagrams. Where textual interpretation of a diagram is

given, entities are highlighted in **bold**, multiplicities stated in [square brackets], relations underlined and relationships *underlined and italicised*

5.4.1 Engine Architecture Framework

An engine is considered to be an adynamic system of fixed dimensions with an associated set of components: sensors, actuators and subsystems *(see figure 5.4.1)*; it is simply a target platform for a DCS. The sensors, actuators and subsystems are the set of components required to measure physical parameters and actuate the necessary control action. In the parlance of this research, the term, 'subsystem' implies an engine component incorporating more than one sensor or actuator. Practical examples of subsystems include the Fuel Metering Unit (FMU), the solenoid banks and the T20P20 probe.

The engine framework includes dimensional data used to constrain the placement of nodes and harnesses. Dimensional information includes the length and diameter of the engine core and fancase. Each component has a height, width and depth and a unique location on the engine chassis. The location of a component is deemed the point at which its electrical interfaces is presented to the control system. *eg.* A speed probe may be mounted deep within the engine chassis, yet the electrical interface is located on the engine case. This model is perhaps oversimplified but is considered representative of the perspective that Aero Engine Controls could expect should the company develop DCSs commercially. The optimisation does not consider the placement of engine or Full Authority Digital Engine Controller (FADEC) components beyond the distributed EEC.

Amongst other parameters, the engine environment comprises the temperature and

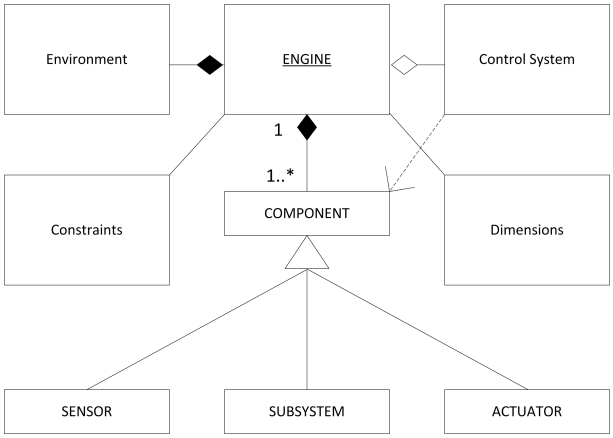


Figure 5.4.1: UML class diagram of the logical framework for the metaphysical engine

vibration experienced during operation and the accessibility of different engine zones to maintenance staff.

The constraints are those factors which limit the layout and composition of the DCS on engine. Constraints include areas where control system hardware cannot be placed and locations through which harnesses may or may not be routed. These constraints are mostly realised as 'keepout zones' which preclude the placement of hardware and harness routing in certain areas of the engine.

The functionality of the engine control system is represented by the 'control system' framework which aggregates the engine framework. It is assumed that the functionality of the control system belongs to the engine rather than the DCS. *i.e.* The tasks required to control the engine are defined independently of the control system hardware necessary to implement them. From the perspective of the optimiser, it is assumed that all engine components fulfil the required specification and that electronic interfaces and the computational resources are available. As a result, it is impossible for the optimisation process to produce DCS architectures that are incapable of meeting the functional requirements. The algorithm is concerned with how best to assemble the constituent DCS hardware rather than the design of the parts themselves.

Set Notation

Only one entity relation in the Engine framework is required during the optimisation process:

The Engine-Component relation may be read as: [one] **Engine** incorporates [one to many] **components**.

- Each engine e belongs to the set of engines in the product family E. Therefore:
  $E = \{\, e \mid e \text{ is an engine in the product family considered during optimisation} \,\}$
  eg. $E = \{\, \text{Trent } 1000, \text{RB}282, ... \,\}$
- Each component, c belongs to a set of components C. Therefore $c \in C$
  $C = \{\, c \mid c \text{ is a sensor, actuator or subsystem to which the control is required to interface} \,\}$
  eg. $C = \{\, \text{FMU}, \text{LP Speed probe}, \text{Oil Quantity Sensor}, ... \,\}$
- The relational set $R^{(EC)}$ is a set of ordered pairs $(e, c) \in R^{(EC)}$ denoting which engines in the product family incorporate certain components.

eg. if the Trent 1000 has an FMU then $(\text{Trent } 1000, \text{FMU}) \in R^{(EC)}$

- The relation $R^{(EC)}$ has an equivalent matrix $\mathbf{R_{EC}}$
- The relation is symmetric and non-reflexive

5.4.2 Control System Framework

The control system framework (figure 5.4.1) aggregates the engine framework. The framework defines the resources required to control, sustain and monitor the engine during flight. The control system is considered a platform independent entity at a level of abstraction above the EEC hardware. That is to say that the control laws are abstract of the underlying hardware and could be implemented on any platform providing the appropriate measurement, actuation and computational resources are available. Indeed, system functionality is usually beyond the influence of Aero Engine Controls when designing present-day centralised systems. Any given control system can be implemented on either centralised hardware or any level of distributed hardware. In general, control laws for different engines are broadly transferable; the control system hardware is considered specific to a particular engine. A UML diagram for the control system framework is given in figure 5.4.2.

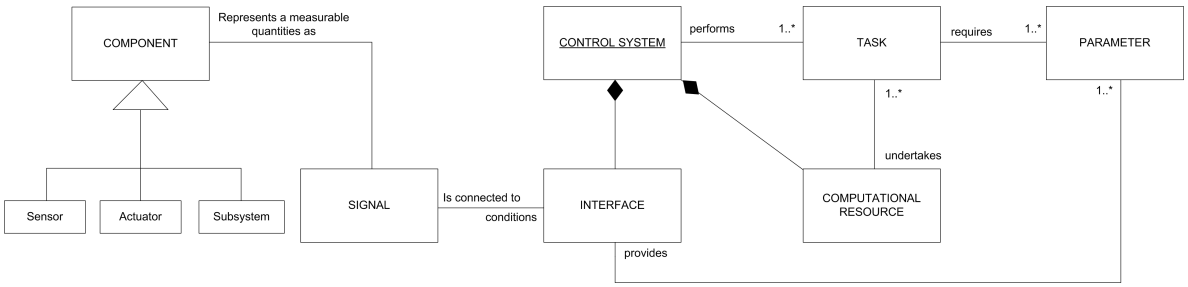


Figure 5.4.2: UML diagram of the control system framework

The control system framework comprises many non-tactile elements: signals, tasks and the input and output parameters on which those tasks depend. It is assumed that the functional behaviour of the control system is predetermined and not influenced by the specific hardware platform on which it is deployed. With reference to figure 5.4.2 the entities in the framework adhere to the following definitions.

Signals Unconditioned inference of a measurable engine parameters. Usually electric quantities (voltage/current) representing transducer measurements or actuator stimuli associated with engine components (sensors, actuators or subsystems). Signals

may be analogue, digital, coded, baseband or modulated. The control system designer has visibility of the component interfaces as shown in figure 5.4.1. Each engine component may provide or require one or more signals.

Interface Conditions, calibrates and converts signals to yield usable parameters. Conventionally, interfaces are implemented as a combination of analogue electronics and signal processing software and realised as Common Circuit Blocks. Interfaces are the only elements of the control system model with a physical analogue.

Parameter References, values or commands representing states of the system - usually as tangible values in standard units commodities. *eg.* spool speed measured in $rads^{-1}$

Task An action or activity performed by the control system in order to start, sustain, monitor and shutdown the engine. Tasks usually perform arithmetic or logical operations on input parameters in order to calculate and perform control system actions. Tasks include both operating system and application system level procedures and may be arranged hierarchically.

Computational Resource A computing platform capable of performing the calculations and logic required to process input signals and execute the engine control algorithm. Typically, this is a microprocessor but could be a power PC or external computer.

Set Notation

Four entity relations in the Control System Framework are used during the optimisation process:

1. The Component-Signal relation may be read as: [one] **component** provides [one to many] **signals**.

- Each component is a member of the set C (defined previously).
- Each signal s belongs to a set of signals S. Therefore $s \in S$

  $S = \{\ s \mid s$ is an electronic quantiy representing a physical quantity measured on the engine $\}$

  eg. $S = \{$ T20_RAW, N1-T1_RAW, AFDX_RAW, ... $\}$
- The relational set $R^{(CS)}$ is a set of ordered pairs $(c, s) \in R^{(CS)}$ denoting which components in the product family supply certain components.

eg. if the FMU (component) provides a raw measurement of the Fuel Metering Valve (FMV) position then $(\text{FMU}, \text{FMV\_POS\_RAW}) \in R^{(EC)}$

- The relation $R^{(CS)}$ has an equivalent matrix $\mathbf{R_{CS}}$
- The relation is symmetric and non-reflexive

2. The Signal-Interface relation may be read as: [one] **signal** is connected to [one] **interface**.

- Each signal is a member of the set S as defined above
- Each interface i belongs to a set of interfaces I. Therefore $i \in I$

  $I = \{\, i \mid i \text{ referes to a common circuit block for conditioning a given signal type} \,\}$
  eg. $I =$

  $\{\, \text{LVDT\_INTERFACE}, \text{SPEED\_PROBE\_INTERFACE}, \text{AFDX\_INTERFACE}, ... \,\}$
- The relational set $R^{(SI)}$ is a set of ordered pairs $(s, i) \in R^{(SI)}$ denoting which signals [are] connected to which interfaces.

  eg. if the FMV position signal is measured by a LVDT then

  $(\text{FMV\_RAW}, \text{LVDT\_INTERFACE}) \in R^{(SI)}$
- The relation $R^{(SI)}$ has an equivalent matrix $\mathbf{R_{SI}}$
- The relation is symmetric and non-reflexive

3. The Interface-Parameter relation may be read as: [one] **interface** provides [one to many] **parameters**.

- Each interface is a member of the set I as defined above
- Each parameter p belongs to a set of interfaces P. Therefore $p \in P$

  $P = \{\, p \mid p \text{ is an input out output value used by a control system task} \,\}$
  eg. $P = \{\, \text{P30}, \text{T25}, \text{N1-T}, ... \,\}$
- The relational set $R^{(IP)}$ is a set of ordered pairs $(i, p) \in R^{(IP)}$ denoting which interfaces provide certain parameters used by the control system.

  eg. if a given instance of the Linear Variable Displacement Transducer (LVDT) interface provides the parameter PRSOV_POS then

  $((\text{instance of}) \text{ LVDT\_INTERFACE}, \text{PRSOV\_POS}) \in R^{(IP)}$
- The relation $R^{(IP)}$ has an equivalent matrix $\mathbf{R_{IP}}$
- The relation is symmetric and non-reflexive

4. The Task-Parameter relation may be read as: [one] **task** requires [one to many] **parameters**.

- Each task q belongs to a set of tasks Q. Therefore $q \in Q$

  $Q = \{\ q \mid q$ is a task (function) performed by the control system $\}$
  eg. $Q =$

  $\{$ THRUST_CONTROL, START_ENGINE, MINOR_LOOP_FUEL_CONTROL, ... $\}$
- Each parameter is a member of the set P as defined above
- The relational set $R^{(QP)}$ is a set of ordered pairs $(q, p) \in R^{(QP)}$ denoting which tasks require certain parameters in order to perform their function.

  eg. if a task START_ENGINE requires the parameter N1T-1 then

  (START_ENGINE, N1T-1) $\in R^{(QP)}$
- The relation $R^{(QP)}$ has an equivalent matrix $\mathbf{R_{QP}}$
- The relation is symmetric and non-reflexive

Having defined the framework for the engine and control system, the following section describes how the metaphysical model of the engine is realised in software.

5.4.3 Metaphysical Engine Model

The metaphysical engine model is based upon the framework presented in section 5.4.1 and implemented as the instantiation of a software class. The class is constructed using information from the relational database prior to the optimisation starting. Once created, the model remains static for the duration of the process.

Attributes of the engine class (figure 5.4.3) include the dimensions of the engine such as the fancase length and diameter and the environment profiles. Sub-classes of the engine class include the sensors, actuators and subsystems. Whilst the engine framework contains no methods, the class relies on some ancillary methods to calculate temperature profiles and convert between coordinate systems.

The engine structure and DCS layout are performed using a 3-dimensional (3D) representation of the engine. The engine has a length l_e, a fancase circumference θ_f and a core circumference θ_c. The position of components on the engine is denoted in 3D space using cylindrical-polar co-ordinates (r, ϕ, z). The z-axis runs through the centre of the engine's core and the angle ϕ is assumed to be zero at the base of the engine and increase in an anti-clockwise direction if looking at the engine from the front. The scheme is shown in figure 5.4.4. The parameter r may take one of only two distinct values: the diameter of the engine's core θ_c, or the diameter of the fancase θ_f - accordingly, all engine

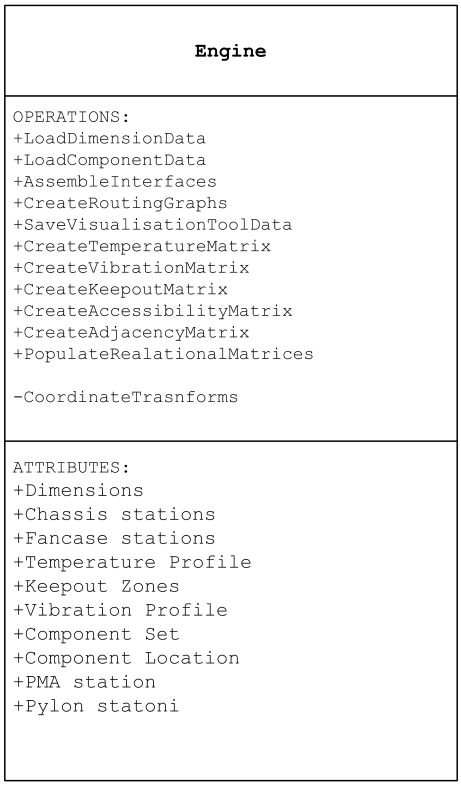


Figure 5.4.3: Engine Class

mounted items appear to be on the upper surface of the engine chassis. The output from the visualisation tool is shown in figure 5.4.5

The following sections detail how each of the class's attributes are realised in software.

Engine Structure and Dimensions

Designing effective DCS architectures relies heavily on knowledge of the engine structure and dimensions. An elementary engine structure has been assumed for the purposes of

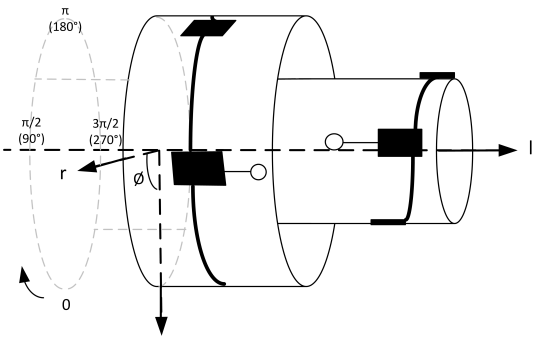


Figure 5.4.4: The 3-dimensional engine

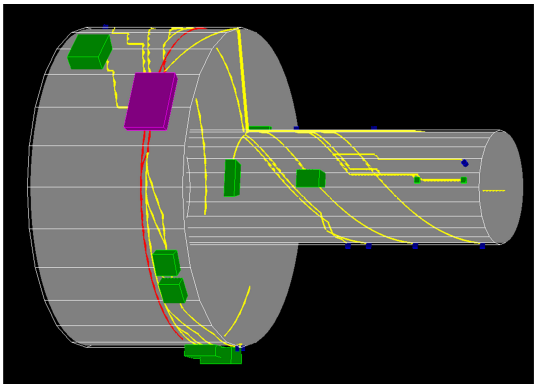


Figure 5.4.5: Engine viewed using the visualisation tool

this work. Each engine has a fancase and core composed of the Intermediate Pressure Compressor (IPC), High Pressure Compressor (HPC), Combustion Chamber (CC) and the turbine stages. The turbine stages are combined to form a single entity. This structure is representative of a large, civil, triple-spool jet engine. The engine core is of uniform diameter and supporting struts and chassis mountings are neglected. The engine structure, dimensions, notation and units are shown in figure 5.4.6 and figure 5.4.7. Dimensions marked with an * are derived from other measurements.

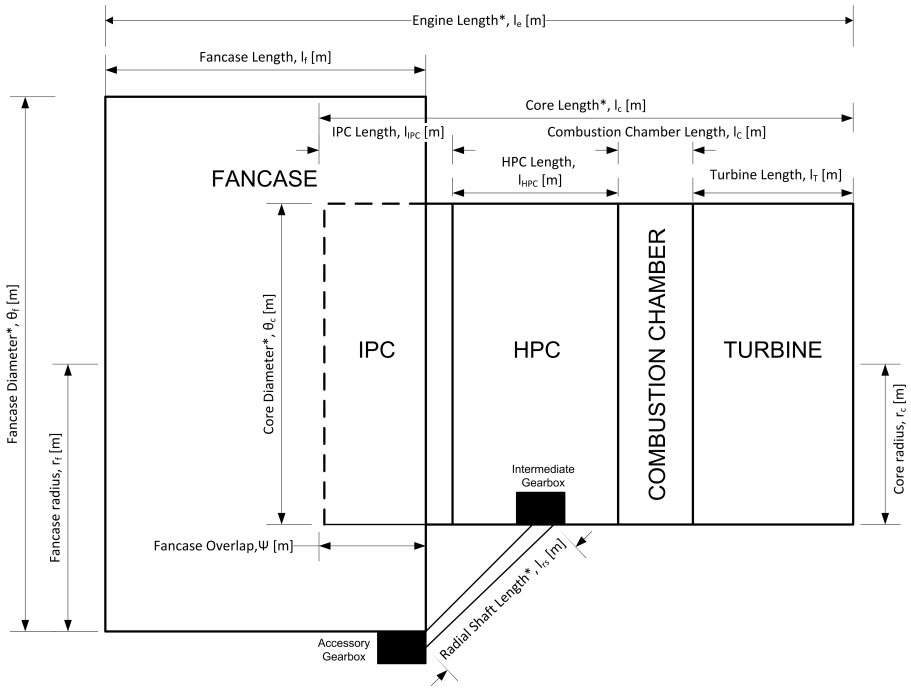


Figure 5.4.6: Elementary engine model showing engine stages, dimensions and nomenclature.

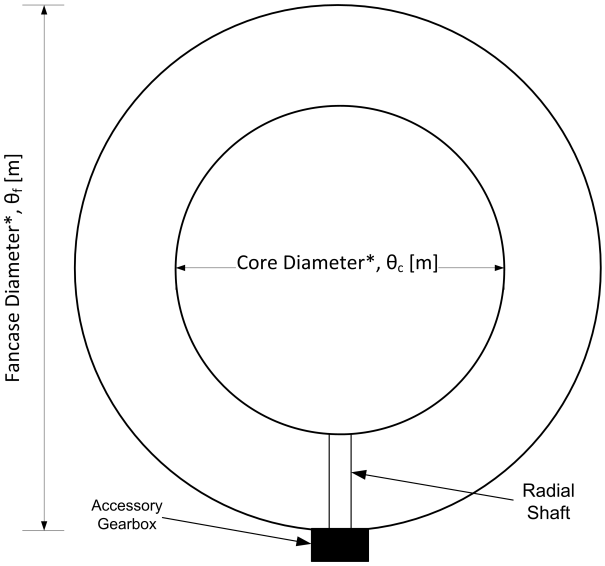


Figure 5.4.7: Engine model from the rear of the engine

The most basic derived dimensions are the fancase diameter, θ_f which is twice the fancase radius, $2r_f$ and similarly the core diameter, θ_c is equal to $2r_c$. Two parameters not shown on the diagram are the core and fancase circumferences, $c_c = 2\pi r_c$ and $c_f = 2\pi r_f$ respectively, The core length, l_c is the summation of the core engine stage lengths: the IPC, l_{IPC}, the HPC l_{HPC}, the combustion chamber l_{CC} and the turbine length, l_T. The engine length, l_e is given by $l_f + l_c - \Psi$. The fancase overlap, Ψ is the distance in metres by which the fancase overlaps the engine core with respect the rear of the fancase.

The radial shaft transmits power from the accessory gearbox to the intermediate gearbox through the bifurcation duct. The shaft is of little interest to DCS designers yet its length provides an important dimension for harness routing *(see section 5.11)*. The length of the radial shaft, l_{rs} is the hypotenuse of the right angled triangle formed from the fancase radius, the IPC length and the HPC length. Assuming that the intermediate gearbox is located half way along the bottom case of the HPC, the length of the shaft may be calculated using pythagoras's theorem as in equation 5.1:

$$l_{rs} = \sqrt{\left(\frac{r_f - r_c}{2}\right)^2 + \left(l_{IPC} + \frac{l_{HPC}}{2} - \Psi\right)^2} \tag{5.1}$$

The structure of the engine may be modified to represent an open rotor engine by assigning the fancase diameter equal to the core diameter ($\theta_f := \theta_c$) and setting the fancase overlap, Ψ, to zero.

Chassis Stations

The engine chassis is divided into a grid which covers the fancase and core. The intersections of the grid's horizontal and vertical lines are known as *chassis stations* and represent a set of discrete locations on which control system components and harness vertices may be placed. The chassis stations are illustrated in figure 5.4.8 and the stations of the real model in figure 5.4.9. A chassis station must either be vacant or uniquely occupied - in most cases, nodes cannot be placed upon existing components and harnesses cannot be routed over nodes or components. Chassis stations are used to form a graph over which the wiring harnesses are routed (section 5.11). The resolution of the grid may be changed to improve placement and routing accuracy at the expense of computational time and architectural permutations. Component locations are mapped from cylindrical coordinates to chassis stations using transform functions associated with the engine class.

The grid resolution is set by the grid square height Δ_ϕ and the grid square width Δ_z. The resolution of the grid is easily changed in software but remains constant for the duration of an optimisation. Therefore, the number of chassis stations available on a given engine is found by equation 5.2:

$$\frac{l_e}{\Delta_z} \times \frac{2\pi}{\Delta_\phi} \equiv x \times y \tag{5.2}$$

The number of chassis stations in the x-axis and y-axis are denoted by the parameters x and y respectively. Typically, Δ_z and Δ_ϕ may be set to a value of between 0.05m and 0.1m meaning that a 4m long engine with a 0.5m core radius would have between 1200 and 5040 chassis stations. The difference in diameter between the fancase and core means

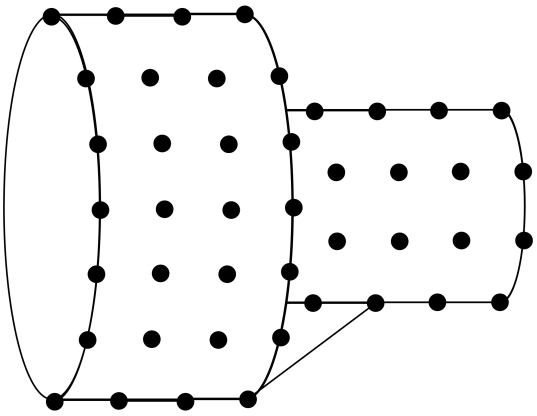


Figure 5.4.8: The 3-dimensional engine chassis stations

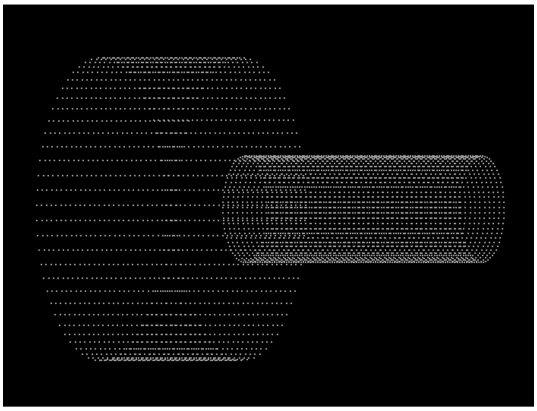


Figure 5.4.9: Chassis stations viewed using the visualisation tool. The engine has 5040 chassis stations

that chassis stations on the fancase cover a larger area than those on the core. Stations are numbered sequentially from the top left to bottom right - numbers are stored in a $x \times y$ matrix $\boldsymbol{L}$ whose elements $\boldsymbol{L}_{(a,b)}$ are the integer station numbers:

$$\boldsymbol{L} = \begin{bmatrix} 1 & \dots & \dots & x \\ x+1 & \dots & \dots & 2x \\ 2x+1 & \dots & \dots & 3x \\ \vdots & \dots & \dots & \vdots \\ (y-1)x+1 & \dots & \dots & yx \end{bmatrix} \tag{5.3}$$

Engine components, nodes and harness vertices are assigned to chassis stations. Once the optimisation starts, cylindrical coordinates are neglected and all positional references are made using chassis station numbers alone. Parameters such as temperature and accessibility are recorded in corresponding matrices where the elements hold values relating to a particular chassis station.

The fancase matrix $\boldsymbol{F}$ has the same dimensions as $\boldsymbol{L}$ and denotes which stations are on the fancase and which are on the engine core. The matrix is essentially a look-up table and used in the translation between cylindrical coordinates and chassis stations.

$$\boldsymbol{F} = \left[\begin{array}{c|ccc} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right] \quad \text{where} \quad \boldsymbol{F}_{(a,b)} = \begin{cases} 1 & \text{if } \boldsymbol{F}_{(a,b)} \text{ is on the fancase} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

The rows of $\boldsymbol{F}$ are always identical - the full matrix is retained to promote simplicity in other parts of the algorithm.

Engine Components

Every engine component is located at a unique chassis station. The component matrix $\boldsymbol{C}$ relates the placement of components to chassis stations. Each component is represented by a unique identifer taken from the relational database (section A). $\boldsymbol{C}$ has the same dimensions as $\boldsymbol{L}$:

$$\boldsymbol{C}_{(a,b)} = \begin{cases} c & \text{where } c \text{ is the unique component ID} \\ 0 & \text{if the respective chassis station is unoccupied} \end{cases} \tag{5.5}$$

eg. For a set of components $C = \{1\ 2\ 3\ 4\ 5\ \}$:

$$\boldsymbol{C} = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 4 & 0 & 5 & 0 \\ 0 & 0 & 2 & 0 \end{array}\right] \tag{5.6}$$

Environment Profiles

Three environmental parameters are considered: temperature, vibration and accessibility. All three profiles are used to constrain the placement of control system hardware and harnesses. Furthermore, keepout zones indicate areas of the engine chassis that may not be populated by DCS components. Uses for keepout zones are plentiful but the notion admits miscellaneous environmental considerations including the avoidance of pipework and areas of the engine deemed at risk from engine debris.

The environment is considered to be uniform across the area covered by a particular chassis station. Consequentially, increasing the number of chassis stations (by changing values for Δ_z and Δ_ϕ) allows for greater resolution in profile data. Each environmental parameter has an associated matrix of dimensions $x \times y$ whose elements represent the value of that parameter at the relevant chassis station.

Temperature

Temperature data may be entered on a station-by-station basis or derived from basic mathematical functions. Each element of the temperature matrix $\boldsymbol{\Lambda}_{a,b}$ indicates the temperature at the respective chassis station $\boldsymbol{L}_{(a,b)}$ in degrees celsius. *eg.* for an engine with 16 chassis stations:

$$\boldsymbol{\Lambda} = \left[\begin{array}{c|ccc} 78.6 & 210.5 & 430.8 & 740.0 \\ 90.2 & 205.4 & 404.6 & 734.7 \\ 87.6 & 200.7 & 420.4 & 699.4 \\ 88.3 & 198.2 & 430.1 & 755.8 \end{array}\right] \overset{\circ}{\cdot} C \tag{5.7}$$

There are many different measures of engine temperature - in this research, temperature refers to Surrounding Air Temperatures (SATs) as this is most important to hardware designers. Other measures of temperature include surface metal temperature and gas stream temperature. The temperatures matrix indicates the maximum SATs at the respective chassis station.

Derived Temperature Profiles

Temperature data for real engines is both commercially sensitive and difficult to accumulate. Rather than requiring high resolution temperature maps for this work, temperatures are derived from mathematical functions aligned with representative values. These functions provide a scalar value of engine temperature at a given cylindrical coordinate. The scalar values are mapped to engine stations and entered into the temperature matrix. Temperature profiles are derived from either a straight line (equation 5.8) or exponential function (equation 5.9):

$$\lambda(z,\phi) = m_\lambda z + \lambda(z=0) \qquad (5.8) \qquad\qquad \lambda(z,\phi) = \lambda(z=0)e^{\alpha_\lambda z} \qquad (5.9)$$

Where λ is the temperature at a given location, z is the position along the length of the engine, m_λ is the temperature gradient, $\lambda(z=0)$ is the temperature at the front of the engine and α_λ determines the rate of exponential growth. In practice, it is easier to specify a fancase temperature $\lambda(z=0)$ and a turbine temperature λ_{max} and calculate a linear or exponential change from one to the other. This approach assumes that the maximum temperature occurs at the turbine - whilst this is not true of the gas-stream, the ducting of hot air around the engine and heat-soak make this a realistic model for component mounting.

Variations in circumferential temperature are less characterisable and highly dependent on the placement of pipework and ancillary systems. Accordingly, the temperature is assumed to be constant for all values of ϕ. SATs vary considerably from engine to engine but typical fancase and turbine case temperatures are 85°C and 600°C respectively. The fancase is generally uniform and every chassis station on the fancase may be set to a single temperature *eg.* $\boldsymbol{\Lambda}(\boldsymbol{F}=1)=85$.

Temperature profiles for real engines are obviously more complex. The profiles above provide sufficient accuracy to demonstrate the principle.

Vibration

Similarly, the elements of the vibration matrix, $\boldsymbol{V}$ indicate the vibration level at a particular chassis station. Obviously, a full spectral analysis would require that a vibration profile for a node be calculated and compared against design limits - this is well beyond the scope of this research although it is not implausible to add such functionality to the evaluation functions. The vibration at a given engine station indicates the severity of vibration on a scale from 0 to 1 with 1 being the most hostile. The fancase is the most hostile environment with the compressor stages being the most benign. As for temperature, vibration profiles may be derived from basic functions.

$$\boldsymbol{V}_{(a,b)} = \left\{ \quad \text{a value representing the vibration environment at } \boldsymbol{L}_{(a,b)} \text{ in the range (0,1)} \right. \tag{5.10}$$

Accessibility

Elements of the accessibility matrix, $\boldsymbol{A}$ state the length of time in hours that a maintenance technician would take to access the specific chassis station. Generally, the fancase cowl makes accessing components at the front of the engine relatively quick. Components towards the rear of the engine are less easily accessed and require time to cool before they become safe to work on. As before, profiles may be derived from basic functions in place of real world data.

$$\boldsymbol{A}_{(a,b)} = \left\{ \quad \text{the time in hours taken to access the chassis station } \boldsymbol{L}_{(a,b)} \right. \tag{5.11}$$

Keepout Zones

Keepout zones are chassis stations where nodes cannot be placed and harnesses cannot be routed. Keepout zones may be used to maintain clearances between control system nodes and existing engine subsystems, pipes or components. Keepout zones may also be used to

isolate areas prone to cross engine debris. Like the temperature and vibration matrix, each element of the keepout matrix relates to the respective chassis station given in $\boldsymbol{L}$.

The keepout matrix $\boldsymbol{K}$ has the same dimensions as $\boldsymbol{L}$ - elements of the matrix take the value one if the station is a keepout zone or zero otherwise:

$$\boldsymbol{K}_{(a,b)} = \begin{cases} 0 & \text{if } \boldsymbol{L}_{(a,b)} \text{ is a keepout zone} \\ 1 & \text{otherwise} \end{cases} \tag{5.12}$$

5.4.4 Finding Harness Lengths and Paths

Further facets of the engine class are the attributes and operations necessary to determine distances between engine stations and route harnesses. The functions used to calculate harness lengths, configure network topologies and route harnesses are based on techniques from graph theory: namely, the Floyd-Warshall algorithm (Floyd, 1962) and the Farthest Neighbour Nearest Insertion (FANI) routing algorithm (Ravikumar *et al.*, 1998). Within the model, the engine chassis is cloaked in a cylindrical graph where the vertices are the chassis stations and the edge quantities the distances between them. Chassis stations (and hence vertices) are considered to be connected if they are vertically, horizontally or diagonally adjacent. There is no connection between the fancase and core other than via two edges protected by the bifurcation duct.

The graph wraps around the entire engine chassis and may be thought of as being, '3-dimensional'. All edge dimensions are positive. Figure 5.4.10 shows how the adjacent chassis stations are connected to form a graph for distance finding and harness routing:

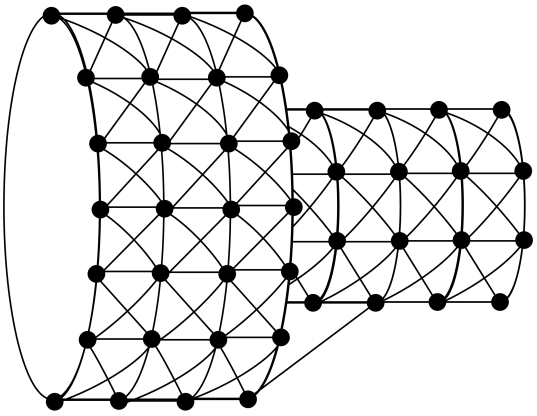


Figure 5.4.10: Pictorial representation of the engine graph used for routing and distance finding. The black circles represent the vertices (chassis stations) and the lines the edges between adjacent vertices.

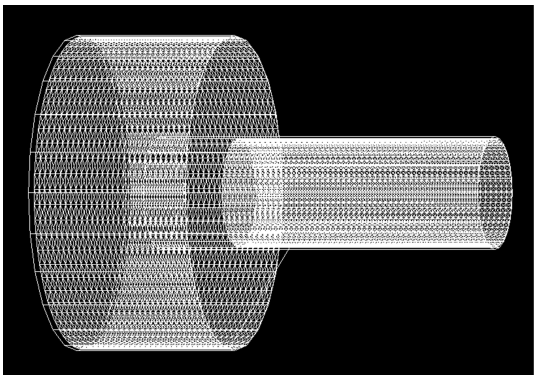


Figure 5.4.11: The routing graph as displayed by the visualisation tool

Vertex and edge dimensions define the graph G:

$$G = \{\{L\}, \{W\}\} \tag{5.13}$$

Where $l \in L$ is a set of chassis stations of size xy and $w \in W$ is a set of ordered pairs denoting connectivity between vertices ($w \subset L$). In this application, vertices are synonymous with chassis stations.

The edge lengths are the distances between chassis stations. If two chassis stations are not connected, the edge dimension is infinity. It is important that the edge dimensions are known and accurate as this information is used to ascertain the length and consequentially, the weight of each harness. Figure 5.4.12 shows a portion of the graph from for a typical engine. The full graph for an engine has $|L| \equiv xy$ vertices (5040) in this implementation).

For the graph in figure 5.4.12 chassis station 1 is connected to chassis stations 2, 6 and 7. Therefore, the pairs $\{1, 2\}$, $\{1, 6\}$ and $\{1, 7\}$ are all members of the set W. The relation is symmetric.

The edge dimensions are a function of the the resolution parameters, Δ_z and Δ_ϕ. Consequentially, harnesses are routed to the same resolution as nodes are and engine locations are placed. This is advantageous as a hareness may be routed between component and node locations using station numbers alone. However, harness routing in real world applications is likely to require a far higher degree of accuracy than is applied here.

The horizontal distance between connected chassis stations is always Δ_z (except at the boundary between the fancase and core). The distance between vertically adjacent

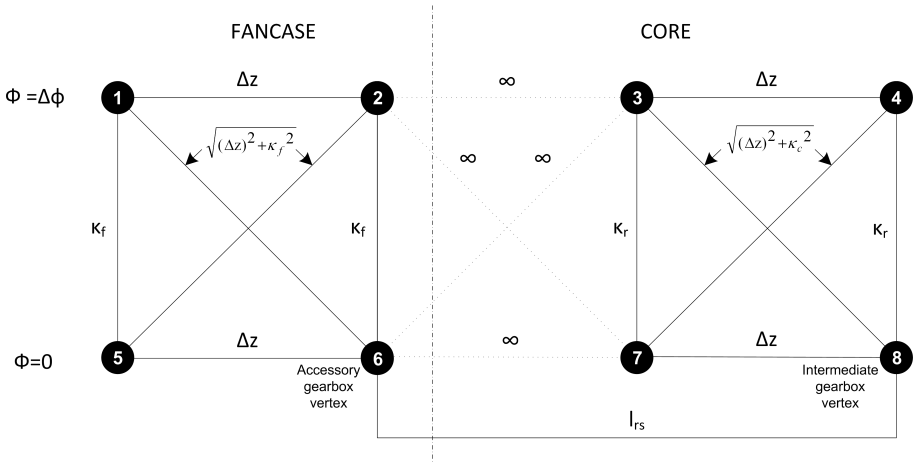


Figure 5.4.12: Section of the harness routing graph

engine stations is dictated by the parameter Δ_ϕ. Therefore, the vertical distance between vertices is dependent on the radius over which the graph is formed and changes depending on whether that portion of the graph relates to the core or the fancase. This approach maintains simplicity and avoids the need for separate fancase and core routing graphs.

As all harnesses are routed on the upper surface of the chassis, the distance between vertically adjacent vertices, κ is the arc length of the segment formed by the angle Δ_ϕ as shown in figure 5.4.13. Therefore, the distance between vertically adjacent nodes on the fancase, $\kappa_f = \Delta_\phi r_f$ and vertically adjacent nodes on the core is $\kappa_c = \Delta_\phi r_c$.

These dimensions are used for harness routing and consequentially, determining total harness length.

These edge lengths are used throughout the graph other than at the fancase core boundary. Wiring harnesses can only be routed from the engine to the fancase at the bottom and top of the engine where they are supported by a frame enclosed in the bifurcation duct. This envelopes the harnesses between the the accessory gearbox on the fancase to the intermediate gearbox on the core. The intermediate gearbox is located on the base of the engine core at approximately the mid-point of HPC chassis. All harnesses routed between the fancase and the engine core must follow these paths. Engine stations at the fancase-core interface are not connected (ie. their length is infinity) except at the vertices corresponding to the locations at either end of the bifurcation duct. Here, the edge length is l_{rs} (see figure 5.4.12).

The dimensions described above are stored in an adjacency-distance matrix $\boldsymbol{G}$. $\boldsymbol{G}$ is a $|L| \times |L|$ matrix whose elements $\boldsymbol{G}_{a,b}$ are the length of the shortest distance between

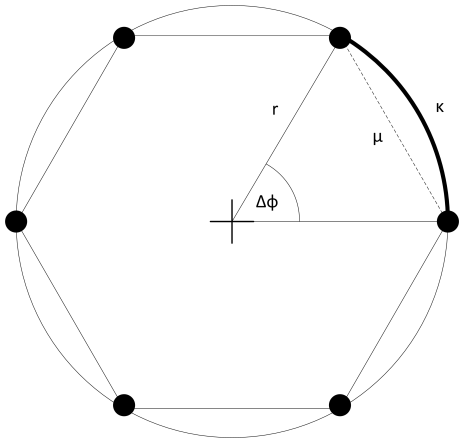


Figure 5.4.13: Graph dimensions across the engine case

vertex a and b. If no path exists, $\boldsymbol{G}_{(a,b)}$ is infinity. The distance matrix for the routing graph of figure 5.4.12 is given in equation 5.14.

$$\boldsymbol{G} = \begin{bmatrix} 0 & \Delta_z & \infty & \infty & \kappa_f & \alpha & \infty & \infty \\ \Delta_z & 0 & \infty & \infty & \alpha & \kappa_f & \infty & \infty \\ \infty & \infty & 0 & \Delta_z & \infty & \infty & \kappa_c & \alpha \\ \infty & \infty & \Delta_z & 0 & \infty & \infty & \alpha & \kappa_c \\ \kappa_f & \alpha & \infty & \infty & 0 & \Delta_z & \infty & \infty \\ \alpha & \kappa_f & \infty & \infty & \Delta_z & 0 & \infty & l_{rs} \\ \infty & \infty & \kappa_c & \alpha & \infty & \infty & 0 & \Delta_z \\ \infty & \infty & \alpha & \kappa_c & \infty & l_{rs} & \Delta_z & 0 \end{bmatrix} \tag{5.14}$$

$\boldsymbol{G}$ is calculated for a specific engine prior to optimisation and used for all subsequent harness routing and distance finding functions. The matrix is symmetrical about the leading diagonal which reflects the symmetry of the relationships represented. Mathematically, there is no need to replicate the information in the lower diagonal of the matrix. However, the Floyd-Warshall algorithm is most easily implemented by exploiting this symmetry and so the lower half is maintained.

5.4.5 Distances between engine stations (Floyd-Warshall Algorithm)

Harness routes and hence lengths are assumed to take the shortest feasible distances between source and destination chassis stations. Assuming that the shortest distance can be found, there is no need to find the path taken. The Floyd-Warshall algorithm provides a computationally efficient method of calculating the minimum distance between all nodes in a graph and can be extended to determine the path between vertices. The algorithm renders only distance information - it does not specify the walk between graph vertices corresponding to the shortest route.

In this implementation, the Floyd-Warshall algorithm uses the adjacency matrix $\boldsymbol{G}$ to produce a distance matrix $\boldsymbol{D}$ of dimensions $|L| \times |L|$ where each element corresponds to the shortest distance between each pair of chassis stations.

Once the distance matrix has been populated, it is used as a lookup table to quickly ascertain the distance between any two chassis stations on the engine. Whist the algorithm is computationally expensive, it proves a markedly efficient alternative to the process of

routing every harness on every iteration of the optimisation. The worst case computational time for the Floyd-Warshall algorithm is $\mathcal{O}(|L|^3)$. The pseudocode for the algorithm is shown in algorithm 5.4.5

Algorithm 5.4.1: FLOYDWARSHALL( )

$\boldsymbol{D} = \boldsymbol{G}$
for $k \leftarrow 1$ **to** $|L|$
  for $i \leftarrow 1$ **to** $|L|$
    for $j \leftarrow 1$ **to** $|L|$
      $\boldsymbol{D}(i,j) = \text{MIN}(\boldsymbol{D}(i,j), \boldsymbol{D}(i,k) + \boldsymbol{D}(k,j))$
      $\boldsymbol{N}(i,j) = k$

The algorithm is implemented as an operation of the engine class.

5.4.6 Control System Meta-physical Model

The control system model is based on the framework presented in section 5.4.2. The model (presented in full in appendix C) is too vast to describe in this thesis and not necessary to explaining its structure or principles. This section describes the main precepts and ideas behind the model using an example subset of functions.

The following paragraphs describe the process of modelling, visualising and verifying the task-parameter relationships. The process presented is broadly similar to that required for the component-signal, signal-interface and interface-parameter relationships discussed later.

Figure 5.4.14 presents an example dataflow diagram showing three control system tasks and the parameters they require. The tasks and parameters shown are a representative subset of a much lager functional model. The arrow headed lines represent the flow of parameters into or out of a particular task. The flows are labelled with the parameter number and name and may be visualised as a graph. Owing to the difficulties of obtaining formal descriptions, the functional model used has been created specifically for this research and is based solely on the author's understanding.

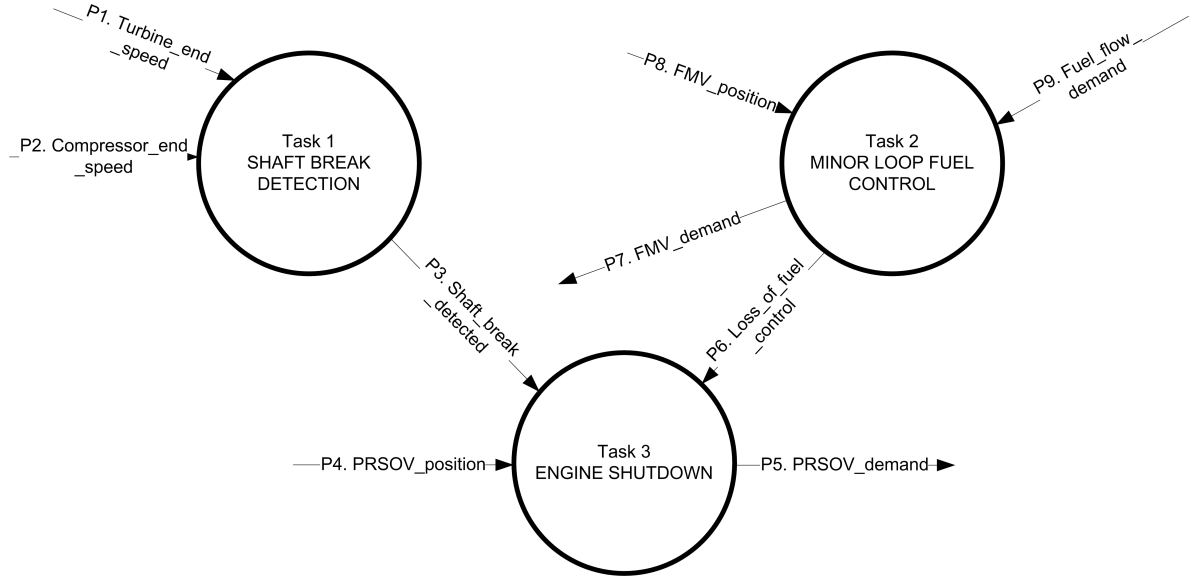


Figure 5.4.14: Data flow diagram showing an example subset of tasks and parameters

This functional representation is realised as a series of sets and matrices. The set of tasks Q, is a set of functions that the control system must perform in order to control and sustain the engine during flight. Each task requires a subset of the parameters in order to perform that task. The list associating parameters with tasks is the relational set $R^{(QP)}$. Both input and output parameters are treated equally as the direction of data flow between tasks is irrelevant to the optimisation algorithm.

In the example above, the three tasks require a total of nine parameters. Assuming that the task numbers correspond to rows and parameter numbers to columns, the relational matrix $\mathbf{R_{QP}}$ for the dataflow diagram in figure 5.4.14 is:

$$\mathbf{R_{QP}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.15}$$

These relationships are stored in the relational database. The algorithm reads the necessary values and populates the sets and matrices during initialisation.

The parameter flow is a closed system - a task may only receive parameters from or output parameters to another task or hardware interface. Parameters may not originate from or be returned to a any other entity. The model is verified by defining an adjacency matrix $\boldsymbol{Aqp}$ for the relation in $\mathbf{R_{QP}}$ and using a graph visualisation tool to produce a

graphical representation.

$$\boldsymbol{Aqp} \triangleq \begin{bmatrix} 0 & \mathbf{R_{QP}} \\ 0 & 0 \end{bmatrix} \tag{5.16}$$

The graph visualisation tool uses the adjacency matrix in equation 5.16 to display the corresponding graph. The output for 5.16 is shown in figure 5.4.15.

Manual verification shows that the graph in figure 5.4.15 has a structure equivalent to the dataflow diagram in figure 5.4.14. As previously stated, the direction of dataflow is unimportant and so an undirected graph is sufficient. This assumption simplifies the process of obtaining of the adjacency matrix.

The framework for the control system defines other relationships between components and signals, signals and interfaces and interfaces and parameters. The diagram below is an extension of the dataflow diagram in figure 5.4.14 and shows the other relationships.

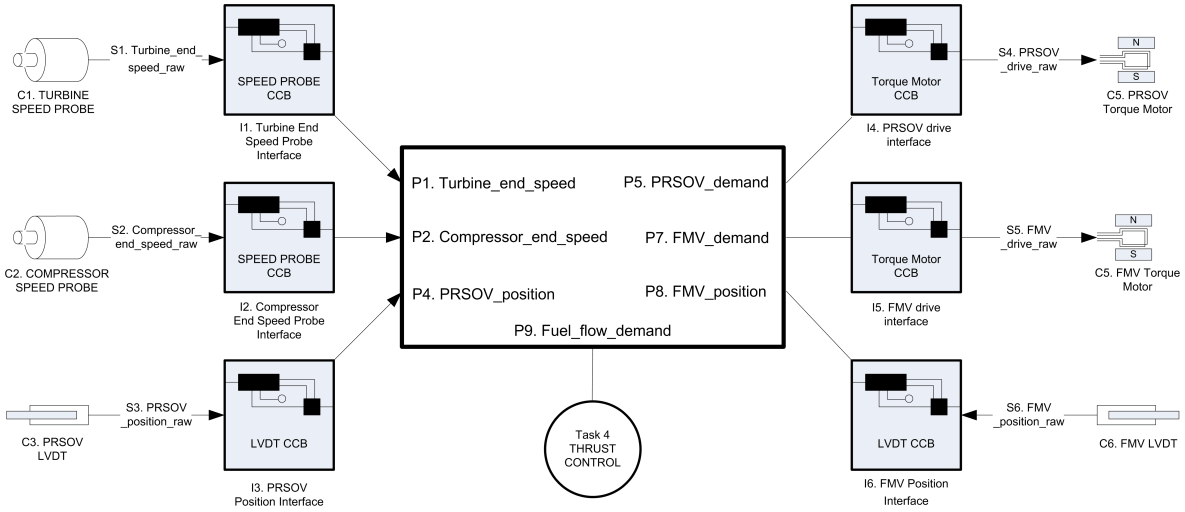


Figure 5.4.16: Logical diagram of the arrangement of components, interfaces and parameters for the example control system model

Figure 5.4.16 shows the components, the signals they produce, multiple instances of interfaces (shown as CCBs) and the parameters they provide. The data flow diagram of figure 5.4.14 is encapsulated in the rectangle with one further task (Task 4) added.

Using the same approach as for the tasks-parameters relationships we can define an adjacency matrix $\boldsymbol{A_{sys}}$ for the whole control system model:

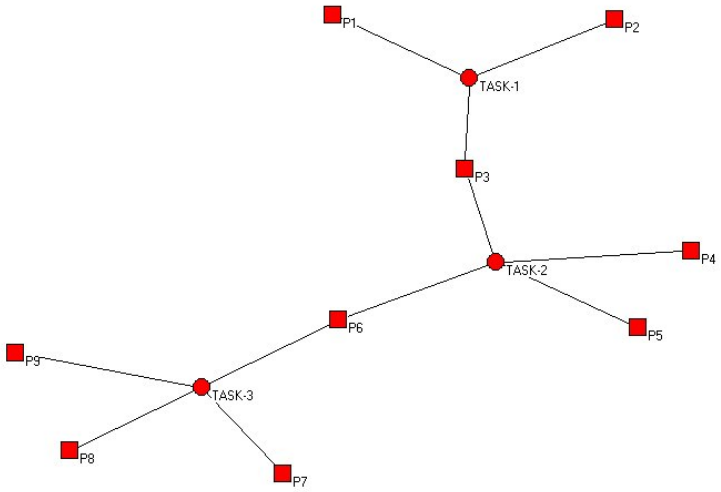


Figure 5.4.15: Graph representation of the task-parameter relationship matrix. The graph has the same structure as the functional flow diagram of figure 5.4.14. The square vertices represent parameters and the circular vertices represent tasks.

$$\boldsymbol{A_{sys}} \triangleq \begin{bmatrix} 0 & \mathbf{R_{CS}} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{R_{SI}} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{R_{IP}} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{R_{QP}}^T \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.17}$$

As before, a graph visualisation tool is used to provide a pictorial representation of the matrix $\boldsymbol{A_{sys}}$. The graph shown in figure 5.4.17 is equivalent to the system diagram given in figure 5.4.16.

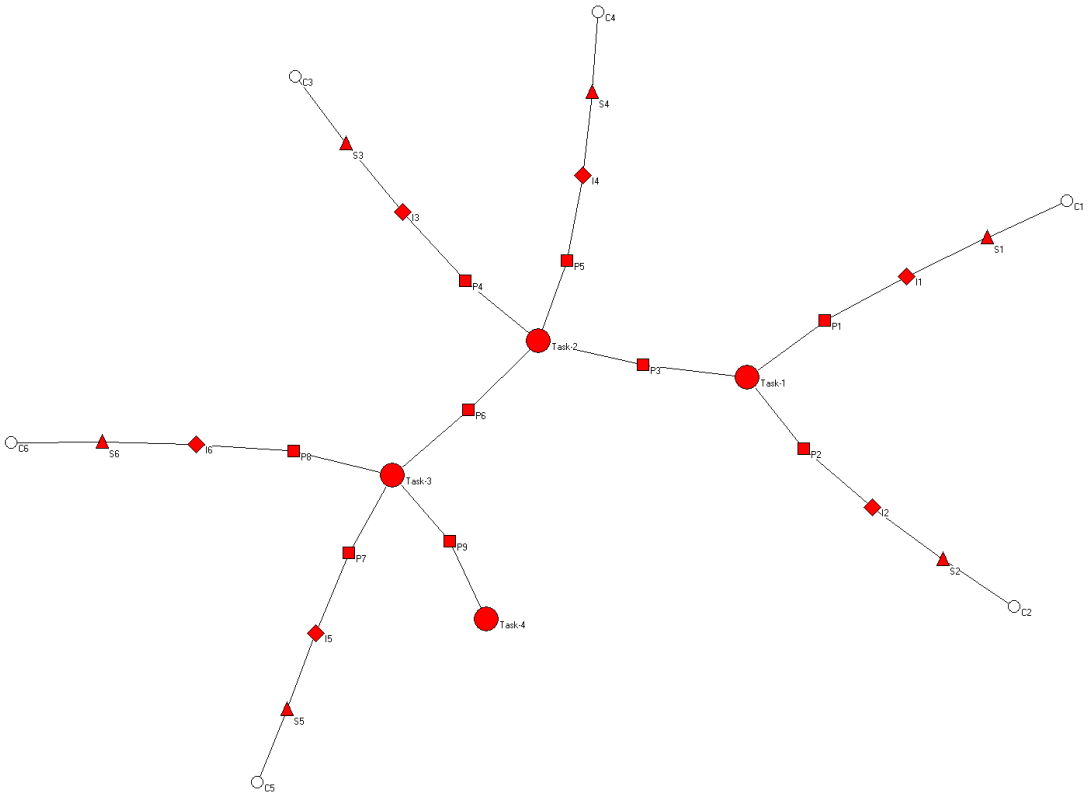


Figure 5.4.17: Graph representation of the logical diagram in figure 5.4.16. Hollow circles represent components, triangles signals, diamonds interfaces, squares parameters and shaded circles tasks.

5.4.7 DCS Architecture Framework

The framework shown in figure 5.4.18 considers a DCS as the aggregation of nodes, signal harnesses, data harness and a power system. The DCS aggregates the engine. Data harnesses carry parameters between nodes whilst signal harness connect engine components to nodes. The power system comprises harnesses, a distribution hub and conditioning circuitry. The structure of the power system is discussed in section 5.12.

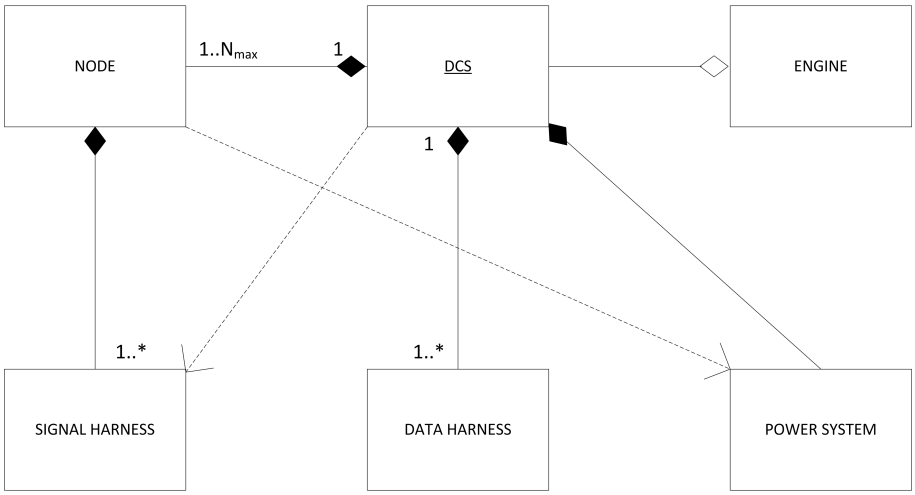


Figure 5.4.18: UML diagram of the logical architecture of the physical segment of the DCS model

A node (figure 5.4.19) is a single physical entity containing processing capability, circuitry and interfaces for power and data. Each node has two redundant channels although the redundancy scheme used herein differs substantially from that used in present day centralised systems. Both channels of the node have separate connectors for data and power harnesses. Every node contains power conditioning circuitry to provide the power processing platform and interfaces. A node need not have any connection to engine components (*i.e.* it is a dedicated computational platform) but may host electrical interfaces that permit connection to sensors, actuators and subsystems. A node can perform any number of system tasks. The interfaces in the node and the tasks it performs are allocated by the GA. The two UML diagrams represent a single framework - they are shown separately to aid presentation and should be thought of as concatenated by the 'Node' entity.

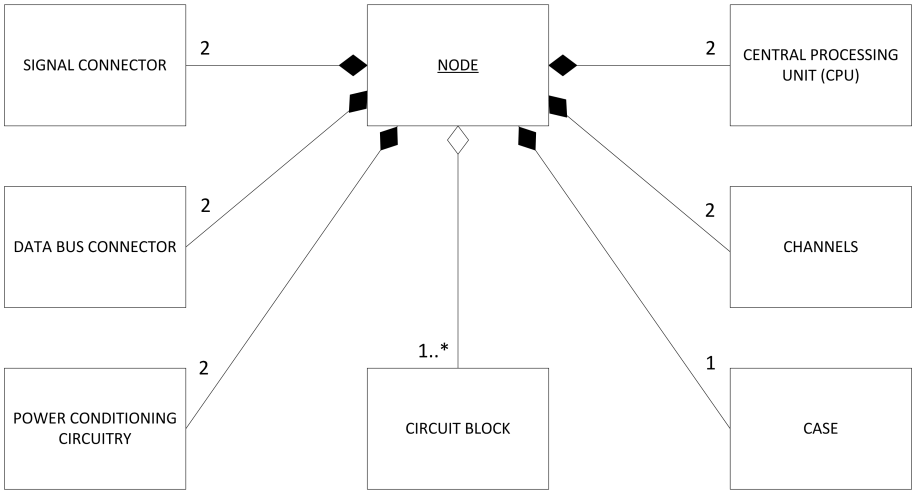


Figure 5.4.19: UML diagram of a node

A single instance of the DCS model is created for every member of the current population on every iteration of the genetic algorithm. Each DCS has three sets of wiring harnesses: a set of signal harness, a set of power harnesses and a set of data harnesses. Signal harnesses connect nodes to engine components whilst data harnesses transfer information between nodes. The number of cores is specific to the harness and the number of signals which require routing to a given node - the number of cores is determined from the interface specifications. Data harnesses carry inter-node communication data and are assumed to consist of a two wire digital bus. Each harness is a dynamically created instance of the harness class within the DCS class. The number of harnesses and their composition is likely to vary considerably between solutions. The source and destination of each harness depend upon the location of the system nodes. Further details on harness creation and routing are given in section 5.11.

Set Notation

The optimisation process does not require knowledge of the relationships between any elements of the DCS framework as all relations are either fixed or derived independently. However, the optimisation process requires the set of nodes.

- Each task n belongs to a set of nodes N. Therefore $n \in N$

  $N = \{\ n \mid (\eta_{min} \leq n\ \leq \eta_{max}) \in \mathbb{Z}\}$

  i.e. A node is identified by an integer number from 1 to the number of nodes considered in the optmisation.

The entities with relations to the set of nodes are described in (section 5.4.8) below.

5.4.8 The Hybrid Framework: DCS on Engine

An architecture framework for an engine (aggregated with a platform independent control system) and a Distributed Control System have been defined. It has been shown how the metaphysical models for the engine and control system are based on these frameworks and realised in software as sets of parameters and relational matrices.

Much of the structure instituted in these frameworks is not required to define or construct the architecture of DCSs during optimisation. This arises because either the entities and relationships remain unchanged for the duration of the optimisation, or the structure of the framework need not be reflected in the software architecture. Therefore, we define a hybrid framework containing only the entities and relations required during the construction and evaluation phase. The hybrid framework is compiled from entities of the engine, control system and DCS frameworks and represents the structure of a DCS on an engine. It should be noted that the hybrid framework does not negate the engine, control system and DCS models. These structures remain present in the software.

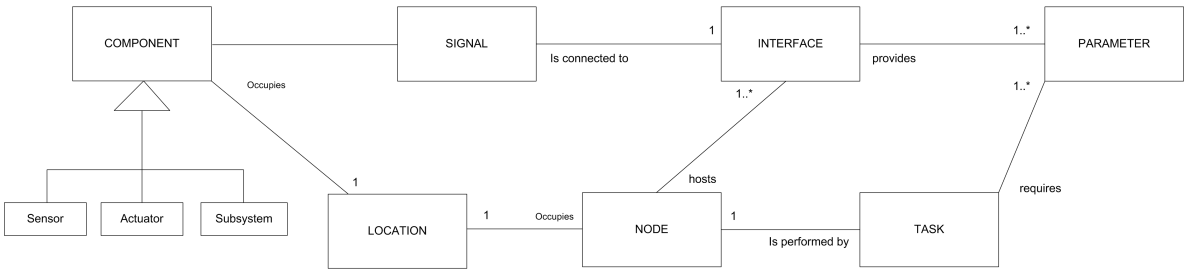


Figure 5.4.20: Hybrid model using elements from the Engine, DCS and control system frameworks

The following section demonstrates how the relationships arising from the hybrid model are used to construct each of the distributed architectures.

5.5 Composition and Analysis of the Hybrid Model using Binary Relations

It has been shown how binary relations between framework entities are recorded in relational sets and matrices. The ordered pairs infer the structure of the engine and DCS. Thus far, only relations between connected entities are considered. This section shows how multiplication of the relational matrices allows the relationships between all framework entities to be calculated; these new relations are used to construct and analyse the DCSs during the optimisation and evaluation.

The control system framework (section 5.4.2) states that interfaces provide parameters and those parameters are required by tasks. Supposing we want to know which tasks are dependent on which interfaces. This relation shows which tasks the control system will no longer be able to perform should a given interface fail.

It is possible to find the relationship between any two entities in the hybrid model using relationship composition. In effect, relationship composition adds a new relationship to the framework. Supposing we have three entities: A, B and C and want to find the relationship A→C. from the defined relationships A→B and B→C. The composition of the relationships is found by multiplication of the two relational matrices.

$$R^{AC} = R^{AB} \circ R^{BC} \equiv \boldsymbol{R_{AC}} = \boldsymbol{R_{AB}} \times \boldsymbol{R_{BC}} \tag{5.18}$$

Where $\circ$ denotes relational composition. If we consider the relationships R^{AB} and R^{BC} to infer that B is a function of A and C is a function of B, the process of matrix multiplication performs function composition:

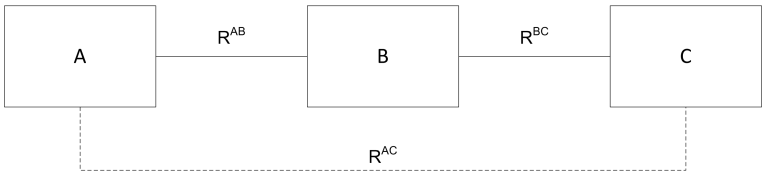


Figure 5.5.1: The new relation R^{AC} created as the composition of R^{AB} and R^{BC}.

$$\boldsymbol{R_{AC}} = \begin{bmatrix} {}_{a_1}R_{b_1}^{(AB)} & {}_{a_1}R_{b_2}^{(AB)} & {}_{a_1}R_{b_3}^{(AB)} \\ {}_{a_2}R_{b_1}^{(AB)} & {}_{a_2}R_{b_2}^{(AB)} & {}_{a_2}R_{b_3}^{(AB)} \\ {}_{a_3}R_{b_1}^{(AB)} & {}_{a_3}R_{b_2}^{(AB)} & {}_{a_3}R_{b_3}^{(AB)} \end{bmatrix} \begin{bmatrix} {}_{b_1}R_{c_1}^{(BC)} & {}_{b_1}R_{c_2}^{(BC)} & {}_{b_1}R_{c_3}^{(BC)} \\ {}_{b_2}R_{c_1}^{(BC)} & {}_{b_2}R_{c_2}^{(BC)} & {}_{b_2}R_{c_3}^{(BC)} \\ {}_{b_3}R_{c_1}^{(BC)} & {}_{b_3}R_{c_2}^{(BC)} & {}_{b_3}R_{c_3}^{(BC)} \end{bmatrix}$$
$$= \begin{bmatrix} {}_{a_1}R_{c_1}^{(AC)} & {}_{a_1}R_{c_2}^{(AC)} & {}_{a_1}R_{c_3}^{(AC)} \\ {}_{a_2}R_{c_1}^{(AC)} & {}_{a_2}R_{c_2}^{(AC)} & {}_{a_2}R_{c_3}^{(AC)} \\ {}_{a_3}R_{c_1}^{(AC)} & {}_{a_3}R_{c_2}^{(AC)} & {}_{a_3}R_{c_3}^{(AC)} \end{bmatrix} \tag{5.19}$$

The same result can be found by defining an adjacency matrix for the relations in equation 5.18 and applying the Floyd-Warshall algorithm (section 5.11) to find the length of the walk between vertices. This approach is less computationally expensive than matrix multiplication, but has some limitations discussed later. The Floyd-Warshall algorithm would require an adjacency matrix $\boldsymbol{A_{sys}}$ in the form:

$$\boldsymbol{A_{sys}} \triangleq \begin{bmatrix} 0 & \boldsymbol{R_{ab}} & 0 \\ 0 & 0 & \boldsymbol{R_{bc}} \\ 0 & 0 & 0 \end{bmatrix} \tag{5.20}$$

By using this principle, it is possible to define an adjacency matrix for the entire hybrid framework - every instance of every entity is considered to be a graph vertex. The matrix $\boldsymbol{A_{sys}}$ records the relationships between every pair of entities in the hybrid framework and consequentially the metaphysical model. In the parlance of this research, $\boldsymbol{A_{sys}}$ is the blueprint for a distributed architecture.

$$\boldsymbol{A_{sys}} \triangleq \left[\begin{array}{ccccccc|c} 0 & \mathbf{R_{EC}} & \mathbf{R_{ES}} & \mathbf{R_{EI}} & \mathbf{R_{EP}} & \mathbf{R_{EQ}} & \mathbf{R_{EN}} & \mathbf{R_{EL}} \\ \mathbf{R_{CE}} & 0 & \mathbf{R_{CS}} & \mathbf{R_{CI}} & \mathbf{R_{CP}} & \mathbf{R_{CQ}} & \mathbf{R_{CN}} & \mathbf{R_{CL}} \\ \mathbf{R_{SE}} & \mathbf{R_{SC}} & 0 & \mathbf{R_{SI}} & \mathbf{R_{SP}} & \mathbf{R_{SQ}} & \mathbf{R_{SN}} & \mathbf{R_{SL}} \\ \mathbf{R_{IE}} & \mathbf{R_{IC}} & \mathbf{R_{IS}} & 0 & \mathbf{R_{IP}} & \mathbf{R_{IQ}} & \mathbf{R_{IN}} & \mathbf{R_{IL}} \\ \mathbf{R_{PE}} & \mathbf{R_{PC}} & \mathbf{R_{PS}} & \mathbf{R_{PI}} & 0 & \mathbf{R_{PQ}} & \mathbf{R_{PN}} & \mathbf{R_{PL}} \\ \mathbf{R_{QE}} & \mathbf{R_{QC}} & \mathbf{R_{QS}} & \mathbf{R_{QI}} & \mathbf{R_{QP}} & 0 & \mathbf{R_{QN}} & \mathbf{R_{QL}} \\ \mathbf{R_{NE}} & \mathbf{R_{NC}} & \mathbf{R_{NS}} & \mathbf{R_{NI}} & \mathbf{R_{NP}} & Rnq & 0 & \mathbf{R_{NL}} \\ \hline \mathbf{R_{LE}} & \mathbf{R_{LC}} & \mathbf{R_{LS}} & \mathbf{R_{LI}} & \mathbf{R_{LP}} & \mathbf{R_{LQ}} & \mathbf{R_{LN}} & 0 \end{array}\right] \tag{5.21}$$

Given that the locations matrix may have over 20 million elements, it is neglected from the adjacency matrix as indicated by the partition. The previously unknown relational

matrices such as $\mathbf{R_{IQ}}$ may be calculated using the procedure given in equation 5.19.

Those matrices that are previously undefined follow the notation used in this work. Therefore, $\mathbf{R_{CN}}$ is the relation between components and nodes and $\mathbf{R_{SN}}$ the relation between signals and nodes. The calculated relations may be used when building and analysing the DCSs. Not all the newly calculated relations are useful for this purpose.

The adjacency matrix is more easily comprehended using the form shown in figure 5.5.2. The relational matrix $\mathbf{R_{SQ}}$ is found at the 'Signals' row and 'Task' column of the matrix

The matrices surrounded by heavy solid lines are those corresponding to the structure of the engine and control system; they are fixed for the duration of the optimisation. Those surrounded by a heavy, dashed line are populated by the GA and represent architectural decisions.

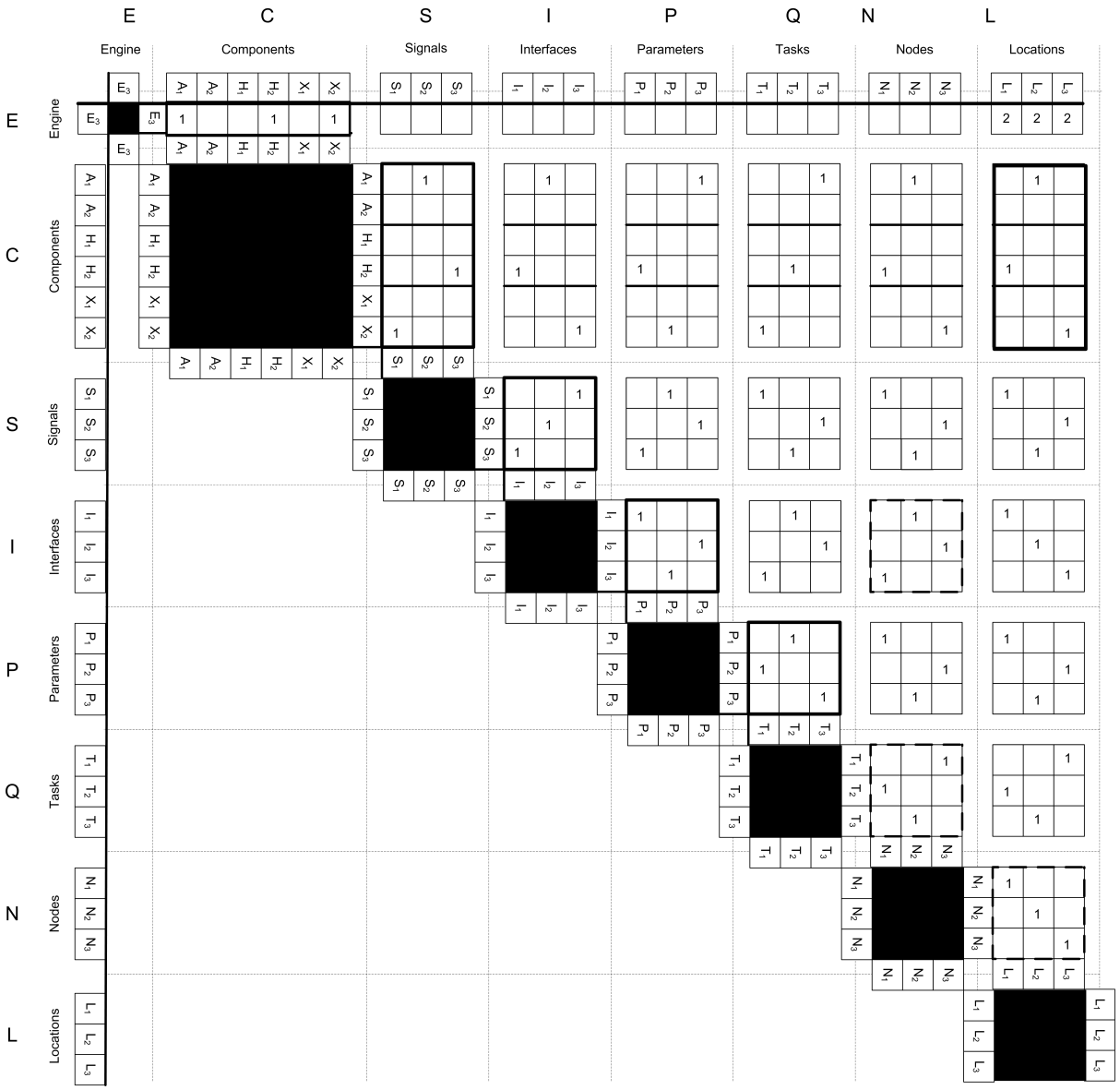


Figure 5.5.2: Complete Hybrid Framework Matrix Structure

$$\mathbf{R_{CP}} = \mathbf{R_{CS}}\mathbf{R_{SI}}\mathbf{R_{IP}} \tag{5.22}$$

The hybrid framework may be used as a map for establishing relationships between arbitrary pairs of entities. If the UML diagram is considered to be a graph, a derived relationship may be composed from the relationships on the walk from the first to the second entity. For example, (with reference to figure 5.4.20) supposing we wish to establish the relationship between signals and parameters - the walk between signals and parameters goes from signals to interfaces and then from interfaces to parameters. Therefore, the relationship may be calculated as:

$$R^{(SP)} = R^{(SI)} \circ R^{(IP)} \equiv \mathbf{R_{SP}} = \mathbf{R_{SI}}\mathbf{R_{IP}} \tag{5.23}$$

Similarly, the relationships between components and interfaces may be calculated by:

$$\mathbf{R_{CI}} = \mathbf{R_{CS}}\mathbf{R_{SI}} \tag{5.24}$$

The process of composition relies on the transitivity of the combined relations. It should be noted that (with reference to the graph of the composition) there are two possible walks between interfaces and tasks: $\mathbf{R_{IQ}}^{(1)} = \mathbf{R_{IN}}\mathbf{R_{NQ}}$ and $\mathbf{R_{IQ}}^{(2)} = \mathbf{R_{IP}}\mathbf{R_{PQ}}$. The two paths do not produce the same result but reveal different information about the system. $\mathbf{R_{IQ}}^{(1)}$ reveals the tasks that could no longer be performed if a certain node fails and $\mathbf{R_{IQ}}^{(2)}$ reveals those tasks that could not be performed if the necessary parameter were unavailable - such an occurrence may be due to network or node failure. The new relation and the implications of the corresponding relationships are not necessarily equivalent. Multiple walks or paths are an inherent limitation of the adjacency matrix as presented in figure 5.5.2 and a justification for using the computationally expensive matrix multiplication to derive relations rather than the Floyd-Warshall algorithm. The Floyd-Warshall algorithm would find the 'shortest' walk between the two vertices. If two (or more) walks between the vertices were of equivalent length, the algorithm would retain the first walk encountered.

The composition process may require inverse relations if walks through the graph move, 'backwards'. In most cases, the relations used in this application are symmetric. The inverse relation is the transpose of the relational matrix.

$$\boldsymbol{R_{BA}} = \boldsymbol{R_{AB}}^{T} \quad \text{if the relation } R^{(AB)} \text{ is symmetric} \tag{5.25}$$

It should be noted that the lexical translation of a symmetric relation may not itself be symmetric. For example, the relation [one] **node** performs [one to many] **tasks** is nonsensical if read as [one to many] **tasks** performs [one] **node**. The inverse relation is commuted to are performed by to preserve the logic.

These relations are referred to during the discussion on architecture composition and evaluation which follows. Those relations that are not derived by composition are stored in the relational database.

5.6 Architecture Construction

Each of the GA's chromosomes provides a coded representation of a DCS architecture. The architecture construction processes (see figure 5.6.1) translates this chromosome into a architectural blueprint which defines the attributes of a metaphysical DCS. This model may be evaluated and visualised. The blueprint is an instance of the adjacency matrix $\boldsymbol{A_{sys}}$ for the specific DCS. The operation is repeated for every member of the current population, (P) and performed in two stages: the chromosome is decoded to reveal a blueprint and subsequently, that blueprint is used as the basis for building a metaphysical DCS. The process is illustrated below:

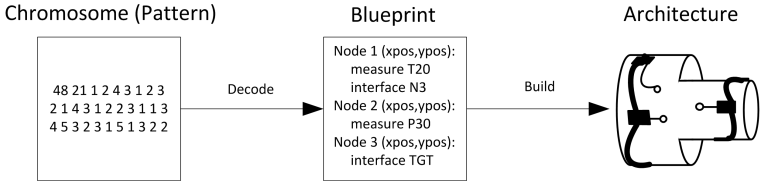


Figure 5.6.1: Process of moving between the chromosome and architecture

The metaphysical model has a defined object-oriented structure based on the architecture frameworks. The structure of the chromosome, blueprint and metaphysical model are discussed in subsequent sections. Each exist only as a software construct.

Intuitively, the 'decode', 'construct' and 'evaluate' functions of the GA are contiguous in time - the process of constructing an architecture cannot commence before its respective chromosome is decoded. However, this linearity is obfuscated by the object-oriented implementation of the metaphysical model. Each major element is represented within the model as the instantiation of a pre-defined class. Each instantiated object has associated operations to decode, construct and evaluate that particular element of the model. Therefore, the model is constructed in a 'parallel' fashion and evaluated sequentially.

5.7 The Chromosome

The chromosome codes the various parameters listed in section 5.3. The string's size changes depending on the number of nodes but remains constant for the duration of a single pass. A diagram of the chromosome is given in figure 5.7.1 below. Each box in the diagram represents a single bit:

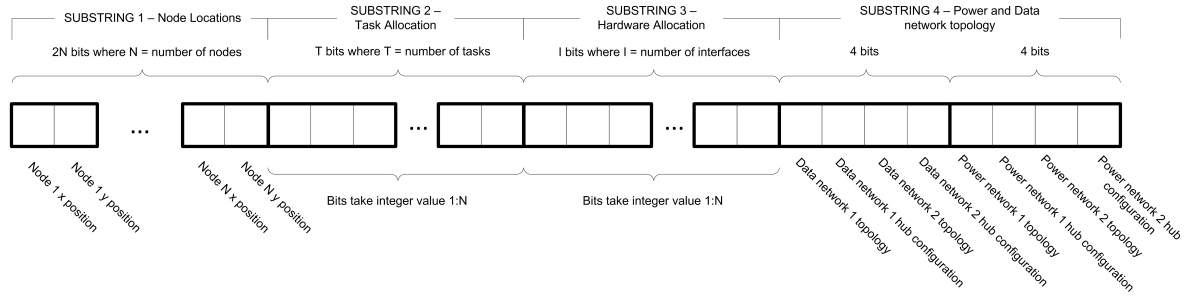


Figure 5.7.1: The chromosome used to specify a DCS. The contents of the chromosome determine the location of the nodes, the tasks performed by each node, the location of the hardware and the power and network topology.

In many applications, the chromosome is a binary representation of a number or solution. In this application the chromosome stores mainly integer values. Although not technically precise, we refer to the chromosome as a string and to each location as a 'bit'.

The string is divided into four substrings and is designed to prevent the creation of infeasible solutions.

The first substring encodes the node positions. Each node is allocated a pair of bits which reference engine chassis stations using cartesian coordinates. The first bit contains an integer value for the x-coordinate and the second bit, an integer value for the y-coordinate. The coordinates reference the elements $\boldsymbol{L_{(x,y)}}$ of the chassis station matrix $\boldsymbol{L}$ and correspond to a particular chassis station. The length of the substring b_N is dependant on the number of nodes under consideration: $b_N = 2|N|$. The value of these bits cannot exceed the number chassis stations in the x and y direction respectively.

The second substring string allocates system tasks to specific nodes. The integer value stored in a bit dictates the node to which that task is allocated. Therefore, if the 5th bit of the second substring has the value 3, then task 5 is allocated to node 3. The number of bits in the second substring b_Q equals $|Q|$. The value associated with a bit must be a node identifier *i.e. bit value* $\in N$.

Allocating interfaces (CCBs) to nodes is performed exactly as for tasks - each interface is assigned a node number within the third substring. The number of bits required for the interface allocation, b_I is $|I|$.

The fourth substring contains 8 bits and determines the network and power topology. Each bit is populated by an integer value (1..4) which infers a certain data network and power topology. Data networks and power system topologies are described in section 5.12.

Therefore, the total length of the chromosome is given by:

$$b_l = b_N + b_Q + b_I + 8 \tag{5.26}$$

For a system with 5 nodes, 65 interfaces and 94 tasks, the chromosome is 178 bits long. One binary string is required for each member of the GA's population.

5.8 The Blueprint - Decoding the chromosome

The decoded the binary string provides the blueprint for the architecture of a DCS. The blueprint does not specify the complete architecture but provides a basis from which the full architecture may be derived. For example, the blueprint specifies the location of the nodes but does not consider node size or harness lengths and routing. The node size and harness routes are, 'derived' by independent algorithms during the architecture construction process. The blueprint itself contains no derived information; it simply represents the information coded in the chromosome:

- Location of each node
- Interface circuitry hosted by each node
- Tasks performed by each node
- Power network topology
- Data network topology

Each of the first three substrings may be expressed as a relational set and an equivalent matrix. Each ordered pair relates one element of the system to another, *eg.* a task to a node or a node to a specific engine location. The relationship between nodes and engine chassis locations indicates the position of nodes on the engine. The node location (as a chassis station number) is determined using the MATLAB® function `l = sub2ind( L, y, x )` which returns a chassis station number from the x and y coordinates given in the first substring.

$$R = \{ (n,l) \mid n \in N,\ l \in L \} \tag{5.27}$$

$${}_nR_l : n \underline{\textit{ is placed at location }} l \tag{5.28}$$

Where $N = \{ n_1, n_2, \ \ldots \ , n_\eta \}$ is a set of nodes in the current system and L is a set of all chassis stations on the engine $\{1, 2, \ldots, xy\}$.

An instance of the relation is created for every node.

The relationships between interfaces and nodes are populated from the second substring.

$$R = \{ (i,n) \mid i \in I,\ n \in N \} \tag{5.29}$$

$${}_iR_n : i \ \underline{is\ hosted\ on}\ n \tag{5.30}$$

where $N = \{ n_1, n_2, \ \ldots\ , n_\eta \}$ is a set of nodes in the current system and I is a set of interfaces required by the control system $I = \{ i_1, i_2, \ldots, i_{|I|} \}$.

An instance of the relation exists for every interface in the system. The node number for the i-th interface is the value in the i-th bit of the second substring.

Similarly, the third substring is related to the the allocation of tasks to nodes. This relationship is given by equation 5.32:

$$R = \{ (q,n) \mid q \in Q,\ n \in N \} \tag{5.31}$$

$${}_qR_n : q \ \underline{is\ performed\ by}\ n \tag{5.32}$$

Where $Q = \{ q_1, q_2, \ \ldots\ , q_{|Q|} \}$ is a set of tasks which are required for the given engine. As for interfaces, an instance of the relation is created for every task in the system. The node allocation for the q-th interface is the value found in the q-th bit of the second substring.

The values associated with network and power topologies are written directly to variables of the solution class.

5.8.1 Matrix form and Relation Composition

The second stage of decoding the chromosome is to translate these relations into their matrix form - That is, to place a '1' in the corresponding elements of the relationship matrices where an instance of the relation occurs.

For example, if there is a relationship between interface 1 and node 2 and a further relationship between interface 3 and node 1, the interface-node relationship matrix, $\mathbf{R_{IN}}$ is populated as follows (assuming that the system has three nodes and the product family contains three interfaces:

$$\mathbf{T_{IN}} = \begin{bmatrix} {}_{i_1}R_{n_1} & {}_{i_1}R_{n_2} & {}_{i_1}R_{n_3} \\ {}_{i_2}R_{n_1} & {}_{i_2}R_{n_2} & {}_{i_2}R_{n_3} \\ {}_{i_3}R_{n_1} & {}_{i_3}R_{n_2} & {}_{i_3}R_{n_3} \end{bmatrix} \quad (5.33) \qquad \mathbf{R_{SI}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (5.34)$$

Similarly, the relationships between nodes and tasks and nodes and locations are entered into the matrices $\mathbf{R_{QN}}$ and $\mathbf{R_{NL}}$..

Once the relations are expressed as matrices, these matrices become part of the system adjacency matrix (figure 5.5.2). The relation composition process (section 5.5.1) is used to calculate other relationships used during construction and evaluation of the architectures. For example, the relation between signals and nodes, $\boldsymbol{R_{SN}}$ is used for harness routing:

$$\boldsymbol{R_{SN}} = \boldsymbol{R_{SI}}\boldsymbol{R_{IN}} \quad (5.35)$$

The relation between parameters and nodes, $\boldsymbol{R_{PN}}$ is used by the architectural evaluation to determine the number of parameters that are placed on the databus:

$$\boldsymbol{R_{PN}} = \boldsymbol{R_{PI}}\boldsymbol{R_{IN}} \quad (5.36)$$

Similarly, the relations $\boldsymbol{R_{SQ}}$ and $\boldsymbol{R_{IQ}}$ are used by the lifecycle evaluation to determine how the system performs to component failures during its service life.

Although not used in this implementation, the relationship between interfaces and locations, $\boldsymbol{R_{IL}} = \boldsymbol{R_{IN}}\boldsymbol{R_{NL}}$ could be used as a reference for quickly setting a component's reliability according to its temperature and vibration environment. Other derived relationships such as components-tasks and signals-parameters could be used to analyse the level of redundancy in the the system and the response to component failures.

These new relations become part of the adjacency matrix $\boldsymbol{A_{sys}}$.

At this point, the blueprint is complete and exists as the system adjacency matrix $\boldsymbol{A_{sys}}$. The blueprint is used as a basis for constructing the distributed control systems.

5.9 Redundancy Scheme

It would be almost impossible to achieve the required standards for safety and reliability without applying some level of redundancy in the engine control system. Further to

safety requirements, Time Limited Despatch (TLD) *(see section 7.5)* relies on redundant elements to minimise service disruption and operating costs. Conventionally, EECs have two redundant channels - if one channel experiences a component failure (or set of component failures) then the second channel takes control of the engine. Sensors and actuators are associated with a single channel. Some critical components such as shaft speed probes are dual redundant to both channels meaning the engine carries four identical sensors for one measurement (this may constitute two dual wound devices).

A notable benefit of Distributed Control Systems is the potential for novel redundancy schemes which reduce the number of components required and increase the number of redundant configurations. Whilst the possibilities are largely intuitive, the area requires a substantial amount of research and development to analyse and improve on the basic postulates.

The basic premise of distributed redundancy is that control system elements become redundant to the network rather than the channel. Rather than a distinct 'A' and 'B' channels, there are 'A' and 'B' instances of sensors, actuators, tasks and computational hardware. If the 'A' instance fails then the 'B' instance takes over. Neither instance need belong to a distinct channel. The two schemes are compared in the figures below:

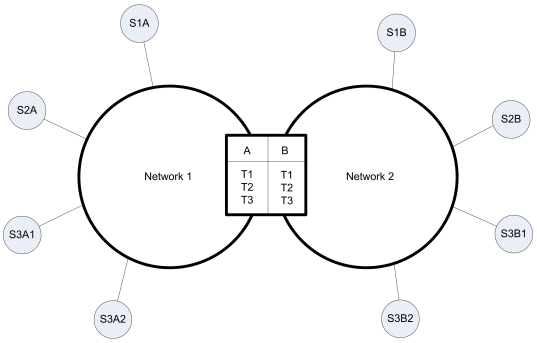


Figure 5.9.1: Centralised system using conventional approach to redundancy. All tasks are performed on the centralised controller

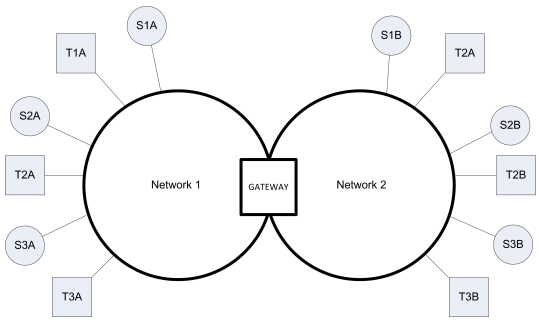


Figure 5.9.2: New redundancy approach using a distributed system. Control system tasks are shared amongst nodes

Under the distributed scheme, some the components that are dual redundant to a channel can be removed allowing three devices to provide a similar availability to four. *eg.* S3A2 could be removed. The two networks may operate as separate channels or become a single network linked by the gateway. This research assumes use of the redundancy scheme in figure 5.9.2. The 'A' and 'B' instances of control system tasks may be performed on different nodes. The 'A' and 'B' networks for power and data need not be identical. A more sophisticated network architecture would use two dissimilar networks to form

a dual-channel surviving-element network where the networks combine to form a new topology should a branch break or fail.

5.10 Node Construction

As defined by the architecture framework, nodes are deemed to be the physical aggregation of a Central Processing Unit (CPU), power conditioning, signal and data bus connectors, a case, mountings and a number of circuit blocks.

The significant tasks of node construction are:

1. Selecting and assembling of the CCBs
2. Adding Signal Connectors
3. Adding data, power connectors and case.
4. Dimensioning the node (to establish height, width, floor area, weight etc)

Stage 1: Selection and assembly of Common Circuit Blocks

The first phase of node construction (figure 5.10.1) is to assemble the CCBs. The interfaces allocated to each node are determined from the matrix $\mathbf{R_{IN}}$. Each mark in the column corresponding to the current node represents a relationship between that node and a given circuit block. The circuit blocks allocated to each node are determined using the MATLAB® command `index = find( R_in(n,:) > 0 )` where the returned variable `index` is the subset of interfaces (referenced by their primary key - see appendix A) pertaining to the node `n`.

All nodes have two channels and two instances of every interface must be present in the system (see section 5.9 on redundancy schemes). Nominally, every interface allocated

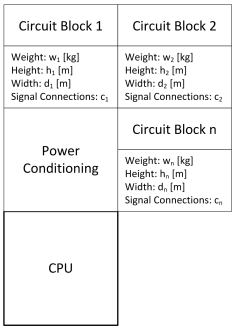


Figure 5.10.1: Stage 1: Assemble CCBs

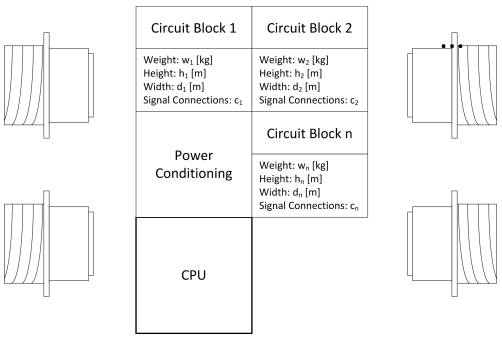


Figure 5.10.2: Stage 2: Add signal connectors

to a node is placed in the 'A' channel. If both instances of an interface are allocated to the same node, then the first is allocated to the 'A' channel and the second to the 'B' channel. *i.e.* if all the interfaces on a node are unique, the node has no 'B' channel.

Each circuit block has a, depth, weight, cost and parameters to define its reliability (discussed later in section 7.7.1).

The width of a node is largely determined by the dimensions of the circuit blocks and the node depth and height by the circuit blocks or connectors depending on which is larger. Each CCB has an associated area. The internal height and width of a node n_h, n_w are approximated to the square root of the sum of circuit block areas. This is a crude approximation but avoids the need for complex tessellation algorithms to optimise the arrangement of circuit blocks within a node. Therefore, the n_h and n_w are given by equation 5.37:

$$n_h = n_w = \sqrt{\sum_{k=\texttt{index}(1)}^{\texttt{index}(|\texttt{index}|)} \texttt{interface[k].area}} \tag{5.37}$$

This height and width may be thought of as the height and width of the node's circuit board which is always square.

The reliability of a CCB changes with temperature which is a function of the node's location. Obtaining reliability data for high-temperature components is very difficult. This research assumes that all electronic devices are able to work at all temperatures on the engine case. The penalty for using high temperature components is reduced reliability and higher cost. In reality, no such set of components is available and the immaturity of many high-temperature devices means that component manufactures are both secretive and pessimistic about reliability. As with the engine temperature profiles, this research uses tangible reliability profiles that are guided by but not aligned to real-world values. If real data were available, it could be used within the models. Furthermore, reliability data on component datasheets is usually presented over a narrow temperature bands on a primitive log-scale graphs and consequentially, difficult to read and reproduce accurately.

Reliability characteristics are established by extending the straight-line characteristic from component datasheets over a wider temperature range. The figure below (5.10.3) shows the original data points taken from a datasheet for a high-temperature operational amplifier. A straight line approximation is used to extend the reliability profile.

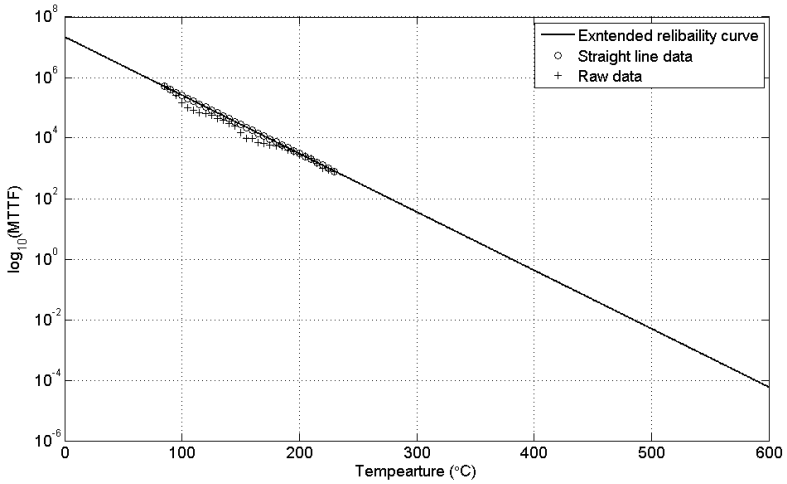


Figure 5.10.3: Extension of component reliability data to cover broader temperature range

The Mean Time To Failure (MTTF) read from the graph is used to determine the reliability characteristic of each interface and hence the node.

Stage 2: Add Signal Connectors

The second stage of node construction (figure 5.10.2) is to add the signal connectors. The connector size and weight are an important considerations in node design. Connectors are specified for the number of signals and the addition of guard pins and spares. Connectors suitable for the engine environment tend to be heavy and space consuming as they are required to be scoop-proof, thread-proof and useable whilst wearing gloves. Therefore, both connector weight and size are likely to dominate the node dimensions. It is assumed that all node connectors are flange-mounted bulkhead connectors.

Weight is not generally specified in connector data sheets as it varies from part-to-part, depending on the number of pins, the materials used and the presence of passive filters within the connector assembly. The optimisation processes ascertains how many connector pins are required by counting the number of cores in a harness and adding 10%. Typically an LVDT may require an eight-wire connection and a thermocouple a two-wire connection. Rather than using a lookup table to find the nearest suitable connector, the weight is taken from a second order polynomial fitted to weight data held by Aero Engine Controls.

The curve is fitted to the data using the MATLAB® command `p = polyfit(x,y,N)` which uses least squares method to return the coefficients of a polynomial, `p` of order

`N` fitted to the data contained in the vectors `x` and `y` - a second order polynomial is considered sufficient. Therefore, the connector weight, c_ψ as a function of the number of pins σ is given by equation 5.38.

$$c_\psi(\sigma) = \mathtt{p}_1\sigma^2 + \mathtt{p}_2\sigma + \mathtt{p}_3 \tag{5.38}$$

The maximum connector size is 61 pins. The fitted curve and original data points are shown in figure 5.10.4.

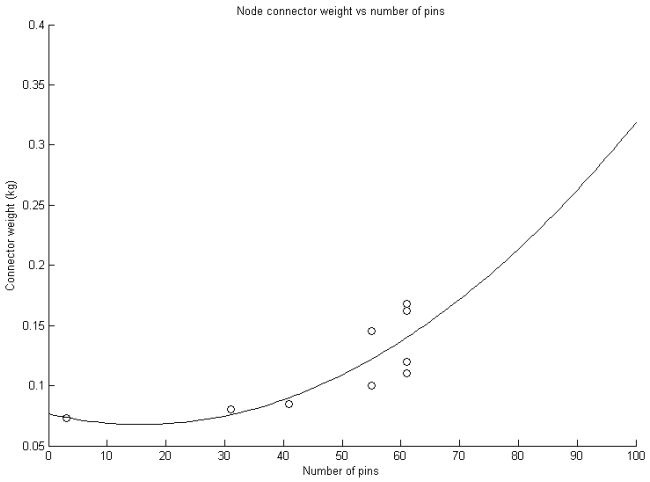


Figure 5.10.4: Connector weight vs number of pins showing original data and second order fitted polynomial

If more than 61 pins are required for a single channel, further connectors are added. The node always uses the smallest possible connectors and assumes that connectors of all sizes are available (up to 61 pins). A similar curve is used to determine the weight of harness connectors.

The number of pins required for the signal connector is determined by summing the number of signals associated with each of the node's interfaces. This information can be derived from the matrix $\mathbf{R_{SN}}$. Therefore, the number of pins σ required for a given connector may be found using the MATLAB® command `sigma = sum( find( R_sn(:,n) > 0 ) )`.

The diameter of the connector is also dependent on the number of pins. It is probable that the size of the connectors will dictate the size of the node itself. Pins in industrial connectors are generally arranged in concentric circles from a single pin in the centre to perhaps 30 or 40 pins in the outer layers. For the purposes of this work, the concentric

circles are approximated to concentric hexagons and hexagonal numbers are used to find the number of pin layers and hence the connector dimensions (figure 5.10.5).

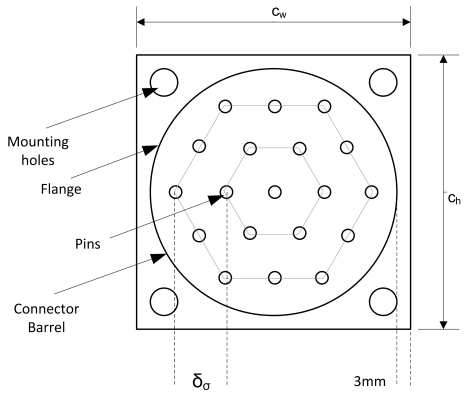


Figure 5.10.5: A bulkhead connector showing dimensions and the hexagonal approximation to concentric circles

Equation 5.39 uses hexagonal numbers to calculate the number of concentric pin layers σ_l as a function of the number of pins σ.

$$\sigma_l = \left\lceil \frac{\sqrt{8\sigma + 1} + 1}{4} \right\rceil \tag{5.39}$$

It is assumed that the connector and bulkhead are square and that connector width c_w and height c_h are equal. Assuming that there is a 3mm section of the flange plate on both sides of the connector barrel and that the distance between layers of pins δ_σ is known, the width of the connector is found using equation 5.40.

$$c_w = c_h = 0.06 + \delta_\sigma(2\sigma_l + 1) \qquad [metres] \tag{5.40}$$

Having added the signal connectors, an instance of the harness class (section 5.11) is created for each connector. This instance is built into a full harness at the end of the node construction process. At this stage, each harness is seeded with the source (node location) and a vector of destination (engine component) locations. This information is derived from the signal-node and signal-location matrices $\boldsymbol{R_{SN}}$ and $\boldsymbol{R_{SL}}$.

Stage 3: Add data, power connectors and case

The third stage of node construction involves the addition of data connectors, power connectors and a case.

Each node channel has an independent power and data connector. Both are 2 pin connectors unless the node is a designated hub for a star network *(see section 5.12)*. Having added these connectors, the width of all the connectors is summed. This value is augmented with a connector spacing (to ensure that sufficient gap is left between each connector) to give a value for the total connector width. If the panel width is greater than the internal height of the CCBs, then the internal node height is set to the connector width. The internal node width is set to the width of the circuit blocks as determined by equation 5.37. The internal node depth is either the greatest connector depth or the depth of the largest circuit block depending on which is larger. The internal height, width and depth are incremented by 1cm to allow a 0.5cm gap to be maintained between the circuit board and case.

Having established the internal dimensions, the node case is added. The node case is assumed to be a hollow box whose inner volume contains the circuitry. At this stage, only the case volume is required. The case volume is found using equation 5.41 and is the difference between the internal and external node dimensions as separated by the node case thickness.

$$n_{case\_vol} = (n_w + 2\vartheta)(n_h + 2\vartheta)(n_d + 2\vartheta) - n_w n_h n_d \tag{5.41}$$

Where ϑ, is the thickness of the case wall (typically 5mm). All other parameters were defined previously.

Stage 4: Dimension the node

The node weight comprises the collective weight of the case, the CCBs and the signal, power and data connectors. The mass of the case may be found using the equation $mass = density(\rho) \times volume(n_{case\_vol})$. The density used is representative of the material densities used in present day casing.

The node's height, width and depth are equal to the internal height width and depth plus twice the case thickness.

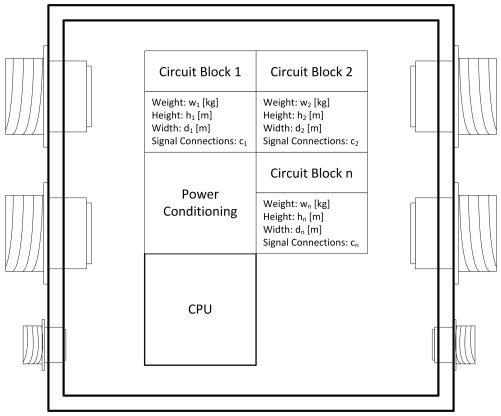


Figure 5.10.6: Stage 3: Add power connector, data connector and case

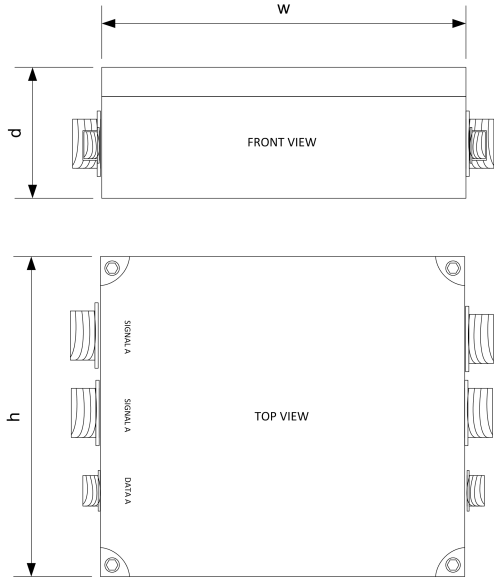


Figure 5.10.7: Stage 4: Weigh and Dimension node

At this stage the weight of the mounting feet and a bond strap are added. A floor plan area for the node is calculated. The floorplan area is the area occupied by the node plus an additional margin for component separation and handling.

The signal harnesses are built and the node is complete.

5.11 Harnesses

Each node has a harness for every signal connector and a set of harnesses are associated with both the data and power networks. The signal harnesses are independent of the data and power harnesses. The harnesses in the metaphysical model reflect the structure of real engines harnesses as closely as possible. It can be assumed that a single node has a separate set of harnesses for both channels although it is unnecessary to distinguish between channels when creating harnesses. The terminology used to describe the construction of the harness has been assumed for this project and is not aligned to industrial terminology.

A harness is composed from branches, clusters, connectors and ties *(see figure 5.11.1)*. Each branch connects to a single sensor, actuator or engine component. A branch carries a number of signal connections in groups known as clusters *(see figure 5.11.3)*. The clusters and branches are bound together using synthetic hemp ties. The ties occur at intervals of approximately 5cm along the harness's length. The whole harness is covered in a protective sleeve designed to protect against fire and debris.

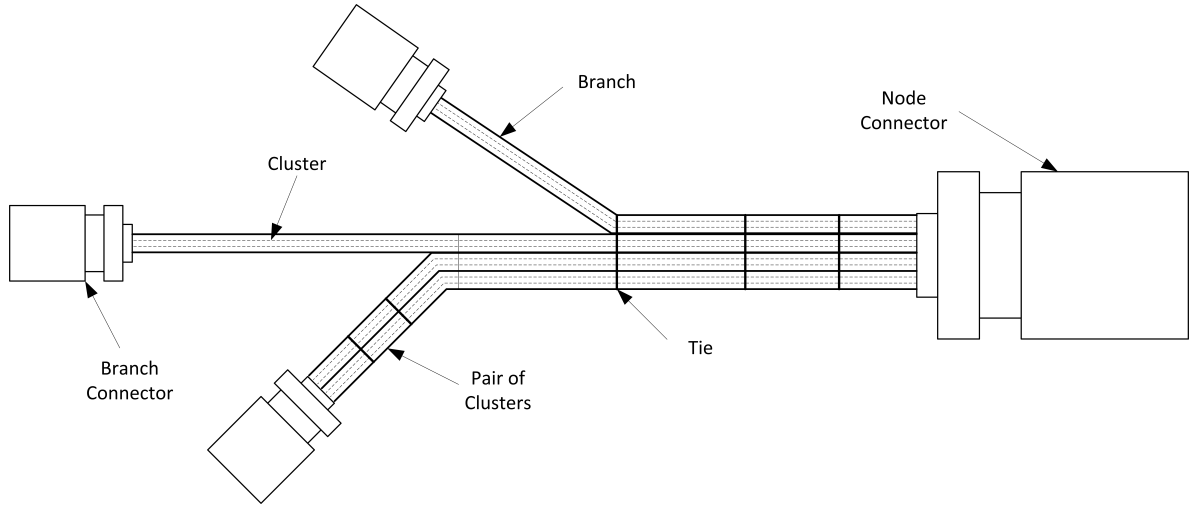


Figure 5.11.1: The structure of a harness as used in the metaphysical model showing the connectors, branches and clusters.

A cluster carries a set of wires within a protective sleeve *(see figure 5.11.2)*. Clusters have a finite capacity of two signal wires. Therefore, if a branch requires 5 signal wires, three clusters are required - the first carries two cores, the second two and the third one; signal wires are usually bound as twisted pairs. The two signal wires contained within the cluster are each coated in an insulating sleeve.

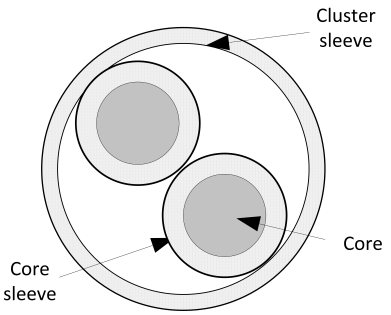


Figure 5.11.2: Cross-section of a harness cluster

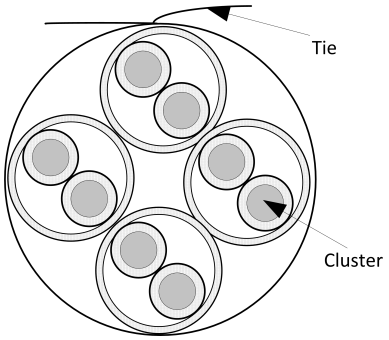


Figure 5.11.3: Cross-section of a harness branch

Like other elements in the metaphysical model, a harness is represented as the instantiation of a software class. The harness class is shown in figure 5.11.4 below:

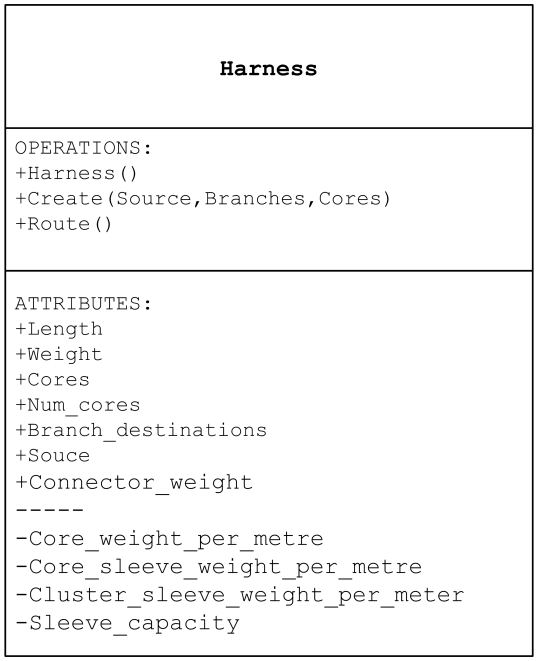


Figure 5.11.4: The harness class

The `Harness()` constructor function initiates the class and calculates the weights per metre for the copper wire, wire sleeve and cluster sleeve. The weight per mete of the copper signal wire is calculated using the following equation:

$$\psi_\sigma = \left(\frac{\theta_c}{2}\right)^2 \pi\rho_{cu} \qquad [kgm^{-1}] \tag{5.42}$$

Where ψ_σ the mass per meter of signal wire, θ_c is the diameter of the harness core in metres and ρ_{cu} is the density of copper in kilograms per cubic metre. The density of copper is taken to be $8700kgm^{-3}$.

The weight per metre of the core sleeve ψ_s and cluster sleeve weight ψ_c may be obtained using similar equations. Both the core sleeve and cluster sleeve are considered to be made of a plastic with a density of $900kgm^{-3}$. The figures calculated above are assigned to the object attributes `Core_weight_per_metre`, `Core_sleeve_weight_per_metre` and `Cluster_sleeve_weight_per_metre`. The attribute `Cluster_capacity` holds the number of cores per cluster - the default value is 2. The various densities, sleeve thicknesses and radii are hard-coded into the class.

The public function `Create()` uses the source of the harness, the branch destinations and the number of signal cores per branch to determine the the length and weight of the harness. The source and branch destinations are passed to the `Create` function as a

vector of engine station numbers.

The length of each branch is determined using the matrix $\boldsymbol{D}$ *(see section 5.4.4)* as a look-up table. Once the length of the branch has been established, the number of cores on the branch is taken from the input vector and divided by two to calculate the number of clusters required for the branch. The wire and sleeve weights for a single branch are calculated using these parameters. All branches are extended by 30cm to allow slack for handling and connection. The weight of the harness connector is taken from a fitted curve *(see section 5.10)* and added to the total weight of the branch.

Once all the branches have been created, the total harness length and weight is calculated. The weight of the ties is assumed to be negligible and therefore ignored.

By assuming that the harness will always take the shortest possible route, the weight and length may be calculated without knowing the path taken. The distance is read from the $\boldsymbol{D}$ matrix. The route itself can be determined using the Floyd-Warshall Algorithm. The `Route()` function of the harness uses Floyd-Warshall algorithm to determine the route and returns a vector of engine stations denoting the path taken. The routing process is described in section 5.11. This algorithm is computationally expensive and only necessary for visualising the final design.

Harness Routing

Harness routing is achieved using the 'Next' matrix $\boldsymbol{N}$ which is generated at the same time as the distance matrix $\boldsymbol{D}$. The Next matrix acts as a look up table for finding harness routes. For example, if we wish to route a harness from chassis station 2 to chassis station 10, then the element $\boldsymbol{N}_{(2,10)}$ holds the next chassis station on the route. *i.e.* If $\boldsymbol{N}_{(2,10)} = 7$ then the next station on the route is the value of $\boldsymbol{N}_{(7,10)}$. The process is repeated until the index matches the destination.

5.12 Data and Power Network Topologies

The final eight bits of the chromosome encode the topology for the power and data networks. Each DCS has two power networks and two data networks; all four networks have their own topology. The topology of each network is determined by two bits; both hold integer values of 1 to 4.

The first of the two bits determines the topology of the network and the second the configuration of that topology. If the chosen network topology requires a network hub, then the second bit determines where that hub is placed:

1. Star
2. Ring
3. Bus
4. Direct Feed

1. Hub at Jordan centre
2. Data Over Power (DOP)
3. Node closest to the pylon
4. Hub in the airframe pylon

All data networks are required to interface to the data connection at the pylon. The node selected as a hub provides that interface. The hub node requires a larger connector and carries the additional weight of the hub circuitry. The hub may be located in the pylon although this incurs a high difficulty score. The four basic network topologies are illustrated below:

The Data Over Power (DOP) option signifies that the data network is neglected and all data is sent between nodes by modulating the power supply voltage. This arrangement adds to the difficulty factor but saves weight by removing harness and connectors. Data over power is an important consideration for DCS architects and has been the focus of industrial research projects.

The same topologies apply to the power network although a further configuration option is available. Option 2 in the configuration bit determines that the power hub will be placed in the node nearest to the Permanent Magnet Alternator (PMA). The control system requires power feeds from the pylon and the PMA. This power may be distributed directly to each node or be managed and distributed by a power hub located in a single node. The node containing the power hub must connect to both the airframe and PMA power supplies. The various network configurations are shown in figure 5.12.2:

Various graph theory algorithms are used to determine the hub location and arrangement of harness connections. The configuration bit determines which node will host the hub for the star, ring and bus topologies and hence how the harnesses are routed. Configuration parameter = 1 places the hub at the Jordan centre of the graph formed by the nodes, pylon and PMA. The Jordan centre is the most 'central' of the nodes with respect to the set of nodes.

The algorithms which determine the hub position use a subset of the distance matrix $\boldsymbol{D}$

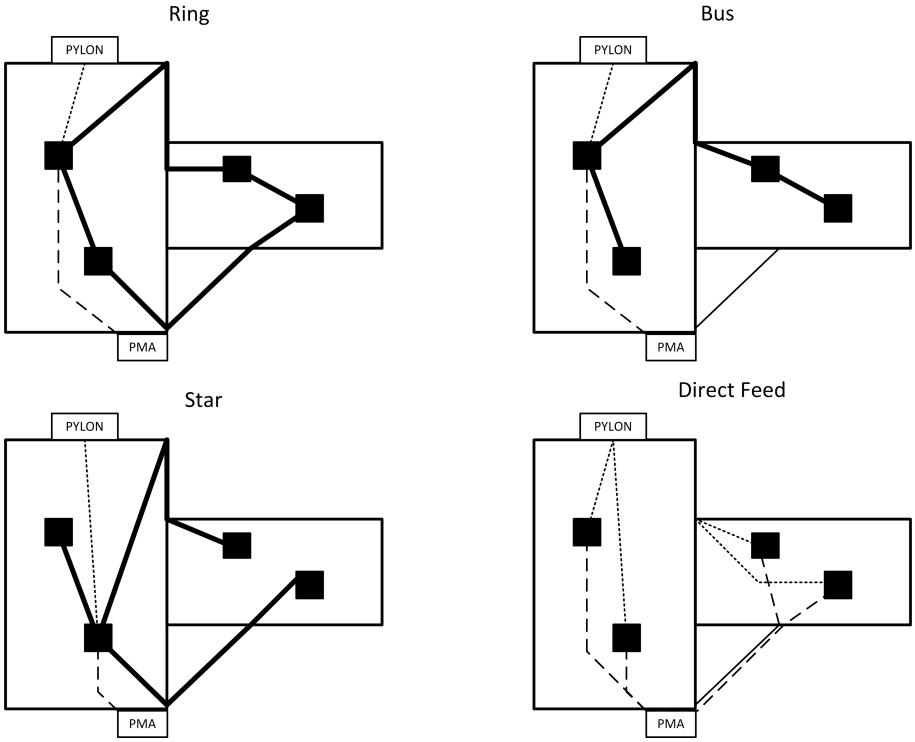


Figure 5.12.1: Data network topologies

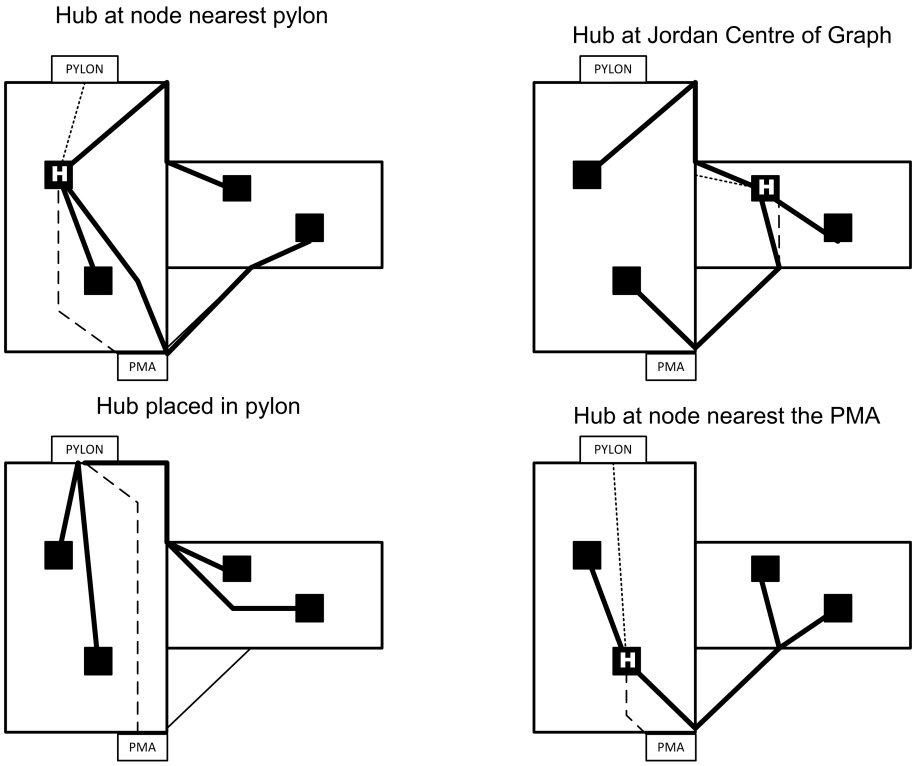


Figure 5.12.2: Power network hub configurations. The nodes marked with an 'H' contain the hub circuitry and connectors. The configurations also apply to data networks although no connection to the PMA is required

(see section 5.4.4) where the rows and columns correspond to the nodes $\{n_1, n_2, \ldots, n_n\}$, airframe a and the PMA p. The edge lengths are the shortest distance between each vertex as defined in the distance matrix $\boldsymbol{D}$. The distance matrix for the power networks $\boldsymbol{D_p}$ takes the following form:

$$\boldsymbol{D_p} \triangleq \begin{array}{c} \\ n_1 \\ n_2 \\ \vdots \\ n_n \\ a \\ p \end{array} \begin{array}{c} \begin{array}{cccccc} n_1 & n_2 & \ldots & n_n & a & p \end{array} \\ \begin{pmatrix} 0 & \boldsymbol{D}_{(n_1,n_2)} & \ldots & \boldsymbol{D}_{(n_1,n_n)} & \boldsymbol{D}_{(n_1,a)} & \boldsymbol{D}_{(n_1,p)} \\ \boldsymbol{D}_{(n_2,n_1)} & 0 & \ldots & \boldsymbol{D}_{(n_1,n_n)} & \boldsymbol{D}_{(n_2,a)} & \boldsymbol{D}_{(n_1,p)} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \boldsymbol{D}_{(n_n,n_1)} & \boldsymbol{D}_{(n_n,n_2)} & \ldots & 0 & \boldsymbol{D}_{(n_n,a)} & \boldsymbol{D}_{(n_n,p)} \\ \boldsymbol{D}_{(a,n_1)} & \boldsymbol{D}_{(a,n_2)} & \ldots & \boldsymbol{D}_{(a,n_n)} & 0 & \boldsymbol{D}_{(a,p)} \\ \boldsymbol{D}_{(p,n_1)} & \boldsymbol{D}_{(p,n_2)} & \ldots & \boldsymbol{D}_{(p,n_n)} & \boldsymbol{D}_{(p,a)} & 0 \end{pmatrix} \end{array} \tag{5.43}$$

A distance of zero dentes that there is no connectivity between vertices. The distance matrix for the data network $\boldsymbol{D_d}$ is similar to $\boldsymbol{D_p}$ but does not require the additional vertex p for the PMA.

The closest node to the airframe connection or PMA is simply the node which corresponds to the minmum value in the column a or p in $\boldsymbol{D_p}$. The Jordan Centre is found using the ‘‘all-pairs shortest path’’ algorithm (Seidel, 1992). Once determined, the appropriate node is augmented with the necessary hardware and connectors to add the hub functionality.

The connectivity of the network harnesses is found using the FANI algorithm (Ravikumar *et al.*, 1998). FANI is a graph insertion algorithm capable of finding the shortest possible walk between a set of vertices. The algorithm can be used to determine the arrangement of harnesses for both the ring and bus networks and is commonly associated with the travelling salesman problem. The star topology simply connects the hub node to every other node and does not require an algorithm to determine connectivity.

The FANI algorithm returns a network adjacency matrix $\boldsymbol{A_n}$. For example:

$$\boldsymbol{A_n} = \begin{array}{c} \\ n_1 \\ n_2 \\ \vdots \\ n_n \\ a \\ p \end{array} \begin{array}{c} \begin{array}{cccccc} n_1 & n_2 & \dots & n_n & a & p \end{array} \\ \begin{pmatrix} 0 & 1 & \dots & 1 & 0 & 1 \\ 1 & 0 & \dots & 1 & 0 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & 1 & \dots & 0 & 1 & 0 \\ 0 & 0 & \dots & 1 & 0 & 1 \\ 1 & 1 & \dots & 0 & 1 & 0 \end{pmatrix} \end{array} \tag{5.44}$$

A '1' indicates that the vertices should be jointed by data or power harness depending on which network is being routed. The matrix is symmetrical across the diagonal. An instance of the harness class is created for every '1' in the upper diagonal. As with signal harnesses, the harnesses have a weight. Once the topology is established, the harnesses are routed in the same manner as signal harnesses.

At this stage, the architectural is complete.

5.13 Visualisation

A basic software application allows the metaphysical architectures to be visualised. Visualisation is important for validation and maintaining industrial interest. The software allows the following items to be visualised:

- Engine case and outline
- Location of sensors and actuators
- Location and size of engine subsystems
- Distributed node locations
- Wiring harnesses between nodes and components
- Databus harness between nodes
- Temperature profiles
- Accessibility profile
- Keepout zones
- Vibration profiles
- Engine station locations
- Power Harnesses

The tool offers a 3D view of the control system as it would look on engine. The user may rotate the engine and zoom in and out using keyboard commands. Each of the items

listed above is a layer in the tool; as with professional Computer Aided Design (CAD) tools, the user may turn various layers on and off to improve clarity.

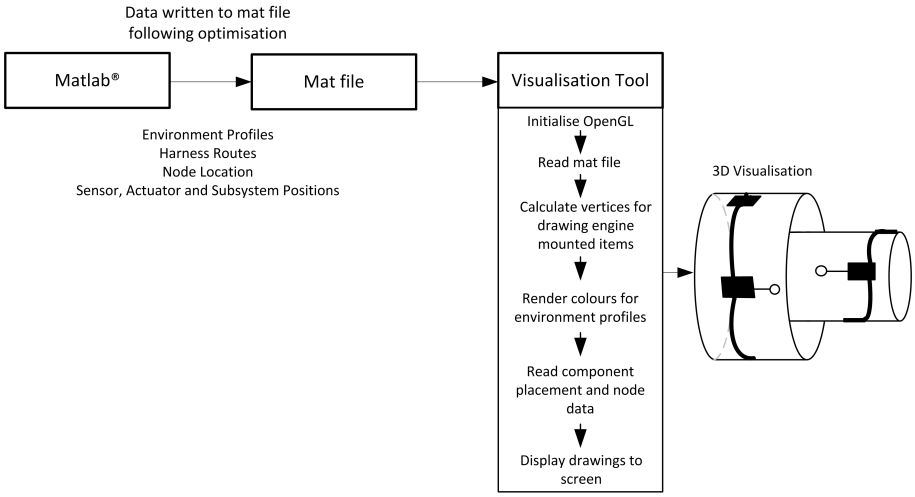


Figure 5.13.1: High level functionality of the visualisation tool

Once the metaphysical architecture has been composed by the GA, data files containing positional, dimensional and environmental profile data are created in MATLAB®. The visualisation tool reads and interprets the information in these files to construct a graphical representation of the architecture as shown in figure 5.13.1.

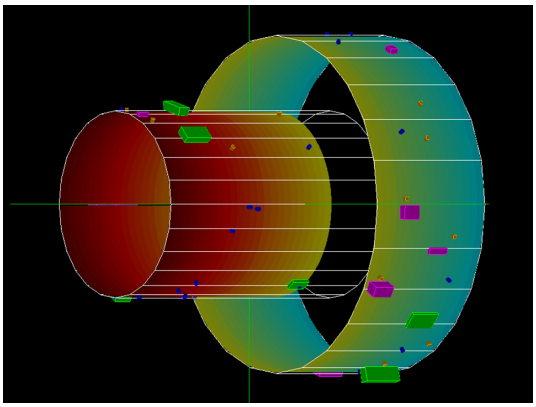


Figure 5.13.2: 3D Engine view

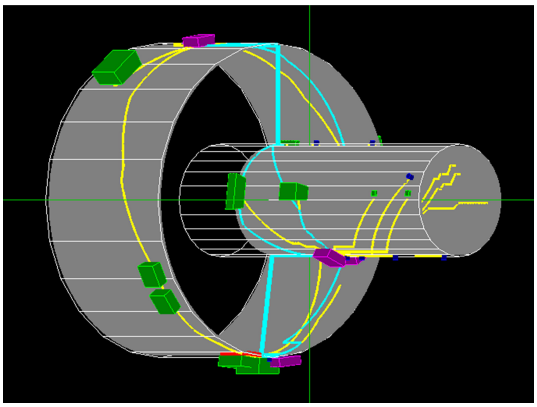


Figure 5.13.3: A 4-node distributed system on engine

The software is written in object-oriented C++ and uses OpenGL®. OpenGL® is a widely used graphical programming Advanced Peripheral Interface (API) allowing custom graphics to be displayed to the screen. The visualisation tool has many intricacies and design challenges of it's own. Coordinate transforms and vector calculations are required to located the vertices and gradients required for drawing and colouring. The software is implemented using a variety of object-oriented constructs.

Examples of the visualisation tool output are shown in figures 5.13.2 and 5.13.3.

5.14 Implementation with Matlab

The GA, architecture construction and architecture evaluation were all implemented in object-oriented MATLAB®. In spite of the slow execution speed, MATLAB® provides easy access to many of the mathematical functions required to implement the algorithms and a suit of plotting functions for easy data analysis. Array manipulation and matrix algebra are simple and despite the lack of functions for set algebra and graph presentation, MATLAB® was the most obvious candidate tool. However, MATLAB® code is slow to execute and when compared to languages such as C++, offers only a limited range of object-oriented constructs. Object-oriented MATLAB® (edition 2008a) and its debugger are finicky and hampered by software bugs which make implementation trying. A full implementation in C++ would have required significantly more programming effort: data visualisation would have been very difficult and the code harder to structure and debug.

The optimisation routine used in this research is eminently suited to parallel implementation - the architecture construction and evaluation functions for individual population members are mutually independent and the algorithm is calculation rather than data intensive. Parallelising the optimisation routine is likely to yield significant time savings and permit an increased number of DCS configurations and improved harness routing algorithms. Opportunities and methods for parallel implementation is discussed in Appendix B.

5.15 Evaluation Functions

This chapter has shown how the metaphysical models of the DCS architectures are constructed to realise a software model of the system. The proceeding chapters detail how these models are evaluated by the GA. The evaluation functions use predicate logic, mathematical functions and meta-heuristic methods to determine the architectural quality, lifecycle performance and business performance of each candidate architecture. At a functional level, the evaluation functions are independent although parameters passed from one to the other require that they be executed in the order architecture, lifecycle, business. Each of the functions aims to derive a number of, ‘‘key performance indicators’’ for each of the DCSs. The scheme is illustrated on the adjacent page (figure 5.15.1) with

the various evaluation parameters and pre-requisite information shown. The figure shows the various input and output parameters for the three evaluation functions.

The evaluation functions produce a large number of output parameters. It is not intended that all these parameters are used by the GA - all the parameters produced are of interest to system architects. The more fundamental parameters such as harness length and Net Present Value (NPV) could be used during optimisation and the remaining metrics for differentiation between architectures on the pareto optimal front.

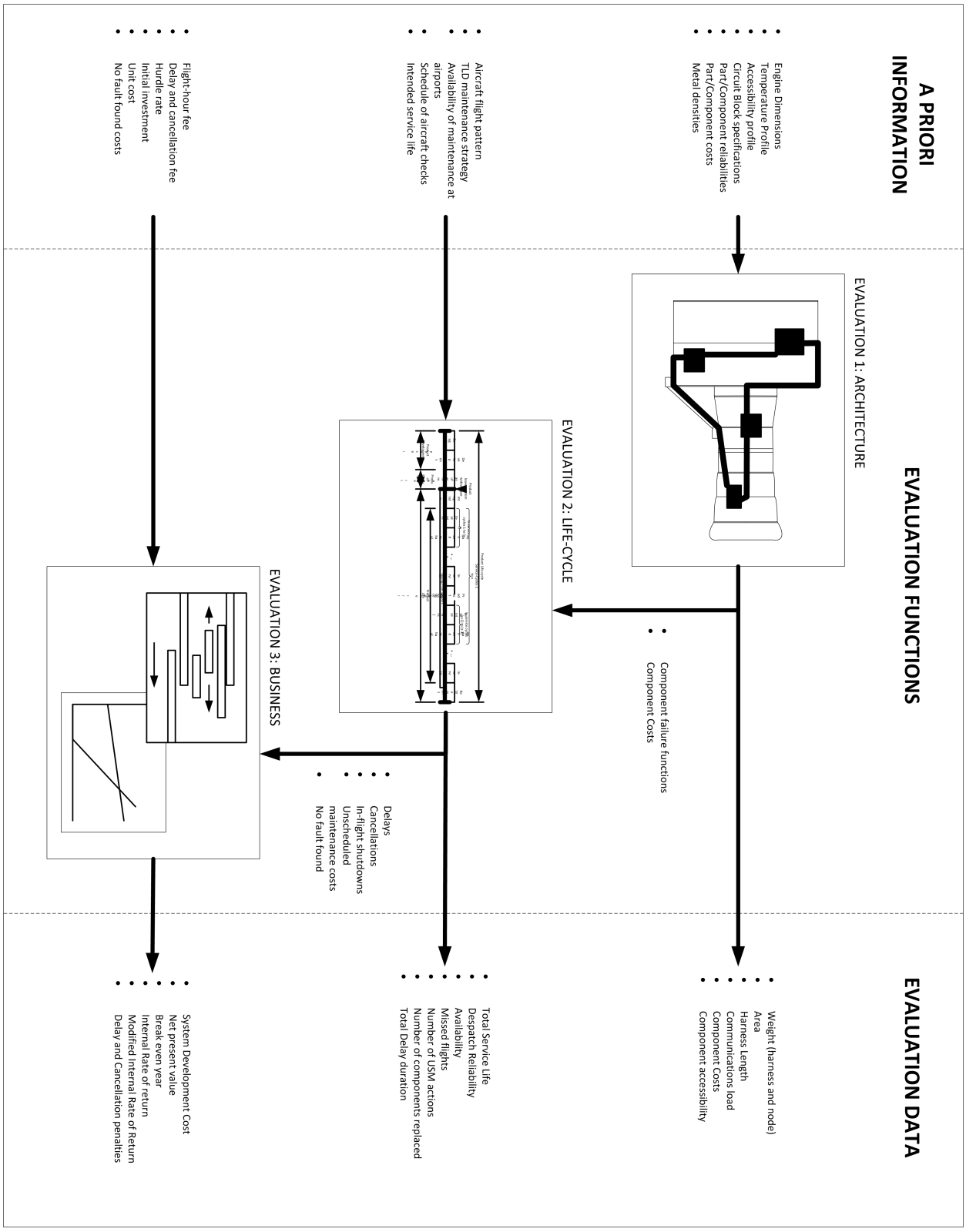


Figure 5.15.1: The chain of evaluation functions showing the pre-requisite information and results parameters used by each of the three evaluation functions. The diagram may be considered as a lower level view of the ''Evaluate Solutions'' block in the synoptic diagram of the overall optimisation scheme (figure 5.2.1)

Chapter 6

Architectural Evaluation

6.1 Introduction

Having built the DCSs, the architectural evaluation considers the physical qualities of the proposed systems. For the most part, the evaluation function is a simple set of summations that combine the weight and size of the nodes and harnesses to find the total system weight and dimensions. The individual node and harness dimensions are calculated during the construction process.

6.2 Calculating the Architectural Measures

In the following section, the variable $W(*)$ signifies a weight, $L(*)$ a length, $D(*)$ a difficulty factor, $V(*)$ a volume, $A(*)$ an area and $T(*)$ an access time. N is the number of nodes and H_s, $H_{d(1|2)}$ and $H_{p(1|2)}$ the number of signal, data and power harnesses respectively.

The system weight is the combined weight of the nodes, signal harness, data harnesses and power harnesses. The signal harnesses are considered to contribute to the total node weight:

$$W(\texttt{nodes}) = \sum_{n=1}^{N} \left[W(\texttt{Node[n]}) + \sum_{h=1}^{H_s} W(\texttt{node[n].Signal harness[h]}) \right] \tag{6.1}$$

$$W(\texttt{data network}) = \sum_{h=1}^{H_{d1}} W(\texttt{data network 1.harness[h]}) + \sum_{h=1}^{H_{d2}} W(\texttt{data network 2.harness[h]}) \tag{6.2}$$

$$W(\texttt{power network}) = \sum_{h=1}^{H_{p1}} W(\texttt{power network 1.harness[h]}) + \sum_{h=1}^{H_{p2}} W(\texttt{power network 2.harness[h]}) \tag{6.3}$$

Therefore, the total system weight is:

$$W = W(\texttt{nodes}) + W(\texttt{data network}) + W(\texttt{power network}) \tag{6.4}$$

Similarly, the total harness length is found by:

$$L(\texttt{signal}) = \sum_{n=1}^{N} \left[\sum_{h=1}^{H_s} L(\texttt{node[n].Signal harness[h]}) \right] \tag{6.5}$$

$$L(\texttt{data network}) = \sum_{h=1}^{H_{d1}} L(\texttt{data network 1.harness[h]}) + \sum_{h=1}^{H_{d2}} L(\texttt{data network 2.harness[h]}) \tag{6.6}$$

$$L(\texttt{power network}) = \sum_{h=1}^{H_{p1}} L(\texttt{power network 1.harness[h]}) + \sum_{h=1}^{H_{p2}} L(\texttt{power network 2.harness[h]}) \tag{6.7}$$

The total harness length L is:

$$L = L(\texttt{signal}) + L(\texttt{data network}) + L(\texttt{power network}) \tag{6.8}$$

The total area of the system is the sum of node footprint areas. The area does not include the area occupied by harnesses:

$$A = \sum_{n=1}^{N} A(\texttt{node[n]}) \tag{6.9}$$

Likewise, the system volume comprises for the node volumes alone.

$$V = \sum_{n=1}^{N} V(\texttt{node[n]}) \tag{6.10}$$

The power consumption of the nodes may be summed in the same way. The accessibility times are used principally by the lifecycle evaluation function to adjust the duration of maintenance actions. The architectural evaluation provides a crude measure of accessibility time by summing the access time for every node:

$$T = \sum_{n=1}^{N} T(\texttt{node[n]}) \tag{6.11}$$

Difficulty Factors

Each node, harness, power and data network has an associated difficulty factor. The difficulty factor is a crude but important, ‘‘catch all’’ metric intended to account for factors which are not in themselves design drivers but important secondary considerations. A prominent example is the number of harnesses crossing between the fancase and core. Within sensible limits, engineers are unlikely to use the number of core-fancase harnesses as a basis for choosing between two systems; control system performance, reliability, power consumption would normally take precedence. However, routing harnesses between the fancase and core complicates harness design and increases the complexity of manufacture and maintenance. If all other factors were equal, the number of core-to-fancase harnesses could be used to distinguish between two designs.

Each network topology has an associated difficulty factor depending on the robustness and redundancy inherent in the network. For example, the ring topology is more robust than the bus topology as communication is possible in the presence of broken links; the bus network carries a higher difficulty score. However, the bus topology requires less wiring and would provide a payback in terms of harness length and weight.

Some qualities have negative difficulty factors meaning that they ease the design problem. Each architecture has two data networks - if both share the same topology,

negative difficulty points are scored. This represents the savings in software complexity, hardware and harnesses design.

The table 6.2.1 below shows how difficulty points are accrued:

Criteria	**Difficulty +/-**
Core-fancase harness crossing	1
Common network topology	-3
Bus Network topology	5
Direct feed network topology	3
Star network topology	2
Ring network topology	-2

Table 6.2.1: Difficulty scores associated with various design features. The difficulty factors for networks apply to both power and data networks.

The difficulty factor for each architecture is calculated as the sum of the difficulty factors for each harness and network.

$$D(\texttt{node}) = \sum_{n=1}^{N} \left[D(\texttt{node[n]}) + \sum_{h=1}^{H_s} D(\texttt{node[n].Signal harness[h]}) \right] \tag{6.12}$$

$$\begin{aligned} D(\texttt{data}) = &\left[D(\texttt{data network 1}) + \sum_{h=1}^{H_{d1}} D(\texttt{data network 1.harness[h]}) \right] \\ &+ \left[D(\texttt{data network 2}) + \sum_{h=1}^{H_{d2}} D(\texttt{data network 2.harness[h]}) \right] \end{aligned} \tag{6.13}$$

$$\begin{aligned} D(\texttt{power}) = &\left[D(\texttt{power network 1}) + \sum_{h=1}^{H_{p1}} D(\texttt{power network 1.harness[h]}) \right] \\ &+ \left[D(\texttt{power network 2}) + \sum_{h=1}^{H_{p2}} D(\texttt{power network 2.harness[h]}) \right] \end{aligned} \tag{6.14}$$

$$D = D(\texttt{node}) + D(\texttt{data}) + D(\texttt{power}) \tag{6.15}$$

It is accepted that the difficulty factor is a crude and unedifying metric. An improved implementation would associate each of the factors in table 6.2.1 with a time or cost that could be accounted for in the business or lifecycle evaluation. For example, the extra

development effort required to realise a system with different network topologies could be lead to an increase in development time. This in turn would be translated into a cost by the business evaluation. The number of core-to-fancase harness crossings may be used as a stand-alone design metric.

An alternative improvement would involve attributing qualitative levels of difficulty based on the magnitude of the score. *eg.* high, medium and low. This would prevent the difficulty score from having an overbearing influence on architectural decision making.

Communications Load

The final architectural parameter is the communications load. This is the number of parameters (expressed as a percentage) that are required to be broadcast on the databus. This arises because tasks performed on different nodes require the same parameters. The relational matrix $\boldsymbol{R_{pn}}$ shows which nodes require which parameters. If the row corresponding to a particular parameter is populated more than once, the parameter is required to be broadcast. For example:

$$\boldsymbol{R_{pn}} = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{ccc} n_1 & n_2 & n_3 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \tag{6.16}$$

Parameters 2 and 4 are required to be broadcast on the databus.

6.2.1 Output

The calculations given above are implemented as functions of a matlab class. The output of the evaluation function is a structure containing all the parameters listed in this chapter. An example output is shown below:

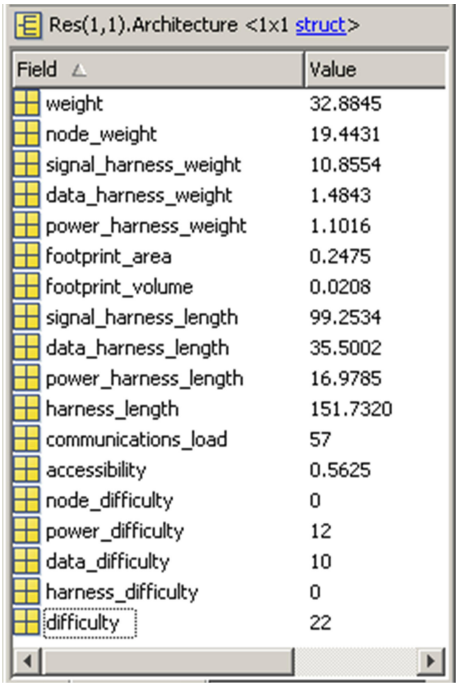


Figure 6.2.1: Typical output from the architecture evaluation function

6.3 Testing the architectural evaluation: A case study

In order to test the architectural evaluation function, a test case was defined. The SPEA2 algorithm was used to optimise distributed control system architectures based on the parameters calculated by the architectural evaluation alone.

The target was a typical, large, civil jet engine with dimensions and functionality similar to that of the Rolls-Royce Trent 1000. The optimiser was configured to consider systems with 2 to 29 nodes. There are twenty-nine distinguishable components on the engine - this allowed the optimiser to allocate one node to every component and provide the highest practical level of distribution. A baseline system with a centralised EEC was used as a benchmark for assessing the quality of the distributed solutions.

Both the centralised baseline and the distributed systems were realised using the construction processes presented in the previous chapter.

6.3.1 The Baseline System

As stated above, the baseline centralised system was based on the Rolls-Royce Trent 1000 Engine. The model engine has similar dimensions and control system functionality. The

locations of sensors and actuators on the engine chassis are approximate. The functional flow diagram of the engine control system is given in Appendix C. This model is recorded in the relational matrices associated with the Control System Architecture Framework (section 5.4.2).

Keepout zones were neglected from the engine model to prevent the computationally expensive process of recalculating the graph routing matrices. Therefore, harnesses may be routed without constraint. The decision space variables were total system weight, total harness length and system difficulty factor.

The engine environment was defined using the functions for temperature, vibration and access times described in the previous chapter. The SAT of the fancase was 85°C and the turbine around 500°C. Vibration varied linearly along the engine from high-levels at the fancase to a low-level at the core.

The baseline system was created using a pre-determined binary string. In essence, a contrived chromosome was created to place a single node at a location consistent with the present day centralised EEC. This system was built using the processes presented in Chapter 5. Accordingly, all tasks and interfaces reside on a single node and thus the two channels are identical. The visualisation of the baseline system in shown in figure 6.3.1:

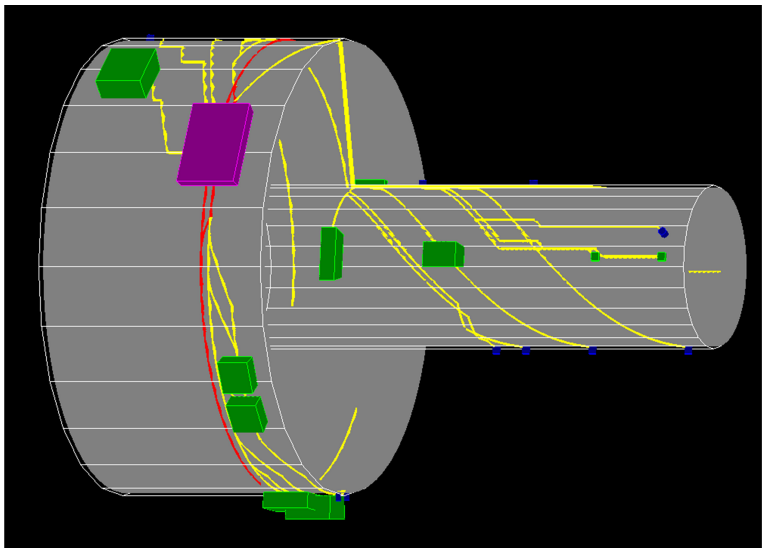


Figure 6.3.1: Baseline centralised architecture

The magenta block is the centralised EEC, the green blocks are the engine subsystems and the blue circles the sensor interfaces. Signal, data and power harnesses are shown in yellow, cyan and red respectively.

Once built, the architectural evaluation function was run on the centralised architecture. The results for each of the metrics are shown in the first column of table 6.3.1.

Attribute	Baseline	Two Node Optimal
Node weight (kg)	17.97	19.25
Signal harness weight (kg)	16.06	11.06
Data harness weight (kg)	0	0.6
Power harness weight (kg)	1.15	0.78
Footprint area (m^2)	0.25	0.25
Footprint volume (m^3)	0.02	0.02
Signal harness length (m)	216.13	106.07
Data harness length (m)	0	5.25
Power harness length (m)	19.44	13.75
Communications load (%)	0	53
Accessibility (hours)	1	0.7
Node difficulty	0	0
Power difficulty	4	-4
Data difficulty	0	-2
Harness difficulty	8	1
Total weight (kg)	35.19	31.7
Total harness wire length (m)	1364	754.6
Total difficulty factor	12	-5

Table 6.3.1: Comparison of architectural evaluation results for the centralised (baseline) system and the optimal two node architecture

Parameters relating to the data harnesses are all zero as no data network is required for a single node. Similarly, the communications load is zero. The node weight and total wire length are comparable to a modern EEC; this provides a primitive verification of the construction and evaluation processes. It should be noted that the centralised system presented is built using the same input data and circuit blocks as for the distributed systems which follow.

6.4 Optimisation for 2 to 29 Nodes

The GA was set to run for 1000 iterations with a crossover probability of 99%, a mutation probability of 3% and 3 dynamic crossover points. The algorithm was executed as shown

in figure 5.3.2 of the previous chapter and optimised the DCSs system weight, combined harness length and difficulty factor. Both the current and archive populations contained 100 members. In practice, only the solutions for 2 to 5 nodes converged in the given number of iterations and the results for 6 or more nodes may be neglected.

Figure 6.4.1 shows the progress of the two node optimisation. The values shown are the averages for all solutions in the pareto optimal front at the end of each iteration.

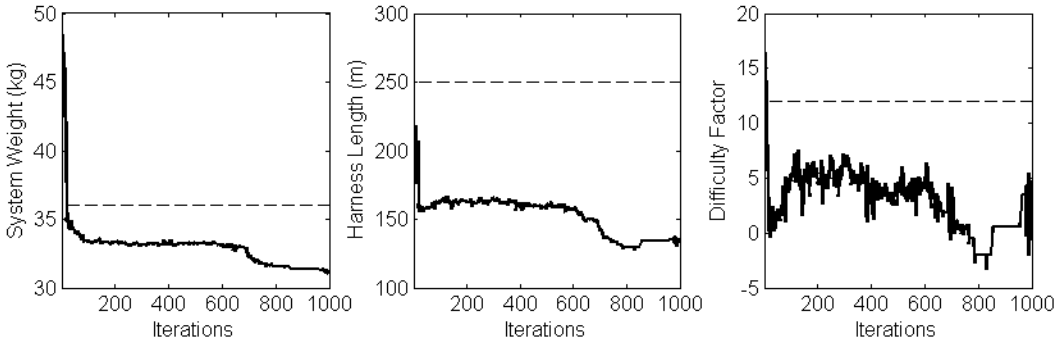


Figure 6.4.1: Average value of system weight, harness length and difficulty factor for architectures in the pareto optimal front of the two node optimisation. The dotted lines represent the values from the baseline system

There is a notable step change at around 700 iterations. This was probably caused by the movement of one of the two nodes form the core to the fancase or the fancase to the core. This would explain the step change in weight and difficulty factor as fewer harnesses are required to cross the fancase-core interface.

After 1000 iterations, the optimiser has produced a two-node DCS that is considerably better than the baseline solution in all three objective functions. A numerical comparison of the baseline and two node system is presented in table 6.3.1. The most notable reduction is the harness length from 250m to 130m. This represents a considerable cost saving and contributes to the 5kg saving in the overall system weight. The two nodes are located on the underside of the engine with one on the fancase and the other on the core. The engine components on the fancase are connected to the fancase node and the core components to the core node. The visualisation of the architecture is shown in figure 6.4.2 below. The solution shown is one of the 7 solutions in the global pareto optimal front. The pareto optimal solutions are found by combining the optimal solutions for every number of nodes and subjecting the resulting set to non-dominated sorting.

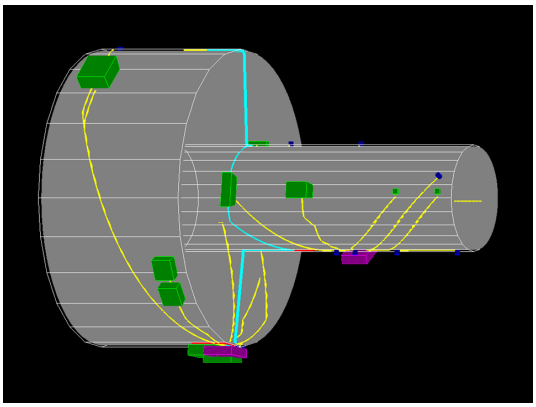


Figure 6.4.2: The two node optimal architecture

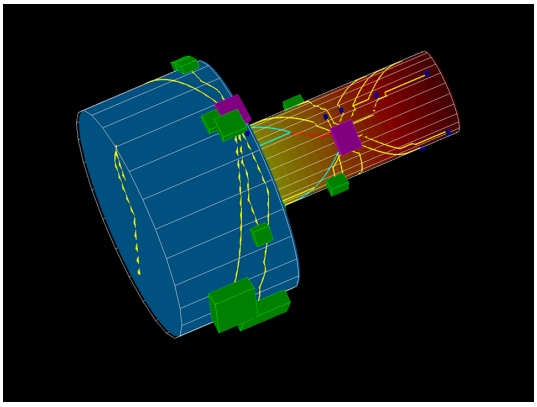


Figure 6.4.3: Two node optimal solution from underside of engine and showing temperature profile

Figure 6.4.3 shows the same system from the underside of the engine and includes the engine temperature profile.

The two node solution presented above is the, ‘‘best rounded’’ of those in the pareto optimal front - in this case, parameters other than those used by the GA have been used to select it. This approach allows the GA to handle the fundamental architectural decisions whilst the designer is free to choose between solutions to meet the needs of a particular application.

Interestingly, many of the three and four node solutions approximate to the architectures of the two node solutions. As shown in figure 6.4.4, a three node solution from the global pareto optimal set has two nodes closely located on the fancase - in effect, one core node connects to components on the port side of the core and the other to components on the starboard side. This trend contradicts the commonly held view that two core nodes would be arranged as a compressor and turbine data concentrator.

Although the results are influenced by the architectural evaluation alone, they serve to challenge and reinforce some of the commonly held notions on DCS design:

- The 2, 3 and 4 node solutions have not allocated a node to either the solenoid banks, oil system, FMU or Turbine Gas Temperature (TGT) thermocouples - these are seen as obvious targets for distributed nodes.
- The system weight has not decreased sufficiently to justify the risk of moving from centralised to distributed architectures.
- The move to a distributed architecture may have a greater impact on dressing time than system weight. This is something not considered by the architectural evaluation function.

- The importance of locating EEC and other FADEC components on the engine core is emphasised.

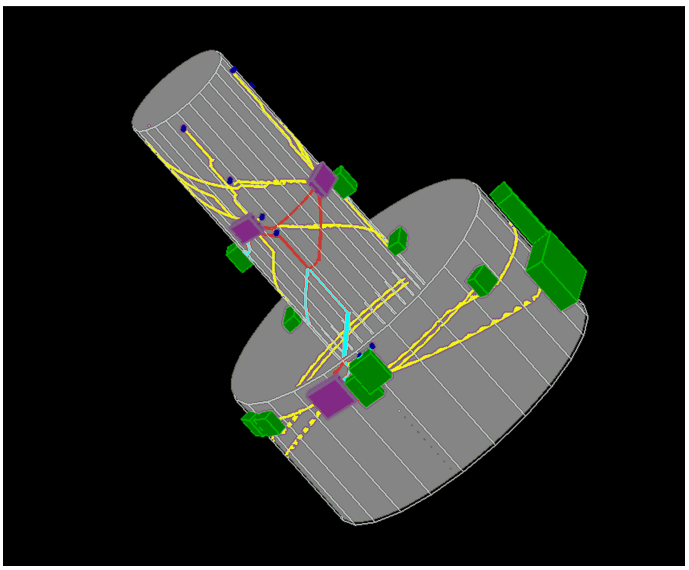


Figure 6.4.4: Three node solution approximating to the two node solution

It is anticipated that the optimal architecture would be very different if the lifecycle and business evaluation functions were able to influence the outcome. In the arrangement presented, the optimiser has no notion of function or redundancy. Therefore, node locations are optimised purely on physical qualities. The inclination of the human designer is to allocate related functionality to nodes rather than consider than asses system architecture from a purely physical perspective. Components such as the TGT thermocouples, oil system and solenoid banks are seen as obvious candidates for distributed nodes yet non of the 3, 4 or 5 node architectures has erred towards either of these configurations.

There is perhaps little sense in considering all configurations from 2 to 29 nodes. It is likely that better distributed solutions will be found where the number of nodes is either low or high. It is suggested that distributed architectures with 7 to 20 nodes need not have been considered.

6.5 Validation

There are no precedent distributed architectures against which to judge designs and the properties of 'good' distributed architecture are principally unknown. Identifying poor architecture is perhaps easier than recognising that which is good. Moreover, Aero Engine

Controls has no immediate intention to build DCSs for commercial use and fabricating prototype systems lies beyond the scope of this project. The outcomes of the research will be evaluated using not only the hard data produced but against conjecture and discrepancies in rational opinion.

The two node solution presented in this chapter represents a, 'plausible' solution to the DCS design problem. The result cannot be shown to be optimal in itself but could be validated against other similar solutions or real life systems if the data were available.

6.6 Conclusion

The method presented in chapter 5 showed how the metaphysical DCSs were constructed from the GA's chromosomes. This chapter has shown how these architectures are evaluated at an architectural level - this process is necessary to verify the architectural evaluation and the performance of the GA on a representative test problem.

Whilst there is no way to formerly verify the architectures are 'optimal', the results obtained appear to provide 'rational' solutions. Despite the simplicity of the results, the disconnection from functional considerations and redundancy engenders architectures which contradict some of the commonly held notions on DCS architecture.

Whilst the constituent parts and procedures are based on elementary methods, this level of architectural optimisation for distributed systems has not been demonstrated previously.

Chapter 7

Lifecycle evaluation

7.1 Introduction

Contractual agreements mandate engine manufacturers to support their product throughout its service life. Engines for large civil aircraft are sold to the airline (in a deal largely independent of the airframer) at a fraction of their advertised price. To recoup the difference, the airline pays a fee to the engine manufacturer for every hour that the engine is operational. The payment is known as the, ''Flight-hour Fee'' or ''Fleet hour rate''. It is expected that the engine manufacturer compensate the airline for any delay & cancelation (D&C)[1] caused by their component and perform or contribute towards the costs of unscheduled maintenance (USM). The arrangement implores the engine manufacturer to supply a reliable and robust product whilst working closely with the airline to ascertain usage, reliability and maintenance patterns. The arrangement is known as, ''Through-life Support'' or more colloquially as, ''Power-by-the-hour''.

Inevitably, the engine manufacturers choose to pass the costs of D&C compensation and USM actions to their subsystem suppliers, if it can be shown that their product was at fault. In return, subsystem suppliers such as Aero Engine Controls recoup their investment as a percentage of the Flight-Hour (FH) fee. The percentage received is known as the, ''programme share'' and is dependent on commercial negotiations normally undertaken prior to product development. Consequentially, subsystem suppliers must consider the lifecycle implications of their design - their major concerns include reliability,

[1] A flight is considered delayed if it is despatched more than twenty minutes following the end of the intended despatch slot. This is not necessarily the same measure used by airlines for evaluation of passenger services.

improving fault isolation and ensuring that component obsolescence does not hinder their long-term ability to support the product.

Traditionally, engine and subsystem suppliers report the lifecycle performance of their products using statistical measures such as MTTF, availability and Mean Time to Unscheduled Removal (MTTUR). These statistics are important for product comparisons, analysis of new and emerging technologies and product promotion. By contrast, the airline's primary measures of lifecycle performance are impact on operational efficiency and public reputation. Airlines are concerned with the number and cost of delays, the fastest and cheapest methods of repair and conserving or improving their public reputation. If a lifecycle evaluation is to hold industrial credence, it must address both statistical measures and the day-to-day impact of product failure.

This chapter details the evaluation function used by the GA to establish the lifecycle performance data of each candidate DCS. Sections 7.3 to 7.6 describe the nature of the engine lifecycle and the various operational actions and behaviours performed whilst in service. A brief literature survey (section 7.2) considers various methods of lifecycle modelling presented in relevant publications before section 7.7 details the techniques and algorithms used in the Monte Carlo simulation. Simulation results and analysis are presented in sections 7.9.1 and 7.9.2.

7.2 Lifecycle Modelling - Literature review

Product lifecycle analysis and modelling has attracted a great deal of interest within many different disciplines. System lifecycle models may be realised using many different methods. References to lifecycle evaluation and modelling may be found in academic texts referring to disciplines as disparate as environmental impact assessment for motor vehicles, to modelling family spending habits in response to changing incomes.

If the lifecycle evaluation is to account for the nuances aircraft operation and maintenance, then Monte Carlo simulation (Kochanski, 2005) is perhaps the only feasible method. Deterministic methods are unable to cope with the logical decisions taken whilst undertaking maintenance actions and determining the implications disruption. Methods based probability theory alone cannot account for many of the considerations most important to the airlines. The suitability of the Monte Carlo approach is reinforced by the

work of Schmitt & Singh (2010) on supply chain modelling, Crk (2002) on component level performance simulation and Marquez & Iung (2007) on using MCS to determine reliability and availability data. Whilst none of these papers is directly relevant the field of aerospace systems, the problems considered have many associating features.

Several authors have considered Monte Carlo methods for the simulation of aircraft fleet performance *eg.* Yang *et al.* (2002). Such simulations do not operate at a component level and do not account for many real-word nuances. There is vast amount of academic literature regarding flight-scheduling and the efficient response of flight-schedules to disruption caused by weather, mechanical failures and air-traffic control problems. Examples include, Lee *et al.* (2007) who use a genetic algorithm to develop a flight schedule which is considered robust to disruption.

Prescott & Andrews present a Monte Carlo based method of modelling time limited despatch for a jet engine EEC. The their work (Prescott & Andrews, 2005, 2006a,b) compares the Monte Carlo method against a time-weighted fault tree approach and a Markov analysis. The EEC architecture considered in their work is comparatively trivial and components fail according to exponential failure functions. Prescott and Andrew's work focuses on the TLD process rather than the performance of the EEC. Their work does not consider the impact of disruption. The author has found no evidence of other component level reliability analyses focusing on the despatchability of aircraft.

Although not implemented in their published form, the MCS algorithm used here incorporates techniques inspired by the work of Yang *et al.* (2009) and Durga Rao *et al.* (2009) on dynamic fault tree synthesis and analysis.

7.3 The Engine Lifecycle

The product support life of a large civil engine may extend to 45 years and consists of three distinct phases: product development, manufacture and in-service support. Typically, large engines take five to eight years to design, develop and deploy; most are expected to remain in service for thirty years. The manufacturing phase runs concurrently with the in-service phase although a small number of manufactured units are required to support test, integration and validation. Generally, the product is manufactured and sold over a ten year period with a peak production output during the fifth or sixth year.

Consequentially, engine and subsystem suppliers consider the product to be a going concern for around forty-five years[1]. The lifecycle of an engine is depicted in figure 7.3.1 below:

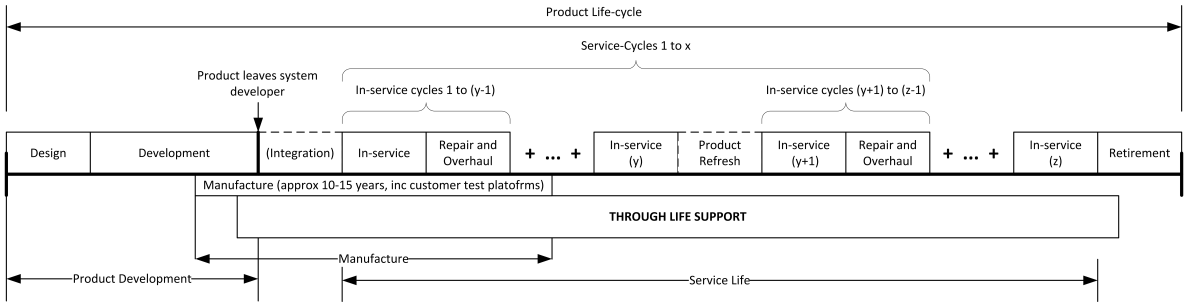


Figure 7.3.1: The engine lifecycle

Once in service, the airframe and engines enter a pattern of service cycles whereby the plane flies for a fixed period before being taken out of service for inspection, repair and overhaul. This cycle usually extends over 4-5 years although the duration is formally measured in flight hours or aircraft cycles. Throughout the in-service portion of the cycle, the aircraft is subject to mandated checks and scheduled maintenance. If component replacement or repair is required outside of these stipulated service times then Unscheduled Maintenance (USM) is performed at cost and inconvenience to the airline. The airlines seek to control, mitigate and recoup the costs of both scheduled and unscheduled maintenance via the through-life service agreement. Support agreements vary from airframe to airframe and airline to airline but most oblige the engine supplier to provide spare parts, perform maintenance and compensate for loss of earnings due to engine-related disruption.

At roughly 15-20 years service (or FH equivalent) most aircraft are subject to a Mid-life Overhaul (MLO). MLO provides an opportunity for more capable subsystems to be installed or for the re-design of recently obsolescent or unreliable parts. Following MLO, the aircraft resumes the pattern of flight and service cycles until retirement.

By the time the airframe retires, it is likely to have been sold to another airline and/or performing a reduced service. Once the through-life service agreement expires, the engine manufacturer ceases responsibility for support of the product and the airframe's owner assumes responsibility for proper disposal.

[1]The service lives of many engines extend beyond thirty years. However, once this period has elapsed and the airframe sold on, the suppliers are rarely accountable for operational support

7.4 Engine Maintenance

7.4.1 Repair and Overhaul (R&O)

As previously described, the airframe and engines are subject to regular checks mandated by various air-worthiness authorities. The checks are referred to as 'A', 'B', 'C' and 'D' checks. Each check is performed at a different interval and removes the airframe from service for a different duration. Each is described presently; the figures given are typical but vary from airframe to airframe and airline to airline:

A-Check - Interval 500FH (approx 25 days). Removal from service: Overnight

The check is usually performed at the airport gate. The operability of the aircraft subsystems is ascertained and the airframe checked for superficial damage. Fault and diagnostic logs may be read from aircraft subsystems. Failed components may be replaced although the limited duration of the check may not permit more demanding repairs.

B-Check - Interval 1500FH (approx 3 months). Removal from service: 1 day

A more comprehensive version of the A-check usually performed in an aircraft hanger. B-checks may be satisfied by a series of A-checks if it can be shown that every element of the B-check has been addressed within a suitable period.

C-Check - Interval 7500FH(approx 15 months). Removal from service: 3 days

C-Checks are usually performed in maintenance hangers. The nature of the check differs with the type of aircraft and flight schedule but generally comprises a more thorough inspection of the airframe, engines and subsystems. Some components are likely to be disassembled to allow for more detailed inspection. The check allows for expendable items to be replaced and provides opportunity for more resource demanding maintenance actions.

D-Check - Interval 22500FH (approx 5 years). Removal from service: 5 weeks

Substantial parts of the airframe and engines are disassembled to allow comprehensive inspection and maintenance. The check is very expensive to perform both in terms of maintenance cost and lost flying hours. A MLO would usually be undertaken in place of a D-Check. Aircraft are usually retired at the time a D-check is due.

In order to simulate the matters described above, the algorithm requires the number of flight-hours flown over the 30 year life-cycle to be calculated. The calculation factors

in the removal from service for maintenance actions. The number of flying hours flown over the lifetime of the aircraft, f_l may be calculated using equation 7.1:

$$f_l = \left\lceil \frac{l \times 365 \times f_{day}}{I_D} \right\rceil I_D \tag{7.1}$$

Where l is the life of the aircraft in years, f_{day} is the number of flying hours per day (16 in this case), 365 is the number of days in a year and I_D is the interval between D-checks in flying hours.

Using this figure, the number of A, B, C and D checks performed over the life of the aircraft may be calculated. The number of D-checks, N_D is:

$$N_D = \frac{f_l}{I_D} \tag{7.2}$$

The number of C-checks performed during the life of the aircraft N_c is then:

$$N_C = \frac{f_l}{I_C} - N_D \tag{7.3}$$

Similarly, the number of B and A checks performed (N_B and N_A) are found by:

$$N_B = \frac{f_l}{I_B} - (N_D + N_C) \tag{7.4}$$

$$N_A = \frac{f_l}{I_A} - (N_D + N_C + N_B) \tag{7.5}$$

The total number of flying hours lost to scheduled maintenance actions, f_m is:

$$f_m = N_A D_A + N_B D_B + N_C D_C + (N_D - 1) D_D + D_{MLO} \tag{7.6}$$

where D_A, D_B, D_C and D_D are the duration of the A, B, C and D-checks in flying hours and D_{MLO} is the duration of an MLO in flying hours. One D-check is replaced by an MLO hence $(N_D - 1)$ in the fourth clause of equation 7.6.

Therefore, the actual number of scheduled flying hours over the lifetime of engine, f_h is:

$$f_h = f_l - f_m \tag{7.7}$$

Scheduled Maintenance

Mechanical, hydraulic and fueldraulic subsystems often require scheduled maintenance to replace worn or expendable parts. It is usual that such subsystems are designed to survive the interval between D-checks without requiring maintenance, although this is not always possible.

Electronic subsystems such as the EEC are usually considered, ''fit-and-forget'' items intended to survive the service duration without requiring attention. In practice, the EEC has mechanical failure modes associated with the temperature and vibration of the hostile engine environment. Common failures include dry solder joints, component and software failure. These failures are considered to be random and are not protected against during Repair & Overhaul (R&O) unless a specific threat or failure mode has been identified. Although software failures are not 'random', it is assumed that the software design process required for certification is robust enough that they may be treated as such. Failed EECs are removed during service phases, replaced with spares and the faulty EEC returned to the manufacturer for repair. Once returned, the EEC is tested and any fault repaired. In some instances, the returned unit is 'patched' to protect against failures seen in similar units or re-programmed with the latest version of software. Once the maintenance action is complete, the unit is returned to the airline. The cost of these repair and protection actions is owned by the EEC developer and absorbed as a cost of doing business.

All these checks provide opportunity for different levels of USM to be undertaken. The complete set of checks, scheduled and unscheduled maintenance are generally referred to as Repair & Overhaul (R&O) actions.

7.4.2 Uncertainty in Maintenance Actions

The EEC is particularly vulnerable to removal in the event of an engine failure. The EEC interfaces to nearly every other engine sub-system, meaning the unit is under suspicion during the investigation of almost any engine fault. The error codes and messages originate from the EEC and maintenance staff often question the integrity of the diagnosis which prompted the message. Sensor and actuator failures are notoriously hard to detect and invariably transient or intermittent. Therefore, the EEC may correctly diagnose a fault which does not re-assert itself if a new EEC is swapped in. Faults in the harnessing and

connectors such as broken pins, shorts or impedance variations often appear similar to internal EEC or sensor faults. Additionally, the electrical connectors make the EEC far easier to remove than hydraulic components which require the draining and bleeding of fuel and oil systems. Sensors are often located deep in the engine chassis and subsequently more difficult to remove. The combination of these factors leads to a disproportionate number of EEC removals over other engine components, many of which are later found to have no fault.

No Fault Found (NFF) rates are a key operational metric for the EEC developer and illustrate the stochastic nature of diagnosis and maintenance actions. Reducing the rate of NFF removal requires a multifaced approach including improving fault diagnosis, better defined lexigraphy in error messages and improved training of maintenance staff. The difficulty of the latter is exacerbated by the high staff turnover, costs and deferred responsibility.

The MCS determines that 40% of node removals will result in a NFF. The current figure is around 50-60%; the lower figure used here reflects the likely improvement in fault isolation associated with distributed systems. The business evaluation function uses these figures to determine the costs of USM to the subsystem provider.

7.5 Time Limited Despatch (TLD)

Commercial Aircraft are permitted to fly in the presence of failures amongst airframe and engine systems under an arrangement known as Time Limited Despatch (TLD). TLD was developed on the premise that modern FADEC systems contain sufficient redundancy and ability to switch between redundant elements, that the risk of hazardous failure in the presence of existing faults is sufficiently small that the airframe may fly for a limited period prior to repair. Depending on the likelihood of further component failures causing a Loss of Thrust Control (LOTC) event, an aircraft acquires one of five despatch statuses: Full Up Despatch (FUD), Short Term Despatch (STD), Long Term Despatch (LTD), Do Not Despatch (DND) or Manufacturer Defined Despatch (MDD). The despatch status determines how long the aircraft may fly for before the fault requires attention. Under MDD the operator consults the manufacturer of the failed part or subsystem to determine the appropriate action. In practice, MDD will result in the fault being recategorised to

one of the other four statuses and need not be handled separately in the lifecycle model (FAA, 2001).

The despatch status is effectively defined using a fault tree where the top event is LOTC. The despatch status of the engine and (and therefore effective on the aircraft) is dependent on the top event probability when calculated in the presence known component failures.

- Full Up Despatch: < 10 LOTC events per 10^6FH. Implies no known fault[1]
- Long Term Despatch: <75 LOTC events per 10^6FH in the presence of a discovered fault
- Short Term Despatch: 75-100 LOTC events per 10^6FH
- Do Not Despatch: >100 LOTC events per 10^6FH

By necessity, the despatch status is dependent on discovered faults only. Accordingly, an engine in FUD configuration need not be fault free and the TLD status is likely to be an under-estimate of the top event probability. In practice, airlines determine a set of, 'despatch configurations' whereby pilots may determine the despatch status of the aircraft based on known component failures. It is assumed that the EEC software is capable of detecting all faults which gives rise to STD and report the error to the cockpit as a maintenance message.

It is assumed that the despatch status of a single engine is effective on the whole aircraft. Therefore, if an engine fault results in STD status, that condition applies to the whole airframe.

7.5.1 Maintenance strategies for TLD

There are two maintenance strategies for TLD: Minimum Equipment List (MEL) and Periodic Inspection/Repair (PIR). MEL assumes that despatchability is dependent on a minimum subset of components which must be in working order for airframe to fly. MEL is generally associated with STD faults and assumes that the time of failure is known.

[1] 10 LOTC events per 10^6FH is the reliability necessary to gain flight-worthiness and refers to the fault-free state. Therefore, if the probability of a LOTC event in the presence of a discovered fault(s) is less than 10 per 10^6FH, the aircraft assumes LTD, not FUD status.

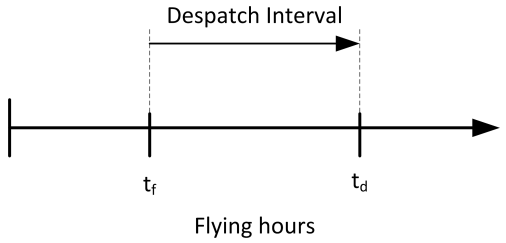


Figure 7.5.1: MEL Maintenance usually applied to STD faults

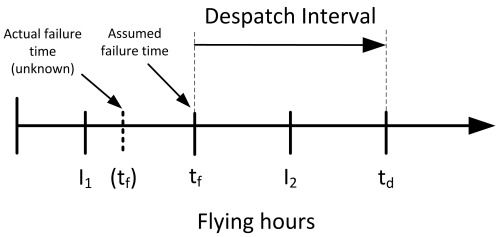


Figure 7.5.2: PIR maintenance usually applied to LTD faults. 'I' denotes an inspection.

DND status requires that corrective maintenance be performed before the airframe is permitted to fly. Typically, STD faults require resolution within a period of 125-150FH and LTD faults within 500FH. All LTD faults are assumed to occur half way between the current and previous inspection. The 500 hour despatch interval is often 250FH from the point of discovery. In FUD, the aircraft is considered free of faults, although undiscovered faults may be present in the system.

7.6 Measures of Lifecycle performance

The principle measure of lifecycle performance is despatch reliability. Despatch reliability is an industry standard measure combing both availability and reliability - it is the ratio of actual despatches to scheduled despatches. Typically, engine manufacturers will maintain records of the despatch reliability with respect to engine failures alone. Subsystem suppliers tend to measure their service life performance by the MTTUR of their component. Important measures of lifecycle performance include:

- Despatch reliability
- Mean Time to Unscheduled Removal (MTTUR) (for subsystems and components)
- Number of flights cancelled
- Number and duration of delayed flights
- Number of missed flights
- Number of components replaced throughout the lifecycle
- Cost of replacement components
- Number of USM actions
- Cost of USM actions

7.7 The Lifecycle Evaluation Function

The lifecycle evaluation function uses a Monte Carlo simulation to model the engine controller's performance throughout its 30 year life. The simulation incorporates flying time, un-scheduled and scheduled maintenance as well as MLO. The simulation ascertains all the parameters listed in section 7.6 for a given architecture. If the lifecycle evaluation were not detached from the architectural evaluation, the parameters would change from architecture to architecture as the GA locates the dual redundant elements of the system to achieve the best architectural configuration. Electronic components in cooler parts of the engine have better reliability and dual redundant elements located on different nodes are likely to increase the system availability.

The simulation mimics the lifecycle of a typical long-haul, large, civil aircraft operated by an established airline[1]. The aircraft flies between three destinations: 'A'. 'B' and 'C' as shown in figure 7.7.1 below:

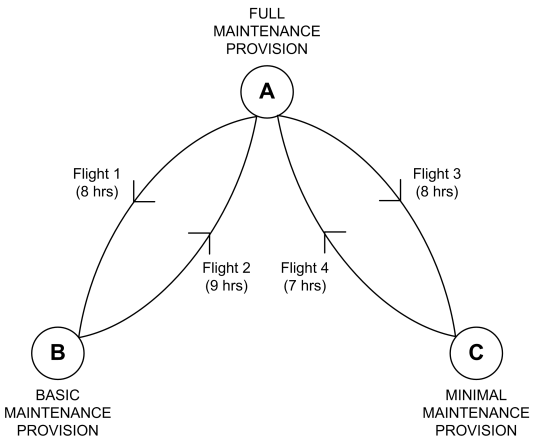


Figure 7.7.1: Flight paths used in the lifecycle simulation. The three destinations offer different levels of maintenance

Four routes are flown over a 48 hour period with breaks between flights and a 7 hour overnight stop at destination A. The 32 hour flying time represents a 66% utilisation of the aircraft which is typical for a long-haul carrier. The timeline for the flight schedule is shown in figure 7.7.2 below:

[1] All values used in this simulation are typical and have not been based on any particular airline's, airframer's or engine's data

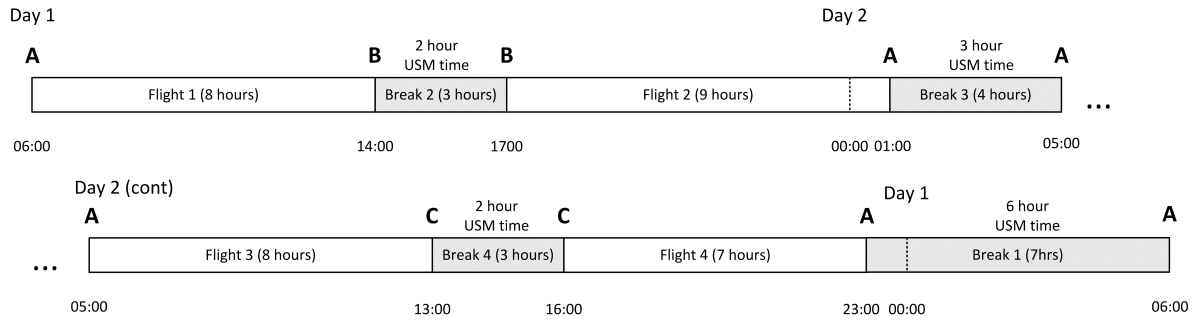


Figure 7.7.2: The 48 hour timeline of flights between destinations A, B and C. Times in between flights are shaded.

Destination A is the airline's home base and thus the preferred location for all maintenance. It is assumed to have the most comprehensive stock of spare parts and is the destination where maintenance is most readily performed. All scheduled maintenance is performed at destination A as well as all USM where possible. Destination B is a secondary base and possess a more limited stock of parts. If a part is unavailable at B, it must be scrambled from another airline or put on a flight from the airline or engine manufacturer's home base. This process is deemed to take 24 hours and has the effect of cancelling one flight and missing three others. Destination C is a tertiary base, carries few spare parts and maintenance actions are difficult to schedule. Like destination B, a part which is not immediately available will have to be scrambled; a process which incurs a 24 hour removal from service.

Each of the inter-flight breaks offers an opportunity for USM, although the time available is considered to be one hour less than the break duration. This time would allow for cleaning, refueling and replenishment of the aircraft. It is important to note that the simulation mimics the performance of the engine's lifecycle with respect to the DCS, not the entirety of the engine, or the service offered by the airline. If a flight is cancelled due to an engine failure, the airline may choose to use a substitute aircraft in it's place. Furthermore, elements of the engine beyond the EEC have their own failure modes not considered by the simulation.

The aircraft is assumed to enter service for 30 years. Based on the 66% flight utilisation, the airframe flies for approximately 5,800 hours per anum excluding time out of service for R&O. The total service life is approximately 175,000FH. It is assumed that MLO is performed at year 15.

As the simulation progresses, it mimics the real-world response to a set of component

failures. These failure events are captured in a software class. The lifecycle simulation works entirely on a set of failure events and does not make direct reference to the metaphysical DCS. The events are derived from the metaphysical model and passed to the evaluation function upon initialisation. Depending on the time and consequences of the failure, the aircraft will undergo scheduled or unscheduled maintenance. Delayed or cancelled flights may result as a consequence of unscheduled maintenance.

7.7.1 Component Failure Models

The despatch status of the DCS is dependent on the likelihood of further hazardous or catastrophic failures during the interval from the time a failure is detected or discovered to a scheduled or hypothetical unscheduled maintenance action. In order to calculate this failure, we require failure distributions for all the components in the system.

The failure processes (cumulative density function) $F(t)$ for each of the components is described by a mixed Weibull distribution in the form:

$$\begin{aligned} F(t, t_c; \lambda, \beta_1, \eta_1, \beta_2, \eta_2) = \lambda \left(1 - \exp\left[-\left(\frac{t - t_c}{\eta_1}\right)^{\beta_1}\right]\right) + \\ (1 - \lambda)\left(1 - \exp\left[-\left(\frac{t - t_c}{\eta_2}\right)^{\beta_2}\right]\right) \quad \text{for } t > 0 \end{aligned} \tag{7.8}$$

Where λ is the mixing parameter, β_1 and β_2 are the shape parameters and η_1 and η_2 are the scale parameters. $F(*)$ is the probability that the component fails in the interval $(0, t]$. t_c is the location parameter and is set to the time at which the component was last replaced. t_c is zero for every component at the start of the simulation. Other parameters may be calculated as follows:

The survival rate:

$$R(t, t_c; *) = 1 - F(t, t_c, *) \tag{7.9}$$

The probability density function:

$$P(t, t_c; *) = \frac{d}{dt} F(t, t_c, *) \tag{7.10}$$

The hazard rate:

$$h(t, t_c; *) = \frac{F(t, t_c, *)}{R(t, t_c, *)} = \left[\frac{F(t, t_c, *)}{1 - F(t, t_c, *)}\right] \tag{7.11}$$

An exponential distribution is too simplistic to provide a plausible representation of real-world electronic component failures. Papers such as Jensen (1989) assert that components such as microprocessors tend to become more reliable throughout their service life given that failures are often associated with flaws in the manufacturing process rather than wear-out. High temperatures and chassis vibration induce wear-out mechanisms such as solder joint failure and package damage. The mixed Weibull distribution allows a single component to have both burn-in and wear-out failure mechanisms and hence a hazard function in the form of a bathtub curve. The flat portion of the curve with a constant failure rate is the, ''useful life period''. Crudely, the scale parameters $\eta_{(1,2)}$ may be thought of as $\mu_{(1,2)} = MTTF_{(1,2)}{}^{-1}$ for the primary and secondary failure mode of the device. The units of MTTF used in the simulation are Flight-Hour (FH). $\beta_{(1,2)}$ determine the kurtosis of the failure function and hence the rate at which failures occur prior to or after the useful life period. Setting β values to zero removes either the burn-in or wear-out mechanism from the function. With both values set to zero, the component will have a constant failure probability for all time. These parameters could be estimated from actual reliability data using techniques outlined in papers from Falls (1970); Cheng & Fu (1982); Cacciari *et al.* (1995). In this simulation, real-world failure trends are represented by arbitrary parameters.

Component failure times are found using the inverse of the mixed Weibull function. The secant method is used to provide a numerical solution in order that component failure times are set according to a random number.

7.7.2 The Failure Event Class

The failure event class contains all the necessary operations and attributes for the Monte Carlo simulation to operate. The class is inherited by the component to which a failure event is associated. This may be a circuit block, node, or harness. A class diagram is given in figure 7.7.4:

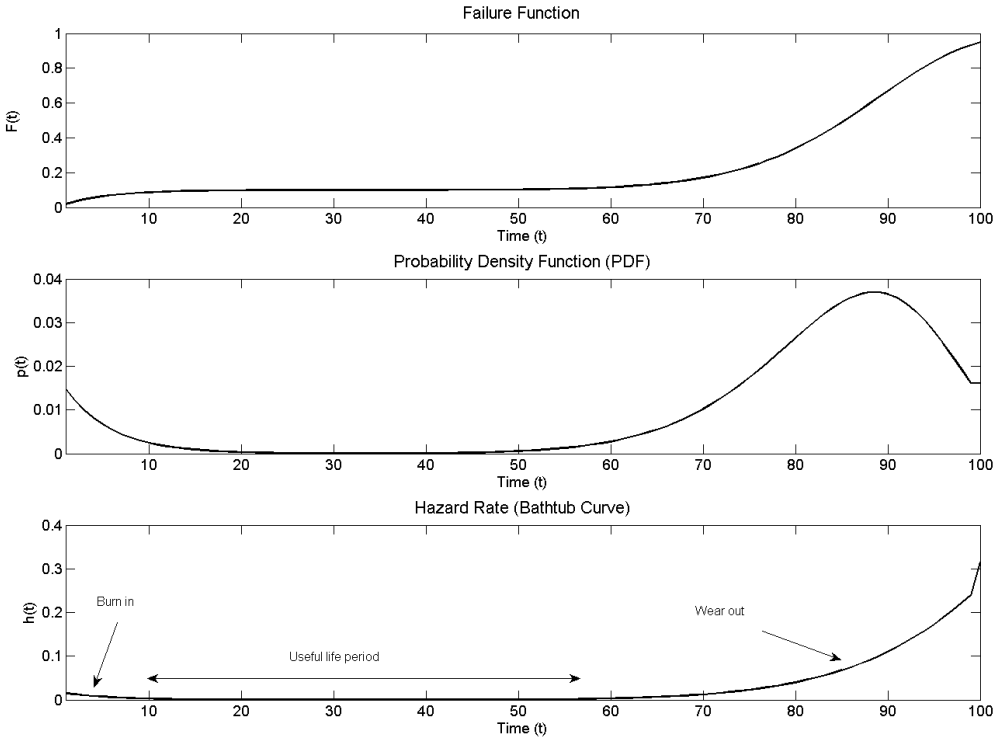


Figure 7.7.3: Mixed Weibull distribution: Failure function, probability density function and hazard rate. The parameters used to generate these example distributions are based on typical failure characteristics of jet engine components. The parameters do not relate to any particular component or failure mode

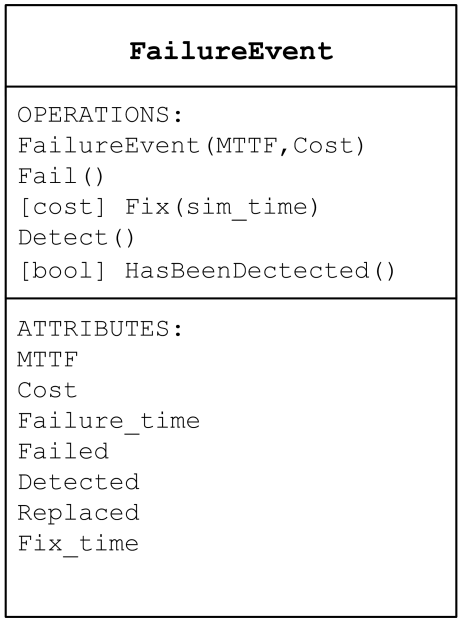


Figure 7.7.4: The FailureEvent class

The constructor function generates a random failure time for the event using the mixed Weibull function as described in (section 7.7.1). The failure time becomes an attribute of the class:

The `Fail()` function is called from the Monte Carlo simulation (MCS) to set the flag `Failed` to true. This indicates that the component has failed but depending on circumstances, that failure may not have been detected. `Detect()` sets the `Detected` flag to true and indicates that not only has the component failed, but that the failure has been detected either by a maintenance technician or automatically by the EEC. This function would usually be called during a USM action where a fault may detected but not necessarily fixed. The `HasBeenDetected` function returns the value of the `Detected` flag. The `Fix` function imitates the replacement of the component and is called during maintenance actions. The function increments the `Replaced` counter, generates a new failure time relative to the current simulation time and sets both the `Discovered` and `Failed` flags to false. The function returns the cost of the replacement part(s). The `Fix_time` attribute stores the number of hours that the component will take to change in the event that it requires replacement.

The MCS requires only the set of `FailureEvent` objects and works at a level of abstraction above the architecture of the underlying distributed control system (DCS). Accordingly, the simulation has no direct reference to the metaphysical model or its

parameters.

7.7.3 TLD Maintenance Strategy

An airline will adjust the level and timing of maintenance to avoid service disruption. These behaviours are reflected in the MCS. The following subsections outline the maintenance strategy employed:

R&O Strategy

All scheduled repair & overhaul (R&O) actions will attempt to return the engine to FUD configuration by fixing every known or discovered fault. This will involve replacing all knowingly faulty parts. However, to reflect reality, the probability of discovering previously undiscovered fault changes with the duration of the check being performed. *i.e.* a fault is much more likely to be discovered at a D-check than an A-check. The algorithm for R&O is given in algorithm 7.7.1:

Algorithm 7.7.1: PERFORMR&O($Components$)

---

for each $component \in Components$
if $component.Failed$ **and** $component.Detected$
  then $cost+ = Component.Fix()$
  else if $component.Failed$ **and** $component.Detected = false$
  then if $(rand < A\_check\_detection\_probability)$
  then $cost+ = component.Fix()$
$sim\_results.cost\_of\_components+ = cost$

The 'Fix()' method of the `FailureEvent` class returns the cost of the replacement component. This is added to the total cost of components in the simulation results. The function presented above is exactly the same for B, C and D checks although the likelihood of fault detection is greater.

USM strategy

USM actions involve the minimum amount of maintenance required to ensure that the aircraft can fly up until its next scheduled check. In doing so, the airline is gambling that no further failures will occur which necessitate USM prior to scheduled R&O. In the past, airlines would fix all known faults and the engine would leave USM in FUD configuration. However, mindful of the costs of delays and acknowledging improvements in component reliability, airlines increasingly choose the minimum fix approach; this is reflected in the simulation.

In order to determine the minimum maintenance actions required at USM, a brute-force algorithm is used to assess every permutation of fixes. Their cost and their potential for seeing the aircraft through to the next scheduled R&O action is assessed and compared. Whilst this could prove computationally expensive, it is unlikely that the number of concurrent failures will be high and hence the number of permutations should remain low (*i.e.* 2-3).

7.7.4 Algorithm

Despite the appearance of the flight pattern timeline, the MCS works entirely in flight hours; this is consistent with industrial measures of lifecycle performance. A notion neglected by the simulation is that of aircraft cycles. A cycle is considered to be one take off and landing, *i.e.* a single full stress of the airframe and engines. This measure is more important to short-haul operators given the increased burden imposed by many take off and landing events. The delays and cancellations which occur in natural time are derived and only an ostensible part simulation's progress.

The algorithm starts with a vector of components each with a predetermined failure characteristic, replacement cost and fix time. The algorithm is most easily explained by the flow diagram given in figure 7.7.5 below although a brief description is given presently.

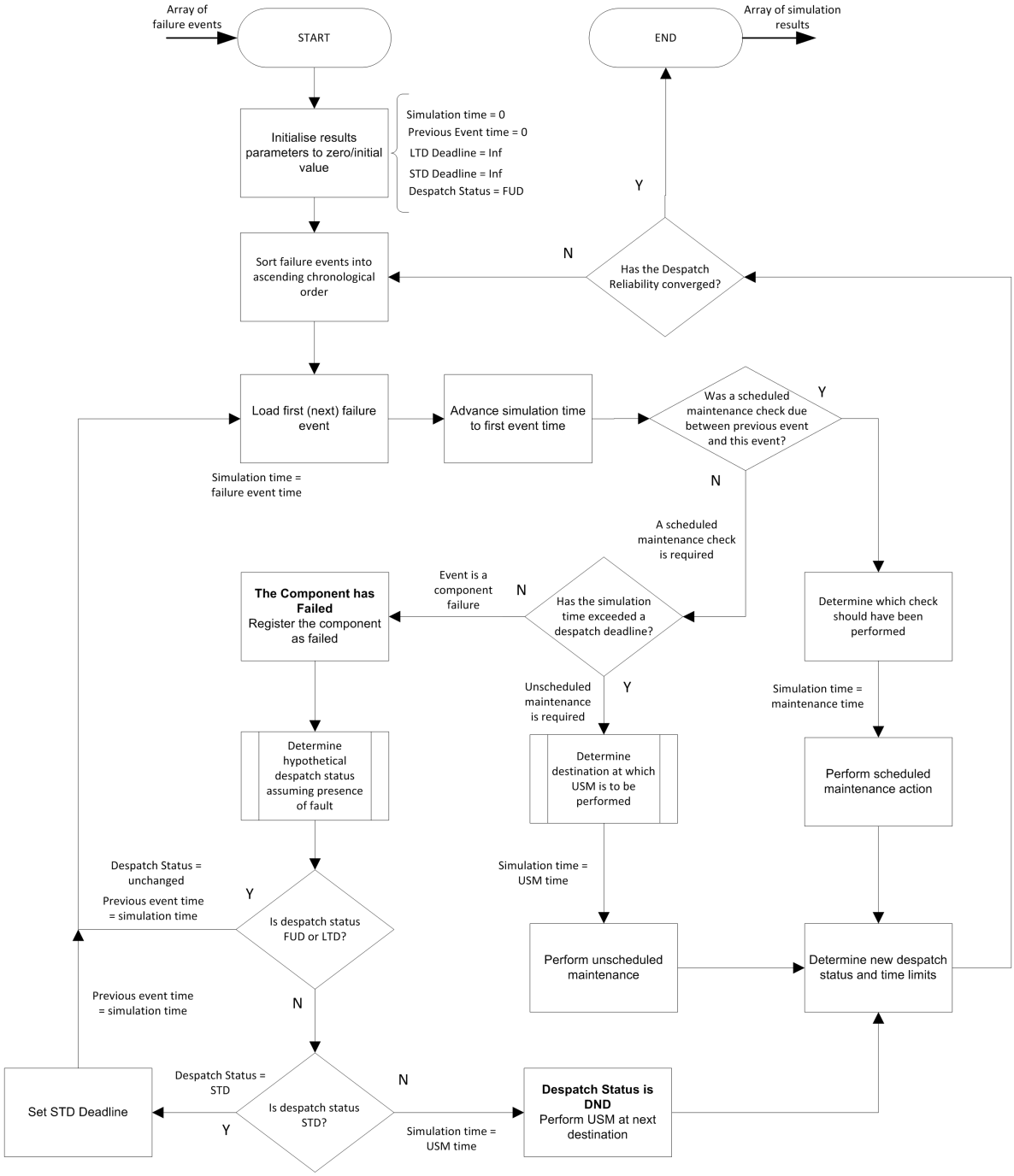


Figure 7.7.5: Top level flow diagram showing the algorithm of the Monte Carlo lifecycle simulation.

Having sorted the events into chronological order, the simulation time is set to the first event failure time and the previous event time to zero. If a scheduled maintenance action was required in the interval between the current failure event and the previous event, the maintenance action is performed. This is achieved by first identifying which of the checks should have been performed and the executing a function to perform the

required action. If no scheduled maintenance is due, the algorithm checks to see that a despatch deadline has not been passed; if it has, the simulation time is put back to the time of the deadline and the unscheduled maintenance performed. If a despatch deadline has not been passed, then the simulation time becomes equal to the component failure time (in FH), the component has its status set to failed and hypothetical despatch status calculated. The hypothetical status allows for components to fail without detection - ie those faults that lead to LTD and are discovered during scheduled R&O actions and those which lead to STD status are discovered immediately.

If the hypothetical despatch status is FUD or LTD (theoretically it should never be FUD in the presence of failure), then the failure event is removed from the array of live failure events and the next failure event loaded into the algorithm. If the hypothetical status is STD then the actual despatch status is set to STD and the STD deadline set appropriately with respect to the current simulation time. If the hypothetical despatch status is DND, maintenance is performed at the next destination.

Delays and Cancellations (D&Cs)

The availability of parts, ability to schedule work and the time taken to perform USM actions determines whether a flight is delayed or cancelled. Flights are considered delayed if the time taken to perform the necessary maintenance would permits flight to depart without compromising the departure of those which follow. A flight is cancelled, if the time taken to complete USM would impinge on future flights. This measure does not necessarily reflect real operational practice.

For example, (w.r.t figure 7.7.2) if an aircraft were to arrive at B with a DND fault and the necessary maintenance takes 4 hours, then the flight from B back to A could still return to A within one hour of the flight due to depart from A→C and hence the flight B→A is delayed. If the fix time were 7 hours, the A→C flight would be affected and so the flight between B and A is cancelled. The same would apply if a part were unavailable or the work could not be scheduled. Missed flights are those which the plane does not fly because it is waiting to synchronise with and rejoin the flight schedule. For the previous example, if the flight from B→A were cancelled, then the plane cannot rejoin the schedule until the flight pattern returns to B and the three flights subsequent to the cancellation (A→C,C→A,A→B) are missed. The distinction between cancelled and missed flights is

important for calculating despatch reliability *(see section 7.7.6)*.

7.7.5 Determining the Despatch Status

During the simulation, a two-tier fault tree is used to determine the despatch status of the aircraft at any given time. The top event is the occurrence of a hazardous or major failure of the engine owing to the malfunction of the distributed control system. Although not the formal definition, the top event is referred to as a Loss of Thrust Control (LOTC) event. The probability of this event occurring determines the despatch status and hence the nature of the ensuing maintenance action within the simulation.

The 'top' tier of the fault tree relates to the functional part of the DCS and the 'lower' tier, to the components which comprise the distributed system under evaluation. The failure events in the top tier relate entirely to control system tasks and are independent of the underlying control system. Each of the top tier events is an intermediate event when the two trees are combined. . Events in the top tier include the failure of the control system to provide the correct fuel flow and the failure of the over-threat tasks to recognise the exceedances and shutdown the engine. Obviously, these tasks depend on the system hardware to measure and condition engine parameters. The failure of this hardware is considered in the lower portion of the fault tree.

The bottom tier of the fault tree relates to the components which allow the tasks of the top tier to be performed. Based on the logical model given in (section 5.4.8), we derive a general fault tree for the failure of any given task. For every event in the top tier of the fault tree, there exists an instance of the general form showing how the task failure is dependent on the system hardware. The general form of a task failure is given in figure 7.7.6 below:

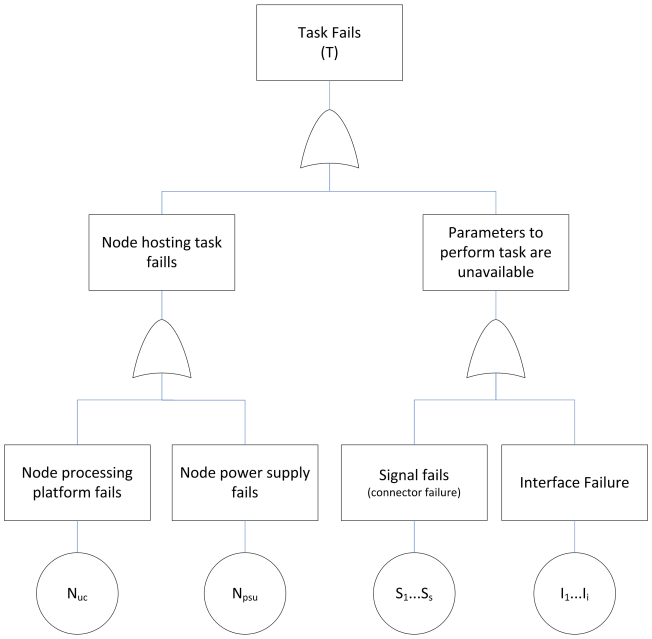


Figure 7.7.6: General model of a fault tree for individual task failure

A textual interpretation of the task failure tree states that, ‘‘A task will fail if the node hosting the task fails, or any of the parameters necessary to complete the task become unavailable’’. Given that the provision of parameters is dependent on the electrical signals from sensors and actuators and the interface circuitry to which they connect, it may also me stated that, ‘‘A parameter becomes unavailable if either the interface providing that parameter or any of the signals connected to the respective interface fail’’. A node failure arises from failure of either the processing platform or power supply. These failure modes many be due to wear-out induced by vibration and temperature or a random failure caused by a design error or software bug. Interface failure modes are the same as for nodes whilst a signal failure is caused by failures of a harness pin or connector. A boolean expression for the failure of a task T performed by node N which is dependent on a single parameter derived form a single interface I, with connection to a single signal S is given as:

$$T = N + (I + S) \tag{7.12}$$

As stated above, node failure is dependent on failure of either the power supply N_{psu} or processing platform N_{μ}. Furthermore, an interface need not be located on the same node as the task is performed and so the failure of an interface is dependent on the interface

itself I, and the node on which it is hosted I_n. Therefore the expression may be rewritten as:

$$T = (N_{psu} + N_{\mu}) + (I + I_{psu} + I_{\mu} + S) \tag{7.13}$$

Where the failure of the node hosting the interface is $I_n = I_{psu} + I_{\mu}$. In practice a task will require many parameters, each with their own interfaces and signals. Equation 7.13 may be rewritten in general form as:

$$T = (N_{psu} + N_{\mu}) + \biguplus_{i=1}^{|I|} \left[(I^i + I^i_{psu} + I^i_{\mu}) \biguplus_{s=1}^{|S|} S^s \right] \tag{7.14}$$

Where $\uplus$ denotes the cumulative logical 'OR' of all interfaces and signals associated with the task. $|I|$ and $|S|$ are the number of interfaces associated with the task and number of signals associated with the interface respectively.

The expression given in equation 7.14 is independent of components and consists of independent failure modes - there are no logical 'ANDs' relating the failure modes. In essence, the probability of a task failure is simply the OR-ing together of all the component failure probabilities for that task, whether they be nodes, interfaces or signals.

In order calculate the probability of failure for each task, the relationships between tasks and nodes, tasks and interfaces and tasks and signals and their multiplicities are required. This information is provided in the matrices $\mathbf{R_{QN}}$, $\mathbf{R_{QI}}$ and $\mathbf{R_{QS}}$ (section 5.5). (The terms I_{psu} and I_{μ} may be neglected from equation 7.14 as the matrix $\mathbf{R_{QN}} = \mathbf{R_{NI}}\mathbf{R_{IP}}\mathbf{R_{PQ}}$ includes the relationship between interfaces and nodes in its calculation).

Component Failure

The relationships between signals, circuit blocks, nodes and the tasks they support are recorded in the matrix $\boldsymbol{R_{FUD}}$ where:

$$\boldsymbol{R_{FUD}} \triangleq \begin{bmatrix} \boldsymbol{R_{SQ}} \\ \boldsymbol{R_{IQ}} \\ \boldsymbol{R^T_{QN}} \end{bmatrix} \tag{7.15}$$

$\boldsymbol{R_{FUD}}$ represents the relationships between components and tasks when every component is working and the engine is in FUD configuration. For the system described in the previous section: $\boldsymbol{R_{FUD}}$

$$\boldsymbol{R_{FUD}} = \begin{array}{c} \\ \texttt{Signal 1} \\ \texttt{Signal 2} \\ \texttt{Signal 3} \\ \texttt{Interface 1} \\ \texttt{Interface 2} \\ \texttt{Node 1} \\ \texttt{Node 2} \end{array} \begin{array}{c} \begin{array}{ccc} \texttt{Task 1} & \texttt{Task 2} & \texttt{Task 3} \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \end{array} \tag{7.16}$$

As described above, the failure probability of a task is the probability that any of the hardware components allowing that task to be performed fails. This is representative of industrial practice where the possibility of software and functional errors is considered to be zero given that the design process is assumed to be robust enough to eliminate them. This of course, is an unrealised ideal.

It is assumed that components (the union set of nodes, signals and circuit blocks) will each fail according to the mixed Weibull distribution presented given in (section 7.7.1). Therefore, the probability of component failure in the interval $(0, t]$ is calculated using the function $P_c(*)$ equation 7.17:

$$\begin{aligned} P_c(t, \boldsymbol{t_c}; \boldsymbol{\lambda}, \boldsymbol{\beta_1}, \boldsymbol{\eta_1}, \boldsymbol{\beta_2}, \boldsymbol{\eta_2}) = \boldsymbol{\lambda} \left(1 - \exp\left[-\left(\frac{t - \boldsymbol{t_c}}{\boldsymbol{\eta_1}}\right)^{\boldsymbol{\beta_1}}\right]\right) + \\ (1 - \boldsymbol{\lambda}) \left(1 - \exp\left[-\left(\frac{t - \boldsymbol{t_c}}{\boldsymbol{\eta_2}}\right)^{\boldsymbol{\beta_2}}\right]\right) \quad \text{for } t > 0 \end{aligned} \tag{7.17}$$

This is the vector form of the mixed Weibull function described in section 7.7.1. $\boldsymbol{\lambda}$, $\boldsymbol{\beta_{(1,2)}}$, $\boldsymbol{\eta_{(1,2)}}$ and $\boldsymbol{t_c}$ are vectors containing the mixing, shape, scale and location parameters for each of the components. t is the simulation time. The location parameter $\boldsymbol{t_c}$ is non-zero where a component has been replaced prior to the current simulation time:

$$\boldsymbol{\lambda}^T \triangleq \begin{bmatrix} \lambda_{S_1} & \dots & \lambda_{S_{|S|}}, & \lambda_{C_1} & \dots & \lambda_{C_{|C|}}, & \lambda_{N_1} & \dots & \lambda_{|N|} \end{bmatrix} \tag{7.18}$$

$$\boldsymbol{\beta}_1^T \triangleq \left[\beta_{1S_1} \quad \dots \quad \beta_{1S_{|S|}}, \quad \beta_{1C_1} \quad \dots \quad \beta_{1C_{|C|}}, \quad \beta_{1N_1} \quad \dots \quad \beta_{1|N|}\right] \tag{7.19}$$

$$\boldsymbol{\beta}_2^T \triangleq \left[\beta_{2S_1} \quad \dots \quad \beta_{2S_{|S|}}, \quad \beta_{2C_1} \quad \dots \quad \beta_{2C_{|C|}}, \quad \beta_{2N_1} \quad \dots \quad \beta_{2|N|}\right] \tag{7.20}$$

$$\boldsymbol{\eta_1}^T \triangleq \left[\eta_{1S_1} \quad \dots \quad \eta_{1S_{|S|}}, \quad \eta_{1C_1} \quad \dots \quad \eta_{1C_{|C|}}, \quad \eta_{1N_1} \quad \dots \quad \eta_{1|N|}\right] \tag{7.21}$$

$$\boldsymbol{\eta}_2^T \triangleq \left[\eta_{2S_1} \quad \dots \quad \eta_{2S_{|S|}}, \quad \eta_{2C_1} \quad \dots \quad \eta_{2C_{|C|}}, \quad \eta_{2N_1} \quad \dots \quad \eta_{2|N|}\right] \tag{7.22}$$

In a full implementation, the mixing and shape parameters would be determined by the component's specification and the scale parameters by the components temperature on the engine *(see section 5.10)*. The simulation requires the probability of component failure over the duration of the next despatch interval dt. This is found using the function P_{ci}:

$$P_{ci}(t, dt, \boldsymbol{t_c}; *) = P_c(t + dt, \boldsymbol{t_c}; *) - P_c(t, \boldsymbol{t_c}; *) \tag{7.23}$$

Where t is the simulation time. The function P_{ci} returns a vector of length $|C|$ where the column values are the probability of the c-th component failing in the interval $(t, t+dt]$. The matrix $\boldsymbol{P_T}$ is defined as a $|C| \times |Q|$ matrix that replicates this vector for all $|Q|$ tasks multiplied by $\boldsymbol{R_{FUD}}$

$$\boldsymbol{P_T} \triangleq \left[P_{ci}(t, dt, \boldsymbol{t_c}; *)_1, \quad \dots, \quad P_{ci}(t, dt, \boldsymbol{t_c}; *)_{|Q|}\right] \bullet \boldsymbol{R_{FUD}} \tag{7.24}$$

Where $\bullet$ denotes element wise matrix multiplication. For the system presented in equation 7.16, $\boldsymbol{P_T}$ may be:

$$\boldsymbol{P_T} = \begin{bmatrix} 0.262 & 0.262 & 0 \\ 0.01 & 0 & 0.01 \\ 0.0245 & 0 & 0.0245 \\ 0.176 & 0 & 0.176 \\ 0.845 & 0.845 & 0 \\ 0 & 0 & 0.143 \\ 0.167 & 0 & 0.167 \end{bmatrix} \tag{7.25}$$

The probability of each task failing in the interval $(t, t+dt]$ is found by applying the inclusion-exclusion principle (equation 7.26) to every column of $\boldsymbol{P_T}$

$$\begin{aligned}\left|\bigcup_{i=1}^{n} A_i\right| = &\sum_{i=1}^{n} |A_i| - \sum_{i,j\,:\,1\le i<j\le n} |A_i \cap A_j| \\ &+ \sum_{i,j,k\,:\,1\le i<j<k\le n} |A_i \cap A_j \cap A_k| - \;\cdots\; + (-1)^{n-1} |A_1 \cap \cdots \cap A_n|\end{aligned} \tag{7.26}$$

$\boldsymbol{P_T}$ represents the system in the initial state when no faults are present. If a component c fails, the matrix $\boldsymbol{P_T}$ is calculated using the function P_{ci}. The c-th row of $\boldsymbol{P_T}$ is set equal to $\boldsymbol{R_{FUD(c,*)}}$. This sets the failure probability of the corresponding component to '1' for every task dependent on the component. The inclusion-exclusion principle is applied to $\boldsymbol{P_T}$ to determine the task failure probability and those probabilities are used as the event probabilities for the top-level fault tree. The top event probability determines the despatch status.

7.7.6 The Results Array

Each pass of the MCS simulates one engine lifecycle and returns an array containing various performance measures. The array is named `lifecycle_results` and pertains to one run of the monte-carlo simulation. Testing has shown that tens or hundreds of iterations may pass before the despatch reliability converges to a sufficient accuracy. The list below details various elements of the results array. Default values are given in parentheses and array sizes in square brackets:

```
lifecycle_results.
  despatch_reliability*
  MTTUR*
  total_service_life*
  availability*
  average_delay*
  cancellations [1:year](0)
  missed_flights [1:year](0)
  USM_actions [1:year](0)
  USM_cost [1:year](0)
  USM_duration [1:year](0)
  components_replaced [1:year](0)
```

```
delays [1:year](0)
delay_duration [1:year](0)
cost_of_components [1:year](0)
standing_time [1:year](0)
```

Where, `year` is the number of years that each DCS remains in service.

Most of the parameters (all those with default values of zero) are simply incremented as the simulation progresses. For example, if a flight is delayed the value of delays is `lifecycle_results.delays += 1`. These results are divided by the number of iterations to give an average value over all iterations. The items marked with an asterisk are calculated either prior to or following convergence of the simulation. The remaining parameters are discussed presently.

Simulation Results

At the end of the simulation, a single instance of the `lifecycle_results` array is returned to the genetic algorithm (GA). Aside from those parameters marked with an asterisk in the list above, the final results array contains averages of all other results parameters from every pass of the simulation. For example:

$$\texttt{final\_lifecycle\_results.cancellations} = \left\lceil \frac{1}{p} \sum_{n=1}^{p} \texttt{lifecycle\_results[n].cancellations} \right\rceil \tag{7.27}$$

Where p is the number of passes taken for the simulation to converge.

Despatch Reliability

Despatch reliability is simply the percentage of scheduled flights that depart on time. In this simulation, the measure equates to the percentage of flights which are not delayed or cancelled from those which the aircraft is scheduled to fly. The despatch reliability calculation excludes missed flights, as from the time of initial cancellation, the airframe is not expected to undertake these flights. Given the 32FH, 4 despatch flight cycle shown in figure 7.7.2, the total number of scheduled despatches f_d may be calculated by $4f_h/32$ and hence the despatch reliability by:

$$\texttt{despatch\_reliability} = \left[\frac{f_d - \texttt{missed\_flights} - \texttt{cancellations} - \texttt{delays}}{f_d - \texttt{missed\_flights}}\right]\% \tag{7.28}$$

Mean Time to Unscheduled Removal

MTTUR is the average time between replacement of any component comprising the distributed EEC, be it a node or harness. It is assumed that circuit blocks are structurally incorporated into the node and cannot be replaced in isolation. If a circuit block fails, the whole node is replaced.

Therefore, `MTTUR` (w.r.t flying hours) is calculated by equation equation 7.29:

$$MTTUR(system) = \left[\frac{f_h}{\texttt{USM\_actions}}\right] \qquad [hours] \tag{7.29}$$

Given that the EEC is intended as a, ''fit and forget'' subsystem, all necessary maintenance is considered unscheduled regardless of whether flights are delayed or cancelled. The MTTUR for a single component (n) may be calculated by:

$$MTTUR(component[n]) = \left[\frac{f_h}{\texttt{FailureEvent[n].Replaced}}\right] \qquad [hours] \tag{7.30}$$

Comparison of component removal rates indicates which technologies need to improve to allow the system to become viable.

Availability

Availability is a time-based rather than despatch-based measure. It is the ratio (expressed as a percentage) of the total time that the engine actually spends in service to the time it is intended to spend in service. In this analysis, this is expressed in natural time rather than flying hours. Availability is the percentage time that the airline may use the airframe at will and therefore, the duration of all USM actions, missed and cancelled flights are set against the availability. The `standing_time` parameter in the results array maintains a record of the total time that the aircraft is stood due to cancelled or missed flights. Therefore,the availability of the engine may be calculated using equation 7.31:

$$\texttt{availability} = 1 - \left[\frac{\texttt{standing\_time} + \texttt{USM\_duration}}{total\_life\_in\_natural\_hours}\right]\% \tag{7.31}$$

Where `USM_duration` is the total number of hours that the engine has spent undergoing USM.

Average Delay

The average delay is simply the total delay duration divided by the number of delays:

$$\texttt{average\_delay} = \frac{\texttt{dealay\_duration}}{\texttt{delays}} \tag{7.32}$$

These parameters are either used by the GA or the subsequent business evaluation function.

7.7.7 Implementation

In order to minimise the execution time of the MATLAB® implementation, the Monte Carlo simulation is structured in a unconventional way. A conceptually simple implementation would use a linked-list of failure events into which new events could be inserted. These events would include the component failures, despatch deadlines and maintenance actions. However, this approach requires the computational burden of searching, sorting and inserting into the list. To reduce this burden, a logic-based implementation was designed. Rather than insert maintenance actions and TLD deadlines into the list of events, they are 'inserted' into the algorithm by logical decisions which determine whether R&O action should have taken place between the time of the previous and current failures. This approach requires failure events to be sorted following maintenance actions rather than after every event or inspection. This gain is realised at the expense of algorithmic complexity, the consequential likelyhood of programming 'bugs' and reduced scope for modification.

7.8 Verification and Validation

The stochastic nature of the Monte Carlo Algorithm makes it very difficult to verify. Verification is difficult because:

- The algorithm is driven by random numbers and producing repeatable results is impossible.
- The response to failure events is highly dependant on previous events which in turn, were a consequence of randomness *i.e.* the response to the second failure event is heavily dependent on the response to and outcomes of the first.
- There are perhaps millions of basis paths
- Contriving inputs to produce definite, repeatable outcomes would mean re-writing the algorithm to the extent that is would not be representative of the of the algorithm under test.
- The algorithm is application specific enough that there are no canonical test problems.
- Due to the unavailability of reliability data for real engine control systems, the algorithm is limited, (by implementation, not design) to using small sets of data that allude to, rather than accurately reproduce reality. Real component failure data would permit validation against real engine performance.

The easiest way to verify the algorithm is through the use of the debugging tools provided with MATLAB®. By stepping through line by line, the algorithm's response to various events, inputs and random numbers can be checked. However, this manual verification technique is inherently non-formal and cannot easily be recorded in a thesis such as this.

7.9 Simulation Test and Results

As stated in the thesis introduction, computational constraints mean that the lifecycle and business evaluation functions have been executed in isolation from the architectural optimisation. In order to verify the algorithm and demonstrate the effect of reliability on lifecycle performance, the algorithms has been exercised on a small but representative set of test data. The data comprises a system of fourteen components which are required for the proper execution of six (three dual redundant) control system tasks. The data used does not approximate to a full scale engine control system but is built on representative structures. Amongst the components are six signals, four interfaces and four nodes. The matrix $\boldsymbol{R_{FUD}}$ for the system is shown in figure 7.9.1.

The simulation is executed for three systems with different component radiabilities.

The component failure functions used for the most reliable case are presented in figure 7.9.3. Data for two less reliable cases was derived by dividing the scale parameters of the most reliable case by three and five respectively - accordingly the test cases are refereed to as 'R', 'R/3' and 'R/5'. In crude terms, the division of the scale factors reduces the MTTF and consequentially increases the likelihood of failure. The outputs from the three simulations is used by the business evaluation (chapter 8). $\boldsymbol{R_{FUD}}$ for the test case is shown below:

$$\mathbf{R_{FUD}} = \begin{array}{c} \begin{matrix} \text{T1A} & \text{T1B} & \text{T2A} & \text{T2B} & \text{T3A} & \text{T3B} \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} \text{Signal 1} \\ \text{Signal 2} \\ \text{Signal 3} \\ \text{Signal 4} \\ \text{Signal 5} \\ \text{Signal 6} \\ \text{Interface 1} \\ \text{Interface 2} \\ \text{Interface 3} \\ \text{Interface 4} \\ \text{Node 1} \\ \text{Node 2} \\ \text{Node 3} \\ \text{Node 4} \end{array}$$

Figure 7.9.1: Full-Up Despatch Matrix for the Monte Carlo Simulation test case

As for classical dual-redundant systems, a LOTC will occur if both the 'A' and 'B' instance of any task (here T1, T2, T3) has failed. The LOTC fault tree for the test case is shown in figure 7.9.2. The Monte Carlo simulation uses this fault tree to determine the despatch status.

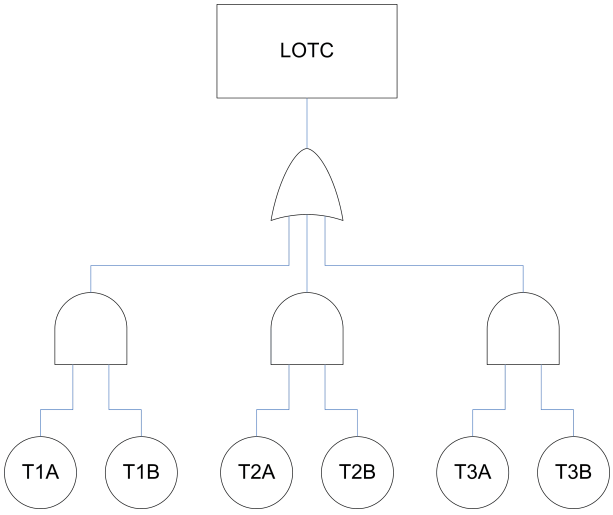


Figure 7.9.2: LOTC fault tree used in Monte Carlo Simulation test case

As described above, each component has a burn-in and wear-out characteristic determined by the shape and scale parameters of the mixed Weibull distribution. The parameters used in the test cases were chosen to be representative of real components and not matched to real reliability data. The failure functions for the first test case are shown in figure 7.9.3.

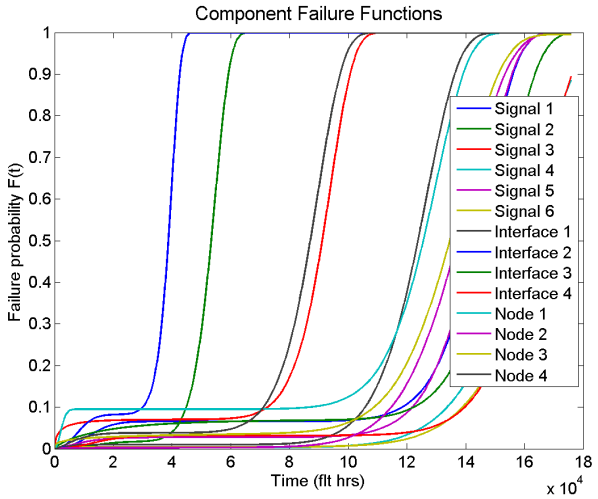


Figure 7.9.3: Component failure functions for the most reliable of three test cases

7.9.1 Test Results

The simulation 'flies' a single system for a 30 year service life (circa 180,000 flight hours) but neglects MLO. In each case, the simulation ceases when the despatch reliability of the system has converged to an accuracy of two decimal places. The graph in figure 7.9.4

shows the number of iterations taken for each of the three test cases to convergence.

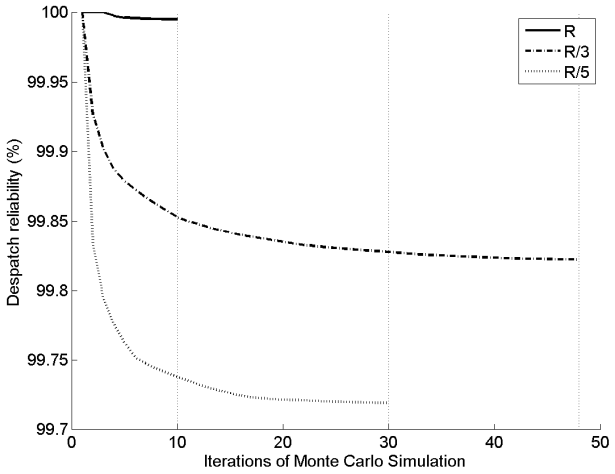


Figure 7.9.4: Convergence of the three lifecycle evaluation function test cases

As the simulation progresses, a cumulative year-by-year record of various performance parameters including the number of cancellations, delays and in-flight shutdowns is maintained. At the end of the simulation, these parameters are divided by the number of iterations to give an average value for each parameter in each service year. Figures 7.9.5 to 7.9.9 show the number of cancellations, delays, the components replaced and the number of USM actions for each of the three test cases.

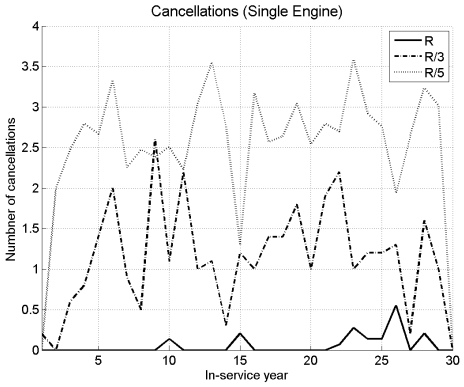


Figure 7.9.5: Annualised cancellations due to control system unreliability and maintenance

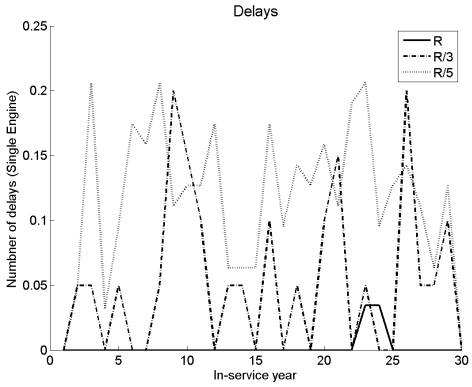


Figure 7.9.6: Annualised delays due to control system unreliability and maintainable

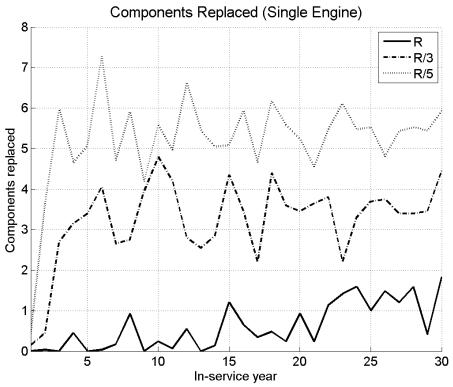


Figure 7.9.7: Average number of components replaced in a given service year

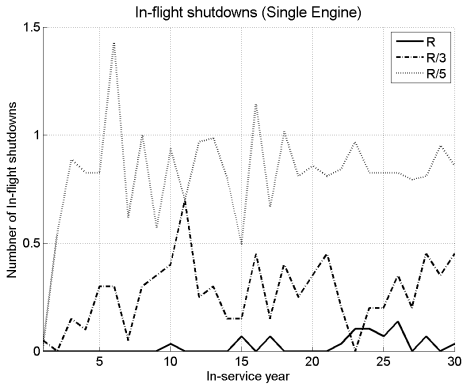


Figure 7.9.8: Average number of in-flight shutdowns due to control system unreliability

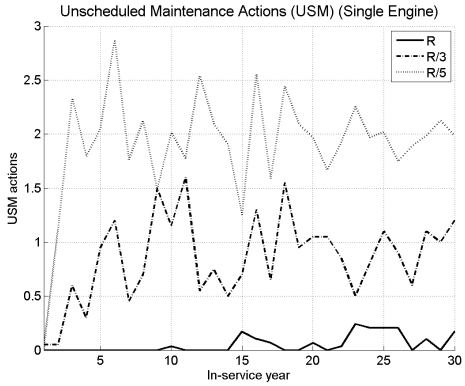


Figure 7.9.9: Average number of USMs in a given service year

We define a vector $\boldsymbol{E(*)}$ where $*$ is the metric name, for each of the metrics shown in the graphs above *eg.* $\boldsymbol{E}(\texttt{cancellations})$. The vector contains the number of cancellations during each service year. The vectors are used by the business evaluation function (chapter 8) to attribute costs to each of the metrics.

The summation of the annualised data gives the total number of events over the lifecycle and may be used to calculate key performance metrics such as the MTTUR and the MTTF. In this instance, there is no MLO and all components are regarded as ‘‘fit-and-forget’’, so the MTTUR is equal to the MTTF. Results are presented in table 7.9.1 below:

Project	**Despatch Reliability (%)**	**MTTUR**(FH)	**Flying hours**(FH)	**Components Replaced**
R	**99.995**	95,220	169,270	22
R/3	**99.822**	5,663	168,886	104
R/5	**99.72**	3965	168,865	132

Table 7.9.1: Monte-carlo simulation results for the three candidate systems

Furthermore, it is possible to isolate unreliable components by considering the number of times they are replaced (figure 7.9.10). This data is useful when isolating technological constraints.

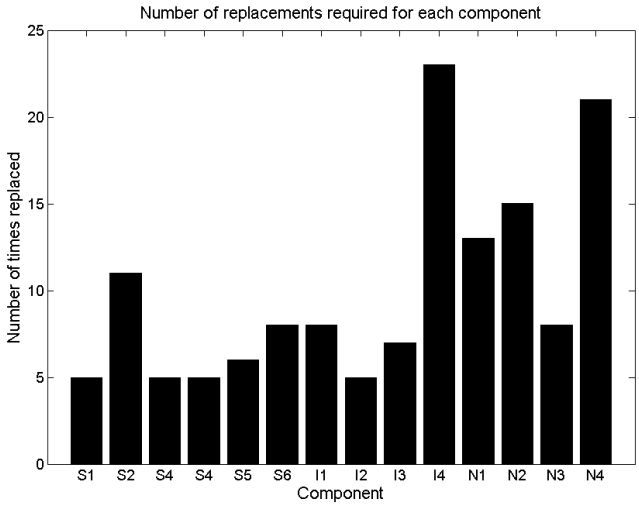


Figure 7.9.10: Number of failures of each component over the service life

In this instance, 'Interface 4', 'Node 2' and 'Node 4' have proven to be particularly unreliable. The MTTUR for each component could be calculated by dividing the number of replacements required by the total number of flight hours.

7.9.2 Analysis

The results presented above show the clear correlation between component reliability and lifecycle performance. Decreasing component reliability has a detrimental effect on performance. Given the arbitrary input data and the non-representative system under test, it is difficult to provide a comprehensive analysis of the results. Principally, these results and the finding of manual verification give confidence that the monte-carlo simulation provides a, 'plausible' analysis of system reliability and lifecycle performance.

Despite the limitations, some features in the output data are typical of characteristics observed in real in-service systems. For example, the number of components replaced each year (figure 7.9.7) has a notable 'frequency' content with equally spaced peaks evident in all three cases. The peaks at year 4, 8, 12 and 15-16 in test case R and years 6, 12, 18 and 24 in test case R/5 are witnessed, if not formerly recorded by airlines. The number of un-scheduled maintenance actions (figure 7.9.9) for case R gradually increases

in the second half the service life whilst the more unreliable cases of R/3 and R/5 require constant maintenance from early on in the lifecycle. Whilst neither of these observations validate the lifecycle evaluation, they give confidence that the monte-carlo simulation provides sufficiently accurate results and considers pertinent metrics.

7.10 Conclusion

This chapter has shown how a Monte Carlo simulation is used to determine the in-service performance of the engine control systems proposed by the GA. The evaluation function 'flies' the distributed system through a 30-year lifecycle operating under Time Limited Despatch. The simulation mimics the actions, behaviours and constraints incumbent on airlines and maintenance technicians. Computational limitations render integration with the main optimisation algorithm impractical. Despite this constraint and the lack of real data for proper validation, the results presented are sufficient to suggest that the algorithm provides a realistic analysis of system reliability. The results and metrics from the lifecycle evaluation are used by the business evaluation function to determine the costs of operating the control system.

7.10.1 Strengths and Weaknesses of the Simulation

The concept of the lifecycle simulation is extensible and could be used to model other aspects of aircraft maintenance strategy were the knowledge and time available. The major benefit of the method is that it reveals ''impact data'' such as delays, cancellations and the number of parts replaced, as well as statistical measures such as despatch reliability and MTTUR. This approach conveys the business concerns of both the airlines and the component manufacturers. Although the structure of the simulation has been heavily contrived to save computation, the basic simulation parameters may be changed to reflect new or emerging operational strategies and constraints. This flexibility is a valuable asset of the method.

Naturally, the algorithm has drawbacks. The following assumptions limit the validity of the simulation:

- All faults occur during flight hours - this is not necessarily the case for real components

- The simulation does not consider the effects of aircraft cycles on reliability
- All components are considered, ‘‘fit-and-forget’’. Although this is the intention of the EEC supplier, this is not necessarily the a reality - an item which is known to be fault prone may undergo preventative maintenance or replacement during scheduled maintenance. This action is not replicated in the simulation.
- All faults are attributable and none are misdiagnosed. Whilst the MCS accounts for the possibility that a fault is not discovered, it does not consider that faults may be misattributed to other components.
- The cost of replacement parts stays constant for the duration of the simulation
- It is assumed that all replacement parts are purchased form the supplier - in practice airlines trade and exchange parts amongst themselves
- The simulation does not consider route-switching or possibility of the plane being used to perform flights in place of other failed aircraft.

Removing many of these limitation would require complex system modelling extending to areas considered beyond the scope of this research. Such areas include supply chain dynamics and aircraft fleet-scheduling.

Whilst the results presented here do not quantitatively validate the MCS, they do provide a qualitative verification of the algorithm's function. The author contends that this verification is sufficient given the scope of this study. The algorithm presented constitutes a significant improvement over the existing algorithms found in both academic literature and industrial use.

Chapter 8

Business Evaluation

The final evaluation function, considers the commercial consequences of the proposed DCS designs. The long product lifecycles and costs of New Product Development (NPD) make the financial returns highly sensitive to in-service performance. In this chapter, it is shown how the metrics from the lifecycle evaluation are transposed to represent the performance of an entire in-service fleet. The business evaluation function is concerned with attributing costs to performance and calculating metrics to establish the quality of investment for each DCS.

The Business view provides an analysis of the operability and viability of the system to both the system developer and the operator (airline). The accounting techniques used to determine the quality of investment are familiar to Aero Engine Controls.

8.0.2 Investment

The commercial model considered herein supposes that the product development and in-service operation are financially indivisible - the money recouped through the flight-hour fee directly covers the investment that funded the design, development and test of the DCS in question. In practice, the company undertaking the project may choose to reinvest income to minimise the value of unfulfilled investments, yet the financial assessment of NPD proposals often follows this assumption. Furthermore, the model assumes that the same company both develops the and supports the product throughout its service life - in the case of Aero Engine Controls, this assumption remains applicable despite being obfuscated by the structure of the joint venture.

The model assumes that the investment necessary to fund NPD is derived from share capital at an annual-effective hurdle rate (discount) of 10%. The hurdle rate accounts for anticipated return on an equivalently sized financial investment, risk and the costs of maintaining non-value added business activities. It would be expected that a prudent investment show an Internal Rate of Return (IRR) of greater than 15% although this is not prerequisite. The investment covers the cost of product design, development, project management, testing, qualification and prototype manufacture. The value of the investment required is equal to the expected cost of the NPD once augmented with inflation and a 20% contingency. In order to maximise returns, the model assumes that the investment is made available annually over the course of the NPD *eg.* a £30m project developed over 5 years is funded by 5 annual payments of £6m.

The predicted value of the investment to the system developer is measured using NPV.

8.0.3 Fleet Profile

All financial calculations are made with respect to the, ''through-life support'' contracts offered by the system provider. Through-life support starts when the product begins development and ends when the last unit is withdrawn from service.

Typically, a control system for a large, civil engine would take between five and eight years to develop and be in production for around ten years. The production volumes rise to a peak at around year 5 and subsequently ramps down *(see figure 8.0.1)*. Some EECs are manufactured for longer periods although this is atypical. It is assumed that all units enter service in the year that they are sold and therefore, manufacture and Entry Into Service (EIS) occur at the same time. The first units enter service in the sixth year and each has a thirty year supported life. Consequentially the first units are withdrawn from service in year 35 with the entire fleet being withdrawn by the end of the 45th year. All distributed control systems are evaluated against the same fleet model.

The graph in figure 8.0.2 shows the EIS profile based on the numbers given the in the paragraph above. Negative values of EIS represent withdrawal from service.

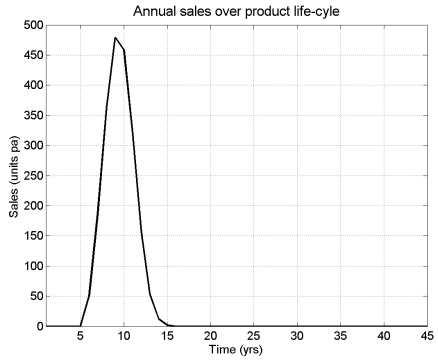


Figure 8.0.1: Total fleet sales

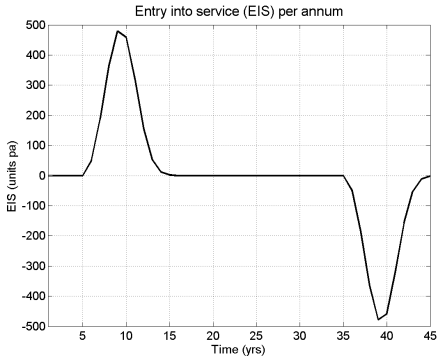


Figure 8.0.2: Entry into Service

The EIS profile is realised using a Weibull distribution. The integral of the EIS profile is the total fleet size which reaches a maximum of 2080 units (figure 8.0.3).

The age of in-service units impacts their reliability and so an accurate picture of fleet performance can only be obtained if consideration is given to the ageing fleet. Figure 8.0.4 shows the fleet age profile over the 45 year support duration. The profile is created by time shifting the EIS profile.

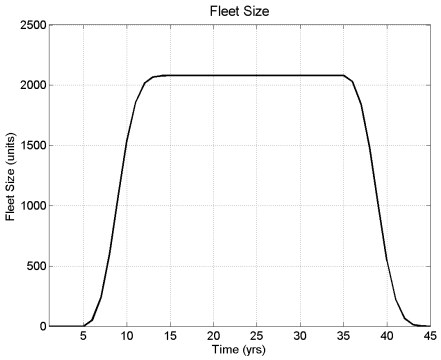


Figure 8.0.3: Total fleet size

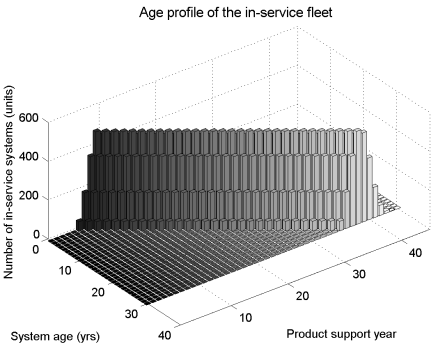


Figure 8.0.4: Fleet age profile

8.1 Determining Fleet Performance

The results obtained from the lifecycle evaluation relate to the performance of a single engine over a thirty year service life. The evaluation function considers that each DCS is a NPD requiring substantial capital investment. The performance of the fleet is determined by the matrix multiplication of the single engine parameters with the fleet-age profile shown in figure 8.0.4. Assuming that the fleet-age profile is expressed as a matrix $\boldsymbol{F_{age}}$ where the rows correspond to the set of service years and the columns to the lifecycle years, the fleet performance of any annualised parameter $\boldsymbol{F}(*)$ may be found by multiplying

the single engine result from the lifecycle evaluation $\boldsymbol{E}(*)$ by the transpose of $\boldsymbol{F_{age}}$. *i.e.* $\boldsymbol{F}(*) = \boldsymbol{E}(*)\boldsymbol{F_{age}}^T$. The parameters once stated over a 30 year service life of a single engine now span a 45 year fleet-support life.

Figures 8.1.1 to 8.1.4 show performance metrics to which costs may be directly attributed. The process of attributing and aggregating those costs is described presently. The metrics presented in section 7.9.1 have been scaled up to fleet level using the multiplication given above.

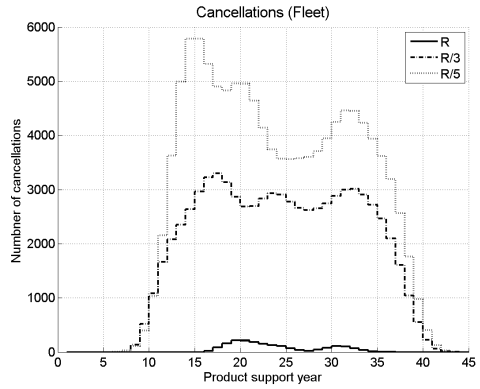


Figure 8.1.1: Fleet Cancellations

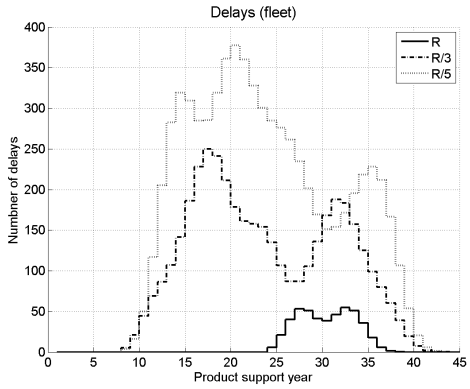


Figure 8.1.2: Fleet Delays

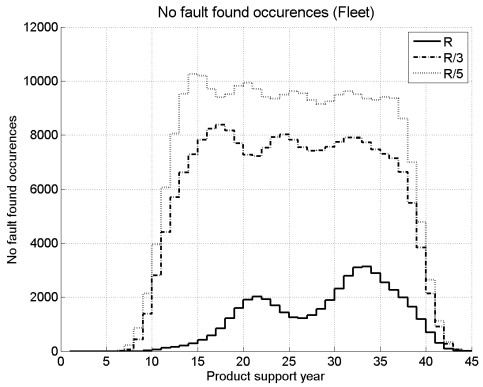


Figure 8.1.3: Fleet NFF

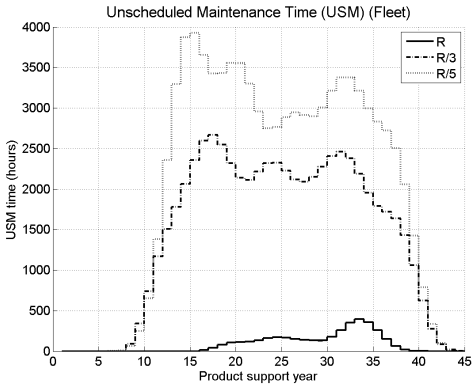


Figure 8.1.4: Fleet USM time

8.2 Attributing Costs and Incomes

Having ascertained the fleet performance, each metric is attributed to an annualised cost or income. These costs and incomes are used to calculate the NPV of the initial investment and from this, the various financial parameters that define its quality.

In the following paragraphs, we assume that the notation $\boldsymbol{F}(*)$ represents the vector of fleet performance results for the metric $*$. *eg.* $\boldsymbol{F}(\texttt{cancellations})$. Costs are denoted by $\boldsymbol{C}(*)$ and incomes by $\boldsymbol{I}(*)$. The value of the metric in the n-th year is $\boldsymbol{F}(*\texttt{[n]})$. Specific

costs are denoted by $£_{cost}$.

Incomes

Incomes are generated from unit sales and engine flight-hours. Units are sold at a fixed cost regardless of the design and manufacturing costs. Therefore, the more cheaply the unit is produced, the more profit is made from sales.

$$\boldsymbol{I}(\texttt{unit}) = \boldsymbol{F}(\texttt{sales}) \times £_{unit\ sell} \qquad [£] \tag{8.1}$$

The flight hour income is considered fixed for the duration of the service life. Therefore:

$$\boldsymbol{I}(\texttt{flight hour}) = \boldsymbol{F}(\texttt{flight hour}) \times £_{fleet\ hour\ sell} \qquad [£] \tag{8.2}$$

Where $£_{fleet\ hour\ sell}$ is the cost of a single flight hour to the operator.

Costs

The fleet hour cost covers the overheads of operating a through-life support operation. Such overheads include staffing, administration, building maintenance and other non-value added items. This does not include the cost of performing unscheduled maintenance.

$$\boldsymbol{C}(\texttt{flight hour}) = \boldsymbol{F}(\texttt{flight hours}) \times £_{fleet\ hour\ cost} \qquad [£] \tag{8.3}$$

Formerly, the subsystem supplier will pay a fee for every Delay & Cancelation (D&C) associated with the failure of their component. In practice, the payment is rarely claimed or honoured and it is assumed that only 5% of D&C claims are successful. The cost of D&C penalties is calculated by:

$$\boldsymbol{C}(\texttt{DandC}) = (\boldsymbol{F}(\texttt{delays}) + \boldsymbol{F}(\texttt{cancellations})) \times 0.05 \times £_{DandC} \qquad [£] \tag{8.4}$$

It is assumed that 40% of nodes returned to the subsystem supplier are later found to have no fault. Each instance of NFF has an associated cost to cover test and adminstration:

$$\boldsymbol{C}(\texttt{NFF}) = \boldsymbol{F}(\texttt{NFF}) \times £_{NFF} \qquad [£] \tag{8.5}$$

The costs of USM is quoted in £/hour and considered a cost of doing business. The hourly rate covers the costs of staffing and administering the maintenance operation. The cost of spare parts is considered separately. Therefore, the cost of USM is:

$$\boldsymbol{C}(\texttt{USM}) = \boldsymbol{F}(\texttt{USM hours}) \times \pounds_{USM\ hour\ rate} \qquad [\pounds] \tag{8.6}$$

Materials and labour costs for unit production are owned by the subsystem supplier.

$$\boldsymbol{C}(\texttt{unit}) = \boldsymbol{F}(\texttt{sales}) \times \pounds_{unit\ cost} \qquad [\pounds] \tag{8.7}$$

The cost of replacement components was calculated during the monte carlo simulation (section 7.9.1). Therefore, $\boldsymbol{C}(\texttt{componets}) = \boldsymbol{F}(\texttt{components})$

Total Incomes and Costs

By summing the incomes and costs stated above, the total costs and incomes $\boldsymbol{C}$ and $\boldsymbol{I}$ my be calculated on an annualised basis. The total in income is:

$$\boldsymbol{I} = \boldsymbol{I}(\texttt{unit}) + \boldsymbol{I}(\texttt{flight hour}) \qquad [\pounds] \tag{8.8}$$

And the total costs $\boldsymbol{C}$:

$$\begin{aligned} \boldsymbol{C} = {} & \boldsymbol{C}(\texttt{investment}) + \boldsymbol{C}(\texttt{flight hour}) + \boldsymbol{C}(\texttt{DandC}) + \boldsymbol{C}(\texttt{USM}) \\ & + \boldsymbol{C}(\texttt{unit}) + \boldsymbol{C}(\texttt{componets}) + \boldsymbol{C}(\texttt{NFF}) \qquad [\pounds] \end{aligned} \tag{8.9}$$

8.3 Present Values, Net Present Value and Rate of Return

These incomes and costs are discounted back to their present value using equation 8.10:

$$\boldsymbol{P}\texttt{[n]} = \frac{\boldsymbol{I}\texttt{[n]} - \boldsymbol{C}\texttt{[n]}}{(1+0.10)^n} \quad \text{for all n} \tag{8.10}$$

Therefore the net present value of the initial investment is:

$$NPV = \sum_{n=1}^{Y} \boldsymbol{P}\texttt{[n]} \qquad [\pounds] \tag{8.11}$$

Where Y is the product support duration in years. Y is equal to the development time + manufacturing time + service life. In this case the product support duration is 45 years.

The NPV is the total value added to the company by pursuing the development of the DCS in question.

The break even year is simply the last year at which the present value rises above zero pounds.

The NPV is perhaps the most important parameter calculated during the optimisation process and will ultimately define the worthiness of the project. Any project with a negative NPV should be discarded.

8.3.1 Rate of Return (IRR and MIRR)

The IRR and Modified Internal Rate of Return (MIRR) are other important measures of the investment quality relating to the rate of earnings. Both measures are complementary to NPV. The IRR is the effective interest rate i that makes the NPV equal to zero:

$$\sum_{n=1}^{Y} \frac{\boldsymbol{I}\texttt{[n]} - \boldsymbol{C}\texttt{[n]}}{(1+i)^n} = 0 \tag{8.12}$$

The secant method is used to find the vale of i that makes the NPV zero. Whilst the IRR is indicative, it does not account for reinvestment of earnings into other company projects. This means that IRR projections are generally optimistic and not a real-world measure of rate of return. The MIRR corrects for this by including a reinvestment rate in the return values. The MIRR is calculated by:

$$MIRR = \sqrt{\frac{\sum_{n=1}^{Y} \frac{\boldsymbol{I}\texttt{[n]}}{(1+i)^n}}{\sum_{n=1}^{Y} \frac{\boldsymbol{C}\texttt{[n]}}{(1+r)^n}}} \qquad [\%] \tag{8.13}$$

Where i is the discount rate and r the reinvestment rate. The reinvestment rate is set 1% higher than the discount rate at 11%.

8.4 Results

The results presented here are based on the performance parameters generated by the lifecycle evaluation in chapter 7. The results show how the most reliable set of components produces the greatest financial return. As the Monte Carlo simulation is detached from the

architectural evaluation, all projects are deemed to cost £40m over a 5 year development period. Each system costs the same amount and the sales volume is identical for all projects.

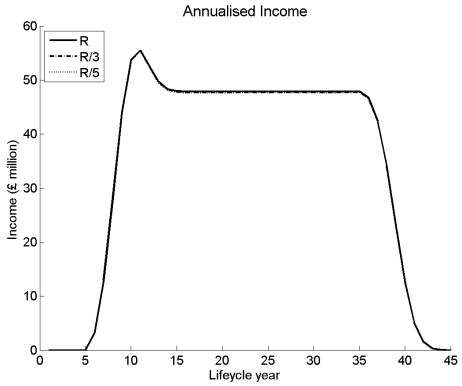


Figure 8.4.1: Total annualised income

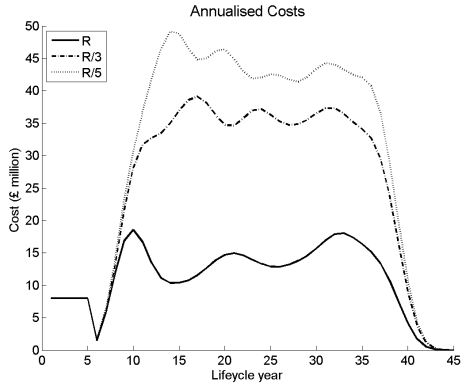


Figure 8.4.2: Total annualised costs

The income graph figure 8.4.1 shows the five year development time in which the company receives no money. The peak between years five and ten is due to income from unit sales. The remaining income is from the flight-hour fee. The initial £40m investment is spread across the 5 year development process.

The costs (figure 8.4.2) are more complex. The costs from years five to ten are associated with unit production. Later in the product support life, costs are due to component replacement, maintenance time and D&C penalties. The incomes and costs are used to calculate the NPV of the £40m investment.

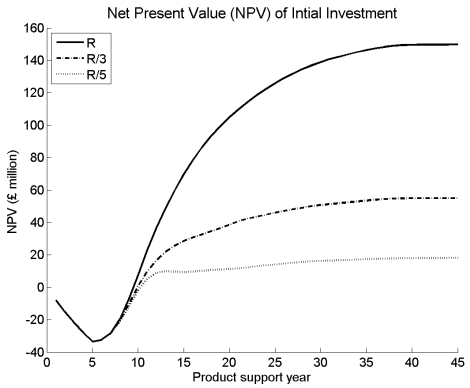


Figure 8.4.3: Annualised present values of initial investment

Table 8.4.1 shows the total expenditure and returns for each of the three projects.

Table 8.4.1: Total costs and returns associated with the three NPV projects.

Project	**Maintenance**(£m)	**D&C**(£m)	**NFF**(£m)	**NPV**(£m)	**IRR** (%)	**MIRR**(%)
R	4.78	3.47	93.37	**149.72**	28.61	12.62
R/3	75.49	120.09	434.4	**55.04**	21.36	11.1
R/5	107.9	186.55	549.9	**18.06**	15.7	10.69

The break even years are 10, 10 and 11 respectively.

8.5 Conclusion

As anticipated, the three projects are distinguished by every available metric. There is a clear relationship between reliability and NPV. Given that the dataset and attributed costs are arbitrary, it is difficult to draw meaningful conclusions from the results, other than to demonstrate the plausibility of the business evaluation function. The results presented verify the performance of the evaluation function and could be validated against real financial records should the reliability data for system components be available.

Part III

Conclusion

Chapter 9

Conclusions

This thesis has presented a novel method for the architectural optimisation of Distributed Control Systems (DCSs) for large, civil jet engines. The method is capable of optimising DCS architectures as well as comparing candidate solutions from a number of different viewpoints. The extensibility of the method and the number of design considerations addressed are significant strengths of the approach. Whilst it has not been possible to realise a full implementation, the results presented in chapters 6, 7 and 8 demonstrate the potential of the method and give confidence that a full implementation would produce the desired outcomes.

Fundamentally, the aims declared in the introductory chapter have been achieved. The results given in the architecture evaluation chapter show the architectures produced and the architectural, life-cycle and business evaluation functions ensure that the architectures are devised using a novel systems engineering approach.

A comprehensive understanding of the DCS design problem has been fundamental to developing the method. The evaluation functions incorporate a breadth of technical and commercial considerations that reflect the industrial context of the research. Having realised the multi-objective nature of the design problem, the Genetic Algorithm (GA) was considered the most suitable means of addressing the trade space and an appropriate algorithm was selected. The architecture frameworks provided a solid foundation for building the metaphysical models and the binary relations an effective means of analysis. The tools and techniques chosen to construct the models proved very effective - the application of graph theory approaches to harness routing and network layout meant

that otherwise complex construction tasks could be greatly simplified. The evaluation functions provided information relevant to both system architects and business leaders. In these respects, the work presented has realised the objectives stated in chapter 1.

9.1 Contributions

This research has made the following contributions:

- Novel method for the architectural optimisation of distributed control systems. The method permits multiple design considerations and could be applied to many different systems.
- The use of binary relations in the composition and analysis of architectures based on architecture frameworks.
- A Monte Carlo simulation (MCS) for a large-civil jet engine control system operating under Time Limited Despatch (TLD). The simulation considers the implications of component failures to both the operator and through-life support provider.

Chapter 3 and section 7.2 provide literature surveys that clarify these contributions.

9.2 Strengths and Weaknesses of the Method

The principle strengths of the method presented are extensibility and scope. Whilst the architectural frameworks and evaluation functions presented pertain to DCSs for jet engines, the principles could easily be applied to other complex systems. The decision criteria and decision variables could be modified to add or reduce the number of factors considered.

Despite being computationally expensive, it is shown in Appendix B that the scheme could be made feasible using only a small array of parallel processors. This is important for acceptability amongst industrial practitioners.

9.2.1 Potential Improvements

As with all projects, there are many ares where the implementation could be improved for both accuracy and performance. An enhanced algorithm should aim to:

- Use a more realistic and detailed data set
- Apply the optimisation to a larger number of engines
- Define component reliability characteristics using real world data
- Add dynamic models for design considerations such as thermal soak, vibration analysis and the engine control laws.
- Increase the number of redundancy schemes available
- Improve the harness routing algorithms
- Optimise the location of Full Authority Digital Engine Controller (FADEC) subsystems beyond the Electronic Engine Controller (EEC)
- Include additional analyses in the evaluation functions:
  - Redundancy analysis
  - Better product costing
  - Fly two or four DCS on an airframe in the monte-carlo simulation
  - Consider technology maturity and Technology Readiness Levels (TRLs)
- Extend the business evaluation to encompass system re-use across future products.
- Use a larger number of derived relationships for analysis

...and modifications to refine the implementation and realise a more viable tool:

- An improved software architecture
- Design for implementation on a parallel computing platform
- A structured approach to adding and removing evaluation criteria and model elements

These additions would engender a more rigourous and, ‘‘industrial strength’’ tool capable of finding better founded solutions; their absence is indicative of the resources afforded to this research rather than the inability of the method to accommodate them. Applying these modifications would not fundamentally change the approach or its underling premises.

9.2.2 A Modified Approach

There are number of weaknesses in the method itself which undermine its value as an approach for architectural design. Firstly, the DCS are optimised for a single engine and not a product family. It is a commercial necessity to formulate architectures that could be deployed across a wide range of engines.

The approach relies heavily on a large and complete data set. The method cannot cope with ambiguity and results are dependent on the data employed. Furthermore, the algorithm is computationally expensive.

Most critically, the method does not determine the facets of 'good' distributed architecture. Although the algorithm provides a best solution, it is difficult to draw conclusions from the data produced. Poor solutions are neglected as the optimisation progresses. These limitations mean little is learned about the facets of good architecture and the shortcomings of poor architecture. In this respect, the method presented is flawed and requires modification.

An alternative approach *(see figure 9.2.1)* would require the optimisation process to develop a strategy rather than a one off solution. The chromosome may encode strategic options such as, ''node on every sensor'' or ''make Fuel Metering Unit (FMU) node'' rather than freely defining an architecture. This strategy could be evaluated across a product family of engines with differences in environment, dimensions and control system functionality. A product family could include engines from the large civil, regional and business jet sectors as well as fast jet and military transport engines.

Rather than relying on the GA alone, Particle Swarm Optimisation (PSO) could be used to identify the worthwhile areas of the search space. PSO is widely regarded as being more computationally efficient and able to cover large areas of the search space more quickly. The better solutions from the PSO could be used to seed the GA. The GA would refine the solutions.

Once evaluated on all these engines, the chromosome and fitness values could be stored in a database or file structure based on a binary tree. Not only would this provide an efficient method of storing all the solutions (they could all be reconstructed from their chromosomes) but also a fast access lookup table allowing the GA to avoid evaluating any solution it or the PSO has encountered previously.

Rather than running the optimisation for every number of nodes and combining the outputs, the solutions could be built in such a way that if no tasks or interfaces are allocated to the node, then the node ceases to exist. This approach would add complexity, but permit the optimiser to consider a range of node values in a single execution.

The most important addition to the scheme is the Bayesian network. The network could use the data stored in the tree structure to 'learn' how the architectural strategy

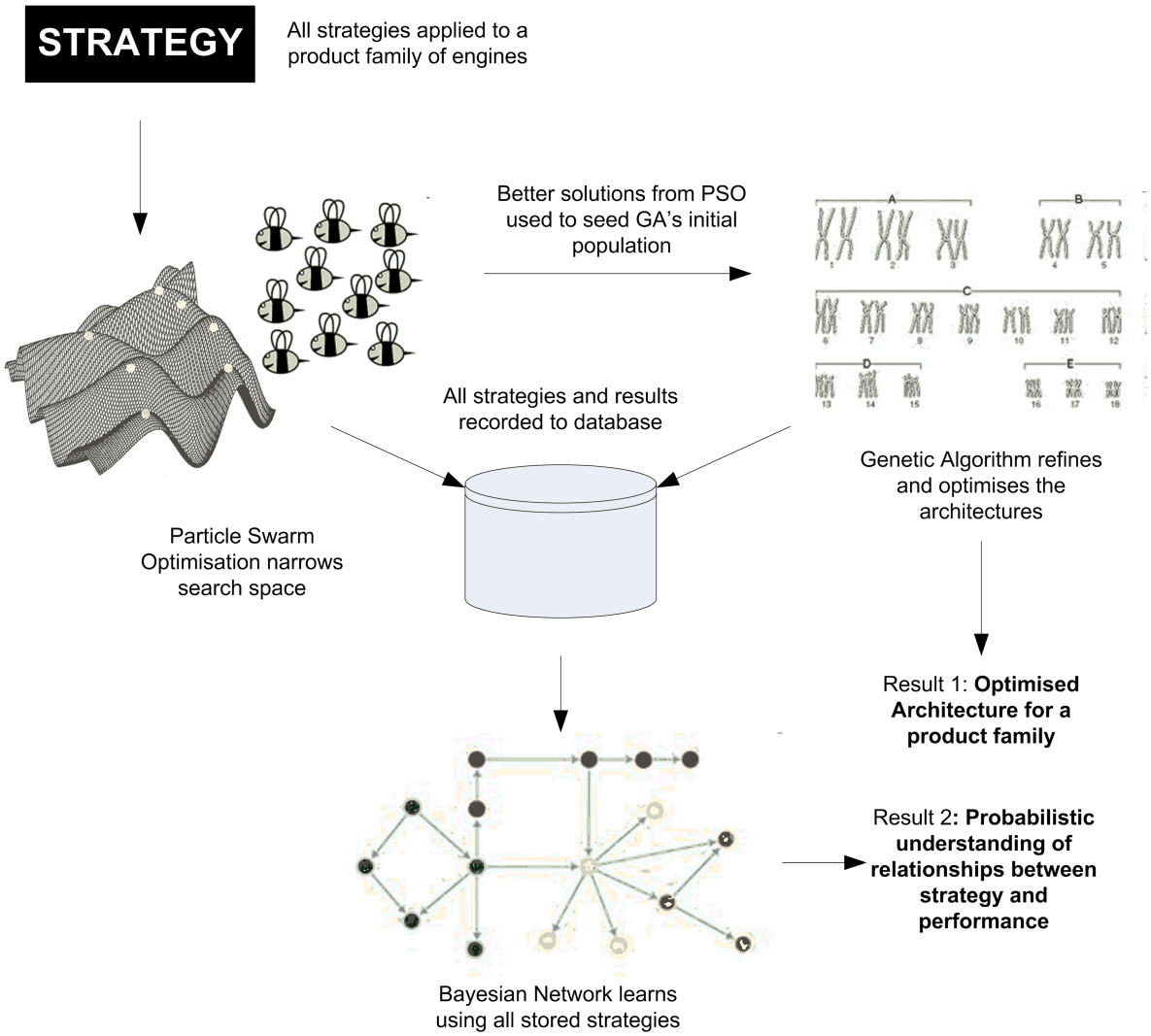


Figure 9.2.1: A modified approach using PSO and a Bayesian network to better understand the consequences of architectural choice.

affects the fitness of the solution.

The scheme presented above should provide two different results: firstly, an optimised architecture similar to that produced by this work and secondly, a set of probabilistic relationships between the strategic options and their lifecycle performance.

9.3 Related Research Topics

Aside from augmenting and improving the algorithm presented here, there are many other areas where research is required to further both this method and the practice of DCSs design. Whilst vital to their adoption, research in other contributing disciplines such as High Temperature Electronic Devices (HiTEDs) and high-temperature polymers is ignored herein.

It is the author's opinion that the two most pressing areas of research are concerned with novel redundancy schemes and surviving element networks. Both are intuitively simple concepts with complexities that belie their apparent simplicity. Study in these areas should reveal how best to structure DCS in order to maximise reliability and despatch configurations whilst reducing the amount of hardware. Knowledge from both areas could be readily incorporated into the optimisation method presented in this thesis. Research in both areas would prove valuable in the development of a real time operating system for deterministic DCSs; a fundamental technology yet to be realised. Surviving element networks allow working parts of two damaged or dysfunctional networks to combine to realise a functional system. The effect is readily achieved in switched networks such as ethernet, but rarely considered for real-time, fully deterministic systems. The two component networks need not have identical topologies or be in themselves complete. Research into surviving element networks would need to account for both topology and network scheduling.

Novel redundancy schemes (briefly covered in section 5.9) are related to surviving element networks but could be studied separately. Research is required to establish the failure modes and probabilities for systems without distinct dual or triple redundant channels.

One of the many stated benefits of distributed systems is their ideality as a platform for hosting advanced health monitoring and control algorithms. Such schemes include

life-extending control, model based fault detection, adaptive control and on-line fault isolation. General wisdom concludes that such schemes are best implemented on DCSs where additional processing power is available and components are more naturally isolated. Researchers studying these techniques for gas turbine applications often take this premise for granted. However, all these mechanisms require many parameters from many different sensors and actuators to update the models on which they depended. It is the author's opinion that the number of parameters passed across the network and the duplication of model elements on different nodes may ultimately become a centralising force. A study addressing the application of these techniques and their influence on decentralisation would have both technical value and guide future research in the area.

Further to analysis above, the partitioning between control and health monitoring functionality requires significant research and technical development. The two functions are linked by implementation. The problem is complicated further by control algorithms which use health monitoring data to refine their behaviour. Research using methods akin to those presented in this thesis is under way at the Rolls-Royce University Technology Centre (UTC) at Sheffield University.

More generally, research into DCSs should focus in areas that distinguish distributed architectures from their centralised counterparts. At present, there are few reasons to invest in distributed systems because the proposed designs offer little more than present day EECs.

From a systems engineering perspective, much work is required to better understand the relationship between architectural decisions and through-life performance. This thesis provides one such method with many shortcomings. It is the author's opinion that systems engineering relies too heavily on process rather than mathematical and technical formalism to derive its outcomes. This research has shown that graph theory holds a vast potential for application to architectural design and evaluation.

9.4 Resume

The decisions faced by system architects are becoming increasingly difficult. As the practice of systems architecting becomes more widely recognised, so greater scrutiny is placed on architectural decisions and their technical and commercial implications. The

growing complexity of modern systems makes the process of defining and validating architectural designs increasingly difficult. Today's system architects are charged with developing effective and viable solutions without the aid of the formalisms and methods found in more traditional engineering disciplines.

The method presented here goes some way to addressing the difficulties of architectural design. It is not a complete or universally applicable method but could be tailored to suit any number of complex systems where contending stakeholder demands complicate the solution space. As the practice of architectural design becomes ever more demanding, so systems engineers will require new formalisms and optimisation techniques to approach many of the considerations addressed in this research. Companies such as Rolls-Royce recognise that methods akin to this will play an important part in the future of system design. This thesis does not present an immediate solution to the challenge of designing value into complex system architectures. However, the approach considered and results presented yield a modest contribution to furthering the art.

Appendices

APPENDIX A

The Data structure and database

The relational database holds all the information not dynamically created during optimisation. This data includes the engine dimensions, the relationships in the control system model and the circuit block specifications. The database is designed to work across a complete product family and therefore permits serval engines and their associated control systems be stored concurrently. Like the metaphysical models, the database structure is aligned with the architecture frameworks.

The database comprises a set of tables. Each table is associated with either an entity or relationship from one of the three architecture frameworks. For example, the 'Tasks' table holds a list of all the tasks performed by the control system and the Interface-Parameter table holds information associating interfaces with the parameters they provide.

Every instance of an entity is uniquely identified by a *primary key*. The primary key is an integer value that, when incorporated into the metaphysical model becomes the index of the entity within its set. For example, if the primary key of the parameter 'P50' is 4, $P(4) = P50$ - the database is designed in such a way that primary keys cannot be replicated within a given table. In most instances, the primary key indicates the order in which the entities were entered into the database. This order is unimportant to the optimisation process and does not influence outcomes. Entity tables contain information about the specific entity. For example, the 'Engine' table shown in figure A.0.1 holds the engine dimensions.

Relationships are stored in entity-relationship tables. Each relationship (an instance of a relation) is identified by a primary key. Relationship tables comprise the primary key,

Primary Key	Engine Name	IPC Length	HPC Length	Fancase Overlap
1	Trent 1000	0.75	0.82	0.91
2	RB211	0.6	0.75	0.82

Figure A.0.1: Example of database entity table

the primary key of the first entity (known as a foreign key) and the foreign key for the related entity. An example is given in figure A.0.2.

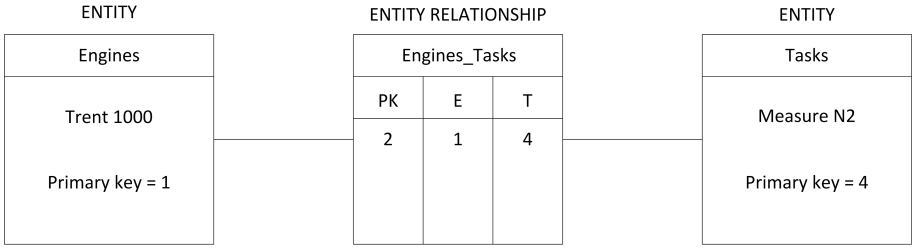


Figure A.0.2: Example entity relationship

The database is realised using a MySQL® relational database. MySQL® is an industrial-strength open-source database solution adopted by many large enterprises and web developers.

Data from the database is read into MATLAB® prior to commencement of the GA using THE MATHWORKS™ database connection toolbox. The data is used to populate the relational matrices associated with the hybrid framework. There is no need to re-read the data or communicate with the database once the optimisation has started.

The database is relational to 3rd normal form. [1]. The database is constructed and populated using Structured Query Language (SQL) scripts.

[1]Information stored in relational databases may be normalised to minimise replicated and unstructured data. There are seven normal forms - third normal ensures that the integrity of the data is maintained whilst maintaining structural simplicity.

APPENDIX B

Parallel Processing

B.1 Introduction

The method used to optimise the DCSs is computationally demanding. Nearly every aspect of the of algorithm is iterative and required to perform a substantial number of calculations on a moderately-sized data set. Proposed extensions such as the use of Ant Colony Optimisation (ACO) for harness routing and a broader lifecycle simulation would increase the demand further. When compared to weather forecasting, seismic monitoring and medical imaging, the computational power required to realise a full implementation is trivial. However, the computational resources usually afforded to these problems vastly surpass those available to this research.

Meta-heuristic methods, computational models and search algorithms are increasingly implemented on computational clusters. As single-core silicon processors near their theoretical limits, the accrual of multiple of processor cores, sockets and machines has become the accepted means of obtaining a more capable computing platform.

This chapter examines future implementation of the optimisation scheme on a parallel computing platform. Through analysis of the tasks itself and the various methods of implementation, we determine indicative values for the potential performance gains.

B.2 Steps taken to reduce computational burden

Whilst the monte-carlo simulation is the obvious culprit, there are numerous 'hidden' elements of the optimisation routine that require significant computational resource. The

mixed Weibull distribution used in the lifecycle evaluation and the Internal Rate of Return (IRR) calculation in the business evaluation cannot be solved algebraically; they require numerical solution using the iterative root-finding secant method. Furthermore, the Strength Pareto Evolutionary Algorithm (SPEA2) is computationally intensive when compared to other GAs less suited to the application.

Reducing computational effort has been a principle concern during software development. Where possible, code has been written in an efficient manner and the adjacency and distance matrices used during harness routing are calculated prior to optimisation. The routine used to calculate the matrices is a core-divisible programme written in C++ and OpenMP™(see section B.4.2). Furthermore, each DCS is evaluated on only one engine and the resolution of the engine chassis stations is fixed to a 5cm×5cm grid. Such a coarse grid is obviously inconsistent with the tolerances applied during real engine design.

These efficiencies are realised against significant compromise: The programme code for the monte-carlo simulation has been contrived to avoid sorting and searching and is unduely complicated and difficult to modify. Calculating the adjacency and routing matrices prior to optimisation makes it prohibitively difficult to change the engine dimensions, keepout zones and routing criteria during and between optimisations.

B.3 Factors Influencing Parallel Computing Performance

Effective use of parallel computing resources is far from simple. There are many factors which determine how efficiently a large scale computing problem may be parallelised. They include:

1. The nature of the problem and the potential for parallelisation.
2. The number of serial functions and the extent of data-interchange between functions
3. Whether the problem is data intensive, computationally expensive or both (counter-intuitively, problems are rarely both)
4. The efficiency and execution speed of the programme code - dependent on both programme structure and compiler performance
5. Virtual machines, hypervisors and abstraction layers between application code and the microprocessor

6. Parallelisation and memory management strategy
7. The number of cores, sockets and multi-threading capability
8. Bandwidth of cluster backbone or socket to socket to communications
9. Interface and peripheral resource contention

The considerations listed above may be grouped into three ares of consideration: Items 1 to 3 are concerned with parallelising the task itself, items 4 and 6 are concerned with the software platform and items 7 to 9 relate to the underlying hardware.

B.4 Performance Gains

The following section considers methods of determining approximate performance gains that could arise from migration to a parallel computing platform. Gains in the task formulation, software and hardware are considered. In this document, gains are stated as a reduction factor referenced against the predicted execution time for the present platform: this platform constitutes a dual core, 1.2GHz processor without significant cache memory or hyper-threading capability. The hardware executes a programme written in object-oriented MATLAB® and only capable of using a single core. The system has 1500MB of RAM and the operating systems starts to page memory at around 90% of RAM utilisation. The hardware executes a programme written in object-oriented MATLAB® working on a data set of approximately 600MB. Tests suggest that the current platform would take approximately 2500 hours (15 weeks) to complete the optimisation. The optimisation requires a data set of around 600MB and is considered to be computationally intensive and data lite.

B.4.1 Gains from Task Parallelisation

Accurately assessing the performance gains associated with task parallelisation is very difficult. Consequentially, there are few formalisms or heuristics which relate the available processing power to gains in execution time. Amdahl's law (Hill & Marty, 2008) provides an estimate of the maximum possible speed gain which may arise by parallelising an algorithm with a parallel portion p:

$$G_T = \frac{1}{\left(\frac{p}{n_c}\right) + (1-p)} \tag{B.1}$$

Where G_T is the maximum possible gain in task execution speed given a resource of n_c processor cores. The parallel portion p is the percentage of the task which can be executed in parallel and is very difficult to quantify. Whilst not a foolproof definition, the parallel portion is a ratio (expressed as a decimal) of the number of executable operations which require serial implementation to the number of operations that may be executed in parallel. Alternatively, Amdahl's law says that if there is a $(1-p)$ serial component in the programme structure, then the speed increase gained through parallelisation, cannot exceed $100/(1-p)$. Amdhal's law considers only the structure of the task - it should not be assumed that exploiting all the opportunities for parallel execution is technically possible.

The DCS optimisation task maybe readily parallelised at three different levels as shown in figure B.4.1. The deepest level of parallelisation (level 3) involves executing the evaluation functions in parallel. Whilst this approach is likely to provide a measurable reduction in processing time, the business and lifecycle evaluation functions are serial processes and will take significantly longer to execute than the architectural evaluation. This approach would make inefficient use of available hardware resources. Alternatively, the algorithm could be parallelised at the solution level (level 2) meaning that the GA would have to wait whilst all solutions are evaluated before moving to the next iteration. This approach is more desirable and would make better use of the hardware but requires synchronicity between iterations of the genetic algorithm. The highest level of parallelisation (level 1) is the node level - essentially, each core optimises for a different number of nodes thus reducing the data passed between tasks and the need for synchronicity. It is common place to parallelise at the highest level and providing that the number of processing cores does not exceed the maximum number of nodes, this is the most readily achieved and computationally efficient of the approaches.

Determining a parallel portion for the DCS optimisation is almost impossible without experimentation: a plausible value of $p = 0.85$ is used herein.

Amdahl's law provides an expression for the potential speed gain associated with task parallelisation. A corresponding rule for hardware and software performance are required

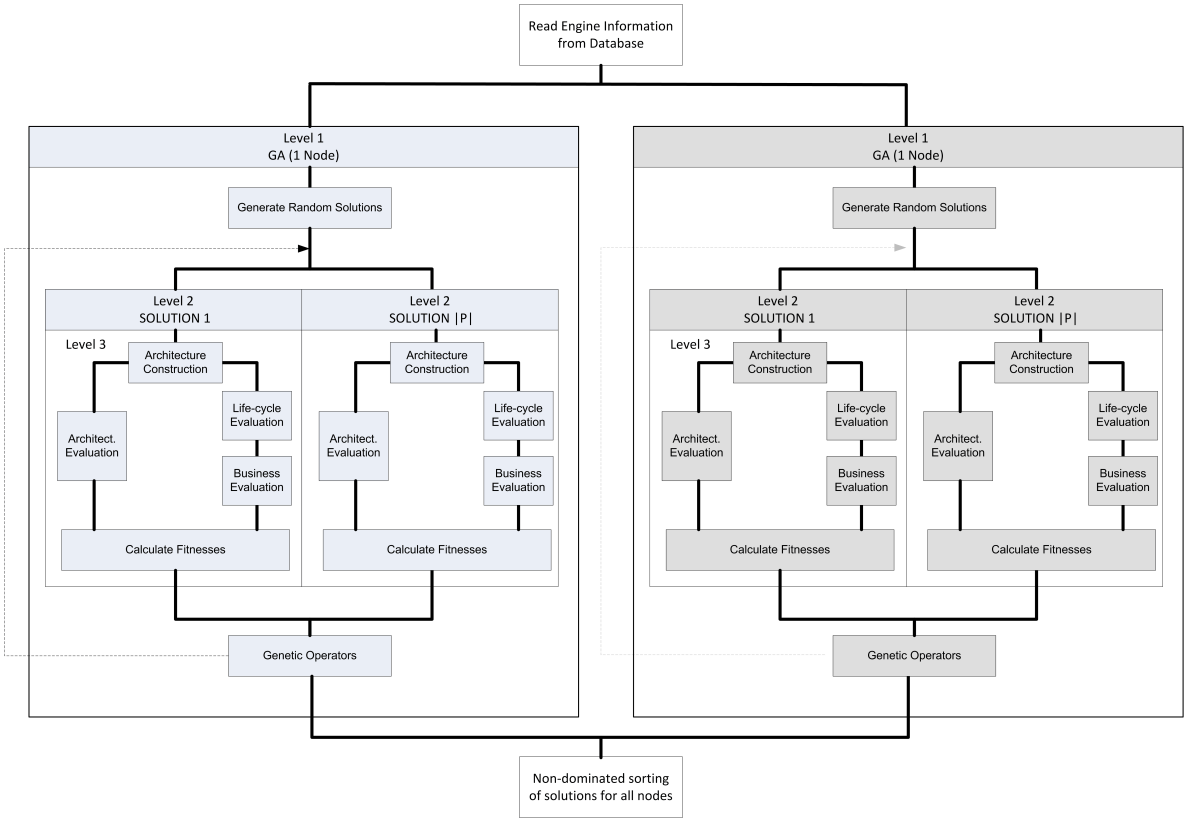


Figure B.4.1: Three different levels at which to parallelise the DCS optimisation

to establish the overall speed gain.

B.4.2 Gains from Software and Implementation

By comparison to C or C++, MATLAB® code is slow to execute. MATLAB® is an interpreted language running on a Java Virtual Machine that abstracts the programme form the assembly level instructions. MATLAB® lacks facilities such as pointers and advanced object-oriented constructs which increase the volume of data passed and replicated been functions. Object-oriented MATLAB® is necessary for a project of this scale but executes more slowly than its linear counterpart.

Whilst gains are application dependent, estimates suggest that a programme written in C++ would execute 10 to 100 times faster than the equivalent MATLAB® code. Compiling C/C++ programmes using a processor specific compiler would also reduce execution time.

Both MATLAB® and C++ offer the facility to parallelise code. MATLAB® offers parallelisation through the THE MATHWORKS™ Distributed Computing Toolbox and C++ via various Advanced Peripheral Interfaces (APIs) including OpenMP™. OpenMP™offers

two methods of parallelisation whereby processes or memory blocks may be associated with parallel threads. MATLAB® offers only loop level parallelisation and would prove less efficient. No parallelisation will be 100% efficient.

Determining a ''software'' gain, G_s is largely based on informed assumptions. Assuming that C++ would execute 50 times faster than equivalent MATLAB® code and that parallelisation using OpenMP™ is 80% efficient, then the execution time will decrease by a factor of 40. $\therefore\ G_s = 40$. The parallelisation efficiency is a measure of the percentage of the maximum task parallelisation (section B.4.1) that is realisable. Inefficiencies result form memory contentions, the need for task synchronisation, scheduler inefficiencies and and software overheads.

B.4.3 Hardware Gains

It cannot be assumed that doubling the number of cores will halve the execution time. The transfer of information between cores and multi-threading capability mean scaling factors apply. For parallel computing on a single Central Processing Unit (CPU) with multiple cores, the effective speed of the secondary cores is reduced by a factor known as the ''core-to-core scaling factor''. This scheme is illustrated in figure B.4.2. A similar ''socket-to-socket scaling factor'' exists between different CPUs sharing the same motherboard figure B.4.2. As both scaling factors depend on the hyper-threading capability of the processor and the number of threads available to the scheduler it is impossible to define an application independent value for either parameter. Articles from computer industry press suggest values in the range of 0.7 to 0.8 for both core-to-core and socket-to-socket scaling factors are typical.

By using the models presented above, we can develop an expression for the likely gain speed gain based on the hardware architecture alone.

$$G_H \approx s_s n_s (1 + c_c(n_c - 1)) \tag{B.2}$$

Where n_s is the number of sockets, s_s is the socket-to-socket scalability factor, n_c is the number of cores and c_c is the core-to-core scalability.

The expression is crude but considered indicative. It does not account for interface resource contests, data transfer between parallel tasks on different sockets or machines,

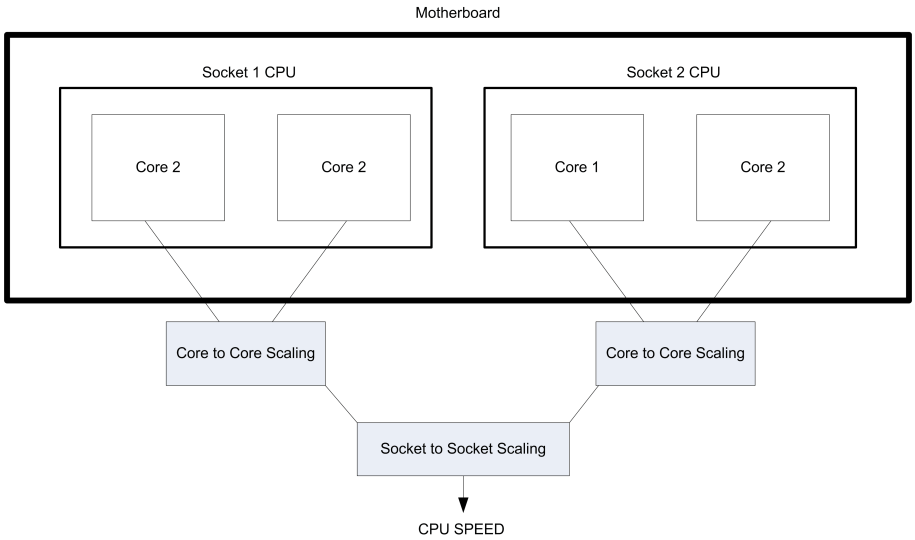


Figure B.4.2: Core to Core and Socket to Socket Scaling Factors as applied to a single motherboard

nor the efficiency of the scheduler. Furthermore, the equation assumes that each processor has the same number of cores and identical clock speed - this commonplace but cannot be assumed. Given that the DCS design problem is highly parallelisable, uses a minimal data set and does not require access to external peripherals, we can assume the expression to hold reasonable value.

The raw increase in core clock speed G_{core} is another important factor in determining an overall speed gain. It is reasonable to assume that the individual cores in a modern platform will operate at a higher clock speed than the baseline system. The core gain is simply a scaling factor:

$$G_{core} \approx \frac{\text{Core speed in alternative parallel platform}}{\text{Existing core speed} = 1.2\text{GHz}} \tag{B.3}$$

The expression assumes that all cores operate at the same clock speed.

B.5 Scalability of the DCS Design Problem

A plausible hardware platform for running the DCS algorithm may comprise a dual socket, quad-core machine (total 8 cores on one motherboard) with each core running at 2-3GHz and allocated 1-2GB of RAM (Total 8-16GB). Based on core-to-core and socket-to-socket scaling factors of 0.7 and 0.8 and a code parallelisation factor of 0.85 we can use the expression for G_{tot}, the total reduction in processing time where $G_s = 40$ and $G_{core} = 2$:

$$G_{tot} \approx \frac{G_s G_{core}}{\left(\frac{p}{G_H}\right) + (1-p)} \tag{B.4}$$

Equation B.4 assumes that the hardware gain, G_H is equivalent to the effective number of processing cores and may be substituted into Amdhal's law (equation B.1) in place of n_c. For the platform described above:

$$G_H \approx 0.8 \times 2(1 + 0.7(4-1)) = 4.96 \tag{B.5}$$

Substituting this and other values into equation B.4 and noting that all values are approximate, the platform described above would reduce execution time by a factor of:

$$G_{tot} \approx \frac{40 \times 2}{\left(\frac{0.85}{4.96}\right) + (1-0.85)} \approx 248 \tag{B.6}$$

The baseline execution time of 2520 hours (15 weeks) is reduced to around 10 hours. In practice, the change to C/C++ would require substantial effort and would not be feasible. Re-calculating the figures for a parallel MATLAB® implementation yields a speed gain of 6.2 thus reducing the execution time to two-and-a-half weeks. More reasonably, two of the machines described above (a total of 16 cores) could be used as a cluster and the total execution time halved to around 8 days (5 hours if implemented with C++/OpenMP™).

B.6 Hardware Configurations Considered

Various hardware options for parallel implementation were considered but ultimately rejected owing to cost and convenience. The difficulty of providing a, ''bang-for-buck'' business case makes hardware and software acquisition difficult to justify.

The first option was to use a desktop machine with multiple multi-core processors - implementation on MATLAB® would require a licence for the THE MATHWORKS™ parallel processing toolbox. The toolbox initiates a 'worker' for every processor core available (up to 8 cores). Each individual worker runs a separate instance of the MATLAB® engine occupying approximately 250MB of RAM each; this increases the cost of hardware significantly. Parallel implementation using C++ and OpenMP on a single machine would be achievable at lower cost and resource consumption but would require greater time and add significant technical complexity.

A second option was to build a small computing cluster. This would involved the purchase of a number of multi-core computers which could be networked and controlled from a master computer known as the 'scheduler'. Tasks could be uploaded to the scheduler and executed independent of the computer on which the code was written. For the same reasons as described above, this implementation was deemed too expensive for implementation using MATLAB® and both technically risky and expensive using C/C++.

Finally, the loan or use of a third party's computing cluster was considered. This proved impractical as processing time is expensive and few clusters are set up to execute parallel MATLAB® code. Other significant issues included the refusal of Aero Engine Controls to permit remote access to external computing resources from company sites.

Having considered and costed these options, it was agreed that the algorithm would be constrained to a single dual core desktop machine. The practicality of obtaining MATLAB® parallel computing licences meant that execution would be undertaken on a single core. This is a significant impediment to both the production of high quality results and the ease of testing/debugging the algorithm.

B.7 Conclusion

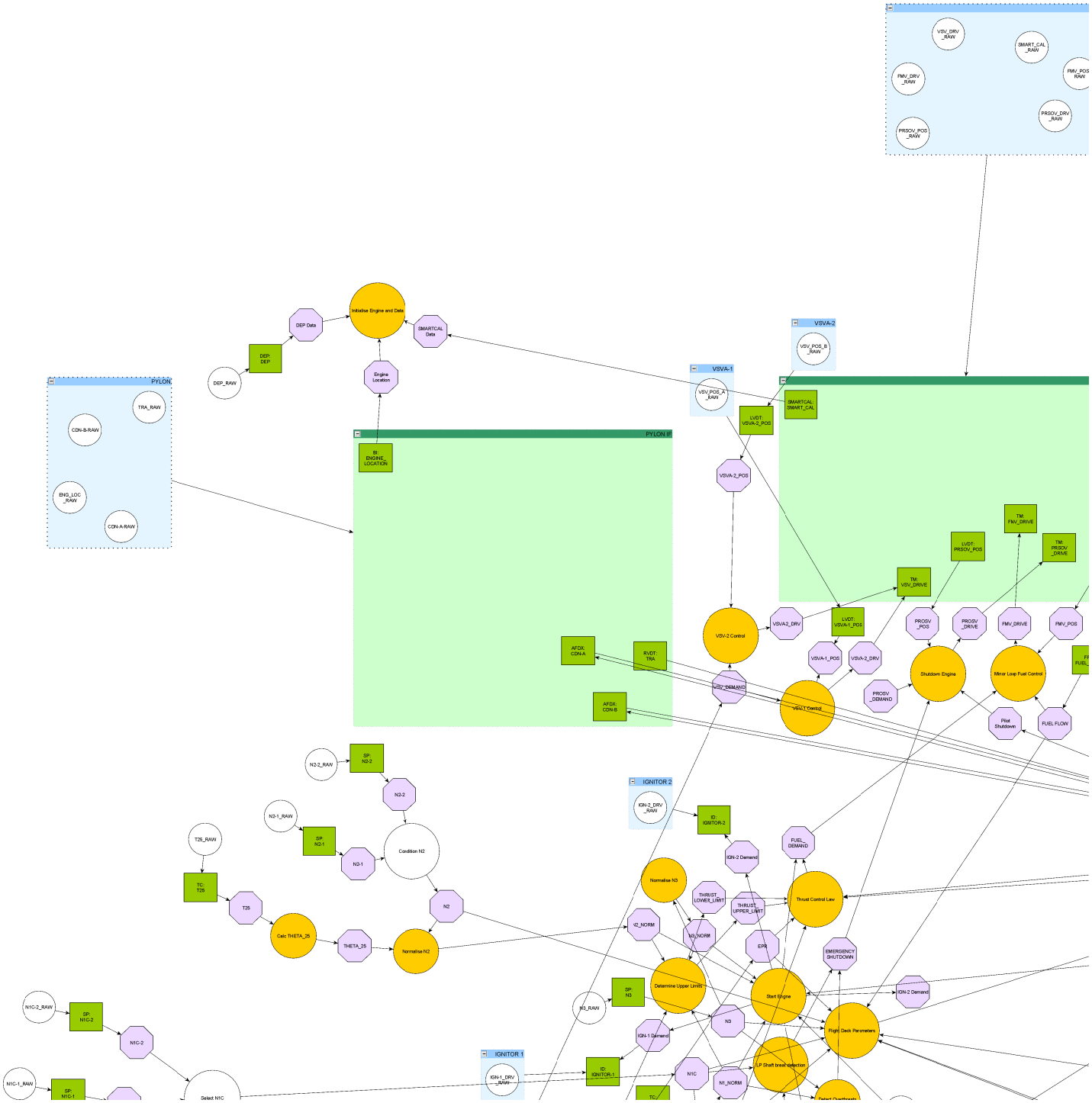
This chapter has shown that despite its complexity, the optimisation scheme does not necessitate super computing resources. If implemented using the hardware and software approaches stated above, the execution time could be reduced to a matter of hours. This makes the approach plausible for the industrial setting for which it was developed.

The values used are approximate and far from scientific. There are may application specific factors which further impact the execution speed. However, the expressions proved usefully accurate when used to compare the MATLAB® and C++ implementations of the algorithm to calculate the harness routing matrices.

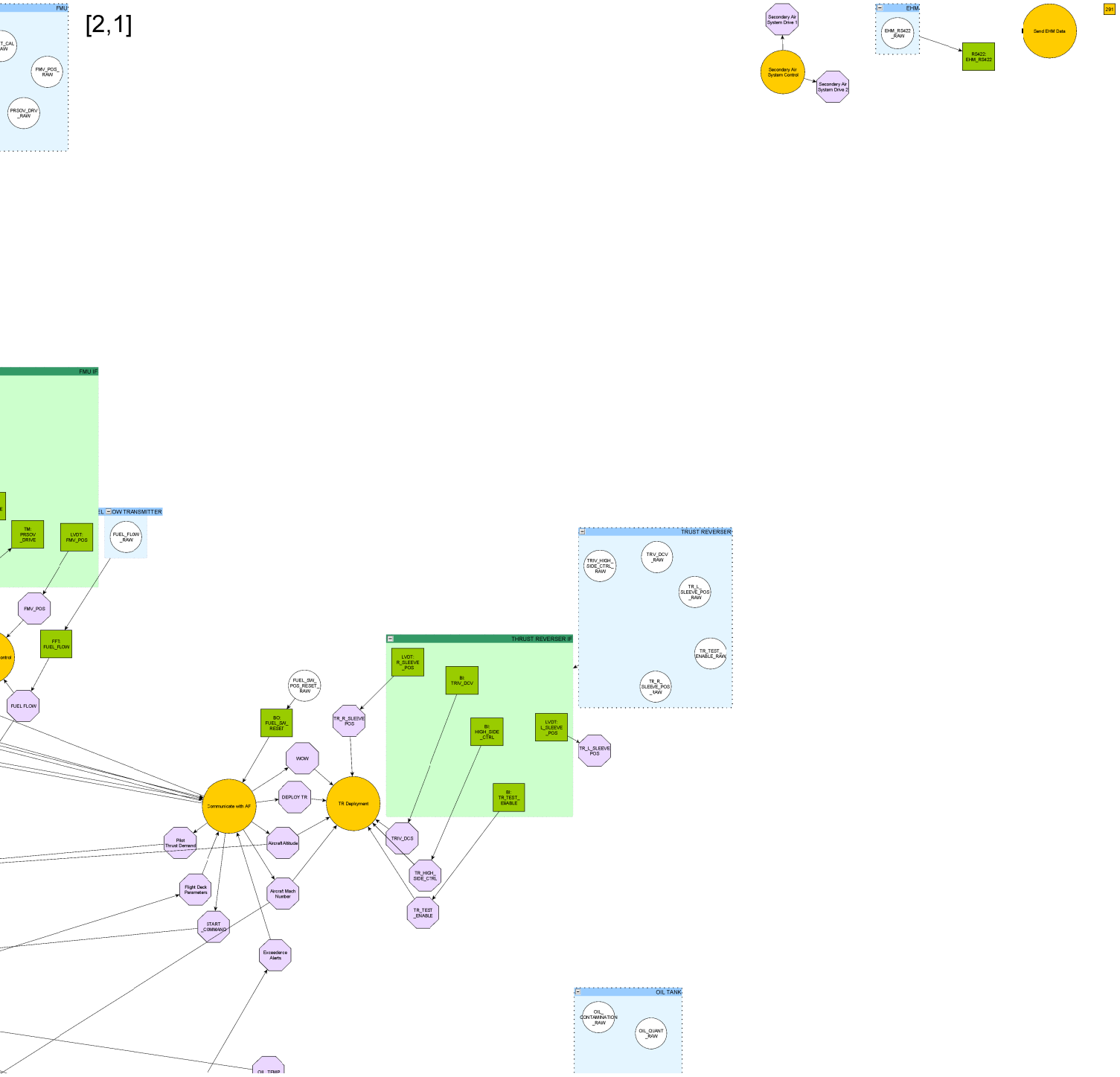
APPENDIX C

Control System Functional Model

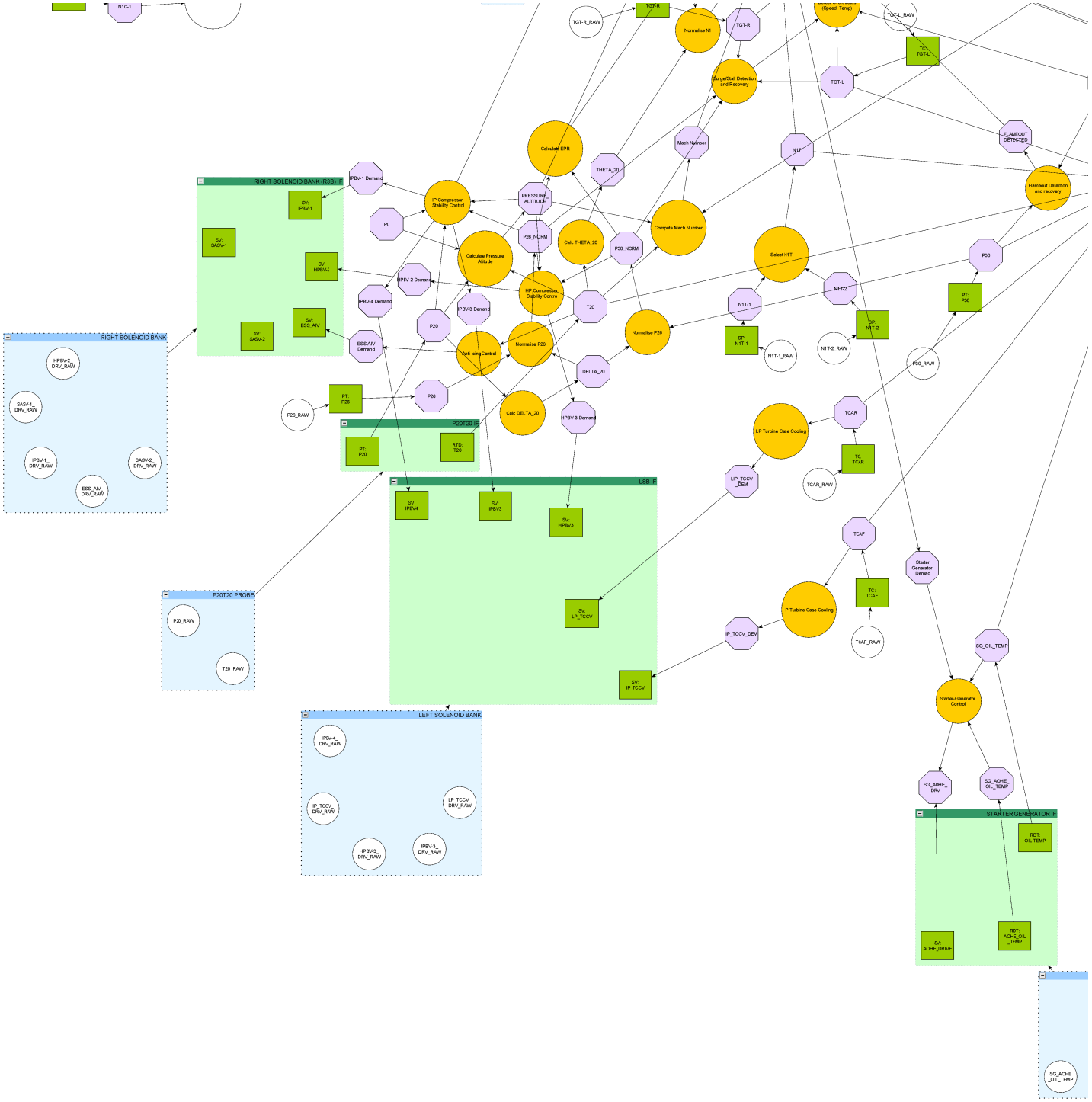
[1,1]

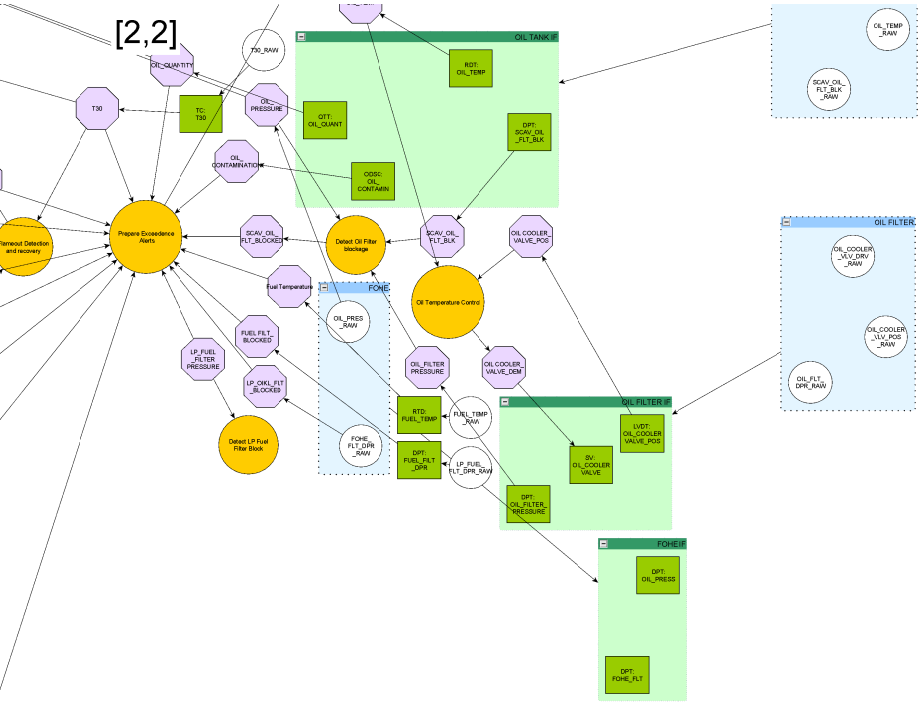

VSV_DRV _RAW
SMART_CAL _RAW
FMV_DRV _RAW
FMV_POS _RAW
PRSOV_DRV _RAW
PRSOV_POS _RAW
Initialise Engine and Data
DEP Data
SMARTCAL Data
DEP: DEP
DEP_RAW
Engine Location
PYLON
TRA_RAW
CDN-B-RAW
ENG_LOC _RAW
CDN-A-RAW
PYLON IF
EI: ENGINE LOCATION
VSVA-1
VSV_POS_A _RAW
LVDT: VSVA-2_POS
VSVA-2_POS
VSVA-2
VSV_POS_B _RAW
SMARTCAL: SMART_CAL
LVDT: PRSOV_POS
TM: FMV_DRIVE
TM: PRSOV _DRIVE
TM: VSV_DRIVE
VSV-2 Control
VSVA2_DRV
LVDT: VSVA-1_POS
PRSOV _POS
PRSOV _DRIVE
FMV_DRIVE
FMV_POS
AFDX: CDN-A
RVDT: TRA
AFDX: CDN-B
VSV_DEMAND
VSVA-1 Control
VSVA-1_POS
VSVA-2_DRV
PRSOV _DEMAND
Shutdown Engine
Minor Loop Fuel Control
Pilot Shutdown
FUEL FLOW
N2-2_RAW
SP: N2-2
N2-2
N2-1_RAW
SP: N2-1
N2-1
Condition N2
T25_RAW
TC: T25
T25
Calc THETA_25
THETA_25
Normalise N2
N2
IGNITOR 2
IGN-2_DRV _RAW
ID: IGNITOR-2
IGN-2 Demand
FUEL_ DEMAND
Normalise N3
THRUST_ LOWER_LIMIT
THRUST_ UPPER_LIMIT
Thrust Control Law
I/O_NORM
I/2_NORM
EPR
EMERGENCY SHUTDOWN
Determine Upper Limits
Start Engine
Flight Deck Parameters
IGN-2 Demand
SP: N3
N3_RAW
N3
IGN-1 Demand
N1C-2_RAW
SP: N1C-2
N1C-2
N1C-1_RAW
SP: N1C-1
Select N1C
IGNITOR 1
IGN-1_DRV _RAW
ID: IGNITOR-1
N1C
N1_NORM
LP Shaft break detection

[2,1]


FMU
FMV_POS_RAW
PRSOV_DRV_RAW
Secondary Air System Drive 1
Secondary Air System Control
Secondary Air System Drive 2
EHM
EHM_RS422_RAW
RS422 EHM_RS422
Send EHM Data
291
FMU IF
TM: PRSOV _DRIVE
LVDT: FMV_POS
FMV_POS
FF1: FUEL_FLOW
FUEL FLOW
FUEL_FLOW _RAW
TRUST REVERSER
TRIV_HIGH_ SIDE_CTRL_ RAW
TRV_DCV _RAW
TR_L _SLEEVE_POS _RAW
TR_TEST_ ENABLE_RAW
TR_R_ SLEEVE_POS _RAW
THRUST REVERSER IF
LVDT: R_SLEEVE _POS
BI: TRIV_DCV
BI: HIGH_SIDE _CTRL
LVDT: L_SLEEVE _POS
TR_L_SLEEVE POS
BI: TR_TEST_ ENABLE
FUEL_SW_ POS_RESET_ RAW
BO: FUEL_SW_ RESET
TR_R_SLEEVE POS
WOW
DEPLOY TR
Communicate with AF
TR Deployment
Pilot Thrust Demand
Aircraft Altitude
Flight Deck Parameters
Aircraft Mach Number
START _COMMAND
Exceedance Alerts
TRIV_DCS
TR_HIGH_ SIDE_CTRL
TR_TEST _ENABLE
OIL TANK
OIL_ CONTAMINATION _RAW
OIL_QUANT _RAW

[1,2]


RIGHT SOLENOID BANK (RSB) IF
RIGHT SOLENOID BANK
P20T20 IF
P20T20 PROBE
LSB IF
LEFT SOLENOID BANK
STARTER GENERATOR IF
Calculate EPR
Calculate Pressure Altitude
Calc THETA_2D
Calc DELTA_2D
Compute Mach Number
Normalise N1
Normalise P26
Normalise P2D
Select NTT
Anti Icing Control
Surge/Stall Detection and Recovery
Flameout Detection and recovery
LP Turbine Case Cooling
P Turbine Case Cooling
Starter-Generator Control
IP Compressor Stability Control
HP Compressor Stability Control
TGT-R
TGT-L
TGT-R_RAW
TGT-L_RAW
TC: TGT-L
TC: TCAR
TC: TCAF
SP: NTT-1
SP: NTT-2
PT: P30
PT: P26
PT: P20
RTD: T20
RTD: OIL TEMP
RTD: ACHE_OIL _TEMP
SV: IPBV-1
SV: SASV-1
SV: HPBV-2
SV: ESS_AIV
SV: SASV-2
SV: IPBV4
SV: IPBV3
SV: HPBV3
SV: LP_TCCV
SV: IP_TCCV
SV: ACHE_DRIVE
IPBV-1 Demand
IPBV-2 Demand
IPBV-3 Demand
IPBV-4 Demand
ESS AIV Demand
Starter Generator Demad
HPBV-2_ DRV_RAW
SASV-1_ DRV_RAW
IPBV-1_ DRV_RAW
SASV-2_ DRV_RAW
ESS_AIV_ DRV_RAW
P20_RAW
T20_RAW
IPBV-4_ DRV_RAW
IP_TCCV_ DRV_RAW
LP_TCCV_ DRV_RAW
HPBV-3_ DRV_RAW
IPBV-3_ DRV_RAW
P26_RAW
N1T-1_RAW
N1T-2_RAW
P50_RAW
TCAR_RAW
TCAF_RAW
LP_TCCV _DEM
IP_TCCV_DEM
SG_OIL_TEMP
SG_ACHE_ DRV
SG_ACHE_ OIL_TEMP
SG_ACHE _OIL_TEMP
P0
P20
P26
P30
P30_NORM
P2B_NORM
T20
THETA_2D
PRESSURE_ ALTITUDE
Mach Number
N1T
N1T-1
N1T-2
TCAR
TCAF
FLAMEOUT DETECTED


[2,2]
OIL_TANK IF
OIL_TEMP_RAW
SCAV_OIL_FLT_BLK_RAW
T30_RAW
OIL_QUANTITY
T30
TC: T30
OIL_PRESSURE
OTT: OIL_QUANT
RDT: OIL_TEMP
DPT: SCAV_OIL_FLT_BLK
OIL CONTAMINATION
ODSC: OIL CONTAMIN
Prepare Exceedence Alerts
SCAV_OIL_FLT_BLOCKED
Detect Oil Filter blockage
SCAV_OIL_FLT_BLK
OIL COOLER_ VALVE_POS
OIL FILTER
OIL_COOLER_VLV_DRV_RAW
OIL_COOLER_VLV_POS_RAW
OIL_FLT_DPR_RAW
Fuel Temperature
FOHE
Oil Temperature Control
OIL_PRES_RAW
FUEL FILT_ BLOCKED
LP_FUEL _FILTER PRESSURE
LP_OIL_FLT _BLOCKED
OIL_FILTER PRESSURE
OIL COOLER_ VALVE_DEM
OIL FILTER IF
RTD: FUEL_TEMP
FUEL_TEMP_RAW
DPT: FUEL_FLT _DPR
LP_FUEL_ FLT_DPR_RAW
FOHE_ FLT_DPR _RAW
LVDT: OIL_COOLER VALVE_POS
SV: OIL_COOLER VALVE
DPT: OIL_FILTER_ PRESSURE
Detect LP Fuel Filter Block
FOHE IF
DPT: OIL_PRESS
DPT: FOHE_FLT
RATOR IF
RDT: L_TEMP
STARTER-GENERATOR
SG_AOHE_ DRV_RAW
SG_OIL_ TEMP_RAW
SG_AOHE _OIL_TEMP