

Selecting a Topology for Safety-Critical Real-Time Control Systems

Mark Nicholson

Submitted for the degree of Doctor of Philosophy

Department of Computer Science
University of York

September 1998

Abstract

In recent years the functionality required of computer based control systems for safety-critical real-time applications has increased dramatically. Inevitably this has led to an explosion in the complexity of such systems and an understanding, in both academia and industry, that existing design methods are no longer adequate. One design issue that has traditionally been addressed in an *ad hoc* and rather simplistic manner is that of setting the topology of a distributed computer based control system.

A topology consists of a configured set of hardware and software units employed to fulfil a set of logical control actions. A topology may employ multiple, possibly diverse, copies of these units to ensure that dependability, timing and functional requirements are met. A designer aims to determine the set of units to be employed and how they should be configured. A maintainer aims to discover the effect of a change in functionality, or the units employed, on the effectiveness of an existing topology.

Potentially there are a large number of alternative feasible topologies. Unfortunately, existing techniques rely on past experience and typically set a topology very early in the design process. At best only a fraction of the admissible topologies are considered and almost no allowance is made for the effect of subsequent design decisions. This raises the spectre of costly reworking of proposed designs in order to provide the required system characteristics within the given topological framework.

This thesis shows that it is possible to resolve the topology selection problem with the aid of quantitative evaluation functions and implicit enumerative search techniques. Automated tool support is restricted to systems that employ bus based architectures.

Dependability and timing attributes of a topology are investigated at appropriate points in the design process. That is, dependability issues are addressed at the architectural level and timing issues at the configuration level. Results of previous applications of the approach are used to determine the set of admissible topologies. Overall, the approach allows a user to consider a much larger set of topologies than existing methods.

It is shown that adaptive search techniques may be used to trawl the set of alternative topologies. Quantitative assessment and prediction techniques are employed to evaluate the quality of a proposed topology. The approach is hierarchical, iterative within and between different stages of the design process, and addresses appropriate characteristics of safety-critical real-time control systems. The proposed approach is evaluated using two tools described in this thesis, X-Topmeter and X-Alloc.

Contents

1	Overview	1
1.1	Safety-Critical Real-Time Control Systems	3
1.2	Supporting SC-RT System Design	5
1.3	The Topology Selection Design Issue	9
1.4	Motivation for Approach Employed	16
1.5	Structure of Thesis	19
2	The Elements of a Topology	21
2.1	Formal Definition of the GTP	22
2.2	Topological Units and Resources	24
2.3	Designing for Dependability and Fault Tolerance	28
2.4	Analysing the Quality of a Dependable System	39
2.5	Designing Hard Real-Time Systems	51
2.6	Analysing the Quality of an HRT System	53
2.7	Search Techniques for Combinatorial Optimisation	57
2.8	Genetic Algorithms	62
2.9	Search Space Evolution via Tabu Search	68
2.10	Simulated Annealing	69
2.11	Discussion	72
3	The Topology Problem at the Architectural Level	75
3.1	The Architectural Framework	76
3.2	The Architectural Topology Problem	79
3.3	Resolving the Architectural Topology Problem	92

3.4	X-Topmeter Illustrative Examples	104
3.5	Discussion	131
4	The Topology Problem at the Configuration Level	135
4.1	The Allocation Problem	137
4.2	Resolving the Allocation Problem	145
4.3	X-Alloc	150
4.4	Illustrative Examples	153
4.5	Discussion	166
5	Case Study and Evaluation of Approach	169
5.1	Case Study: Computer Assisted Braking System	169
5.2	CABV1: Integrated Architecture with 3 Networks	174
5.3	CABV2: Integrated Architecture with IMA Network	184
5.4	Interaction Between Topology Selection and System Design	189
5.5	Evaluation of the Approach	192
5.6	Summary	205
6	Conclusions	207
6.1	Summary of Thesis	207
6.2	Comparison with Related Work	209
6.3	Future Work	214
6.4	Final Comments	217
	References	219

List of Tables

1.1	Design and Topology Issues	7
2.1	Failure Rate Claims for Software Units	27
3.1	Task Attributes	86
3.2	Data Structures	98
3.3	Search Space Data	99
3.4	X-TopmeterV1.5 Parameter Set	101
3.5	Cost of N-Version Software	103
3.6	Hardware Resources for Example 1	105
3.7	Software Resources for Example 1	105
3.8	Task Data for Example 1	106
3.9	Service and Network Data for Example 1	106
3.10	Results for Best Experiment for One Service Example	107
3.11	Library Data for Services and Tasks	109
3.12	Library Data for Resources	110
3.13	MTTF for Task 1	111
3.14	Results for 2 Service Example	111
3.15	Sensitivity Analysis	113
3.16	Removal Experiments	114
3.17	Admissible Resources for Example 3	115
3.18	Overall Topology Results for Default Weightings	120
3.19	Restricted Resource Set	122
3.20	Performance Data for Alternate Resource Set	123
3.21	Performance Data for $k_{rel}=20$	125

3.22	Performance Data for $k_{size} = 200$	126
3.23	Performance Data for $k_{cost}=1$	126
3.24	Performance Data, Redundancy Required	128
3.25	Performance Data, Sensor Topology Equals Actuator Topology	129
3.26	Coverage Factor Tuples	130
3.27	Performance Data, Coverage Factors	131
4.1	X-Alloc Parameter Set	152
4.2	Results for Example 1	156
4.3	Results for Example 2	159
4.4	SetP for Example 3	163
4.5	Results for Example 3	164
5.1	Library of Admissible Resources	174
5.2	CABV1: Sub-tasks	176
5.3	Resource set for CABV1	178
5.4	Results for Typical CABV1 Free-form Experiment	181
5.5	Results for Typical CABV1 Redundant Experiment	182
5.6	Results for Typical CABV1 Free-form Allocation Experiment	183
5.7	Results for Typical CABV2 Experiment	187
5.8	Results for Typical CABV2 Allocation Experiment	188

List of Figures

1.1	A Closed Loop Control System	5
1.2	Simple V-model of the Design Process	6
1.3	Hierarchical Model of a Topology	10
1.4	A Design (Artefact) Space	14
2.1	Software Failures	27
2.2	Laprie's Dependability Terminology	31
2.3	Parallel Hardware Redundancy	34
2.4	Acceptance Hardware Redundancy	35
2.5	N-Modular Redundancy	35
2.6	TMR Hardware Redundancy	35
2.7	Twin TMR Hardware Redundancy	36
2.8	Cold Spare Hardware Redundancy	36
2.9	TMR plus Spare Hardware Redundancy	39
2.10	Simplex Markovian Model	41
2.11	Hierarchy of Reliability Models	43
2.12	Unified Simplex Model	46
2.13	Unified Model for a TMR Hardware/Software System	46
2.14	SDAT Dependability Model Object Types	49
2.15	Periodic Transaction	52
2.16	Search Techniques for Optimisation	58
2.17	General Search Paradigm	59
2.18	SA Minimisation Algorithm	70
3.1	The GUARDS Generic Architecture	76

3.2	An Integrated Modular Architecture	78
3.3	Logical Architectural for a Single Service	82
3.4	Three-task and Three-network Service	84
3.5	Three-task and One-network Service	84
3.6	Sample Topologies	84
3.7	Bit String for a Task and a Network	100
3.8	Three Task Model	104
3.9	X-Topmeter Versus Bruteforce	108
3.10	Services for Illustrative Example 1	109
3.11	Best RBD Formulation	112
3.12	Logical Architecture for Example 3	114
3.13	MTTF Target of 3000 Units	116
3.14	MTTF Target of 5000 Units	117
3.15	MTTF Target of 7000 Units	118
3.16	MTTF Target of 9000 Units	118
3.17	MTTF Target of 11000 Units	119
3.18	Task Topologies for Service 0	121
3.19	Results for Alternate Resource Set	122
3.20	Results for $k_{rel} = 20$	124
3.21	Results for $k_{size}=200$	125
3.22	Results for $k_{cost}=1$	126
3.23	Results for Redundancy Required Alternative	127
3.24	Results for Sensing Equals Action	129
3.25	Results for Coverage Factors	131
4.1	An Allocation Problem	138
4.2	Processing Sub-tasks	139
4.3	Allocation Problem Representation	140
4.4	Moves for the Allocation Problem	143
4.5	Search Techniques for The Allocation Problem	145
4.6	Transactions for a TMR Task	151

4.7	Architectural Topology for Illustrative Example 1	155
4.8	Solution to Illustrative Example 1	158
4.9	Architectural Topology for Illustrative Example 2	159
4.10	Solution to Illustrative Example 2	161
4.11	Architectural Topology for Example 3	162
4.12	Solution to Illustrative Example 3	164
4.13	Platform For Example 3	165
5.1	CAB System Context Diagram	171
5.2	Integrated CAB System	175
5.3	CABV1: System Description	176
5.4	CABV1: Logical Architecture of Processing Tasks	176
5.5	CABV1: Free-form Architectural Topology	180
5.6	CABV1: Redundant Architectural Topology	181
5.7	System-DAG of CABV1 for Free-form Architectural Topology	182
5.8	Results of CABV1 Free-form SA	183
5.9	CABV2: Logical Architecture	184
5.10	CABV2: System Description	185
5.11	CABV2: Architectural Topology	186
5.12	DAG of CABV2 Processing Topology	187
5.13	Results of CABV2 SA	188
5.14	Interactions between Topology Selection & Design	190
5.15	CABV3: System Description	199
5.16	CABV3: Logical Architecture	199
5.17	Structure of an Electronic Unit	201
5.18	FTU Configurations: (a)2 units (b)TMR units (c)Single unit (d) 4 units	201

Notation

All Topology Formulations

x	a proposed set of units and / or resource assignments
u	unit: smallest indivisible item such as processor, bus, or sensor
n	network
r	resource: type of unit that may be selected for a topology
s	logical control action to be provided by the system
$cost_u$	ownership cost of unit u
$WCRT_s$	predicted worst case response time of a service
$\max WCRT_s$	maximum permissible WCRT of service s
$depend_s$	measure of the dependability of service s
$\text{Min } depend_s$	minimum level of dependability of services required
num_u	number of unit u employed in the system, 0 if non employed
$\max num_{u\bar{u}}$	maximum number of unit u employable on unit \bar{u}
$num_{u\bar{u}}$	number of unit u to be implemented on unit \bar{u}
$parallel_u$	number of units actually placed in parallel with u
$\text{Reqd } parallel_u$	number of units required to be placed in parallel with u

GTP as ATP

$F(x)$	quality of proposed resource assignment
W_u	penalty for using unit u
γ_u	penalty weighting factor for cost of unit u

GTP as SAT

cl	clause
$F(x)$	number of unsatisfied clauses for assignment x
$Clause_{cl}$	0 if clause cl satisfied, 1 otherwise (A clause indicates whether a specified constraint has been met.)
$\max \text{ cost}$	maximum cost permitted for the system
ϵ_u	penalty weighting factor for failure to meet clause cl
num_u	number of units used in the topology
$\max num_u$	maximum number of units in the topology
$\max num_u$	max allowable number of unit u in the topology

Architectural Topology Problem as ATP

$F(\mathbf{x})$	objective function indicating quality of proposed resource assignment
x_{ct}	1 if task t employs architectural component c , 0 otherwise (A 1 indicates an assignment)
x_{cn}	1 if network n employs architectural component c , 0 otherwise (A 1 indicates an assignment)
$C_t \subseteq \{1, 2, \dots, C\}$	set of admissible components for task t ($\{t=1, \dots, T\}$)
$C_n \subseteq \{1, 2, \dots, C\}$	set of admissible components for network n ($\{n=1, \dots, N\}$)
$R_{(ct)}$	set of admissible resources for combination ct
$R_{(cn)}$	set of admissible resources for combination cn
$mttf_s$	predicted mean time to failure for service s ($\{s=1, \dots, S\}$)
$\min mttf_s$	minimum acceptable mean time to failure for service s

Acronyms and Terms

ABS	Anti-lock Braking System
Allele	Value of a particular element in a chromosome
Architectural Topology	Resources employed to provide <i>dependable</i> services
ASE	Analysis-Synthesis-Evaluation
ATP	Assignment Type Problem
Availability	Probability that a system is working at time t , regardless of the number of times it may have failed and been repaired in the interval $(0,t)$
CAB	Computer Assisted Braking system
CAN	Controller Area Network
Change move	Value of a single element in a gene structure is changed
Chromosome	Coded version of a solution employed in a GA
Component	Group of units employed to provide a single function
Complex resource	Component
Congruent data	Single message broadcast to multiple locations
COI	Controlled Object Interface
Create function	Applies genetic operators to the mating pool, Q , to generate a new set of chromosomes
Crossover	GA intensification operator
DAG	Directed Acyclic Graph
Deadline	Maximum time allowed from the expected sub-task release until execution is completed
Deception	Elements that are not in the globally optimal solution increase in frequency faster than those that are
Dependability	Trustworthiness of a computer system such that reliance can justifiably be placed on the services it provides
Design space	Set of design solutions (artefacts) that may be postulated for a particular design issue
Emergent property	A system property that is resolved by revisiting a design issue throughout the design process
End-to-End delay	Amount of time before a control action is undertaken from a demand for the action.
EP	Exponential Polynomial
Epistasis	Change in a gene causes a change in fitness that varies in sign and magnitude, depending on the values of other genes
ES	Evolutionary Strategy

Fault Tolerance	Ability of a system to continue to perform its tasks after the occurrence of a fault
FMEA	Failure Modes and Effects Analysis
FTU	Fault-Tolerant Unit
GA	Genetic Algorithm
Gene	Single element in a chromosome. May be binary or integer in value.
Genetic drift	A gene becomes predominant in a population and is eventually copied into every member of that population
GTP	General Topology Problem
GUI	Graphical User Interface
IMA	Integrated Modular Avionics
Integrity	Likelihood that a safety related system will satisfactorily perform required safety functions under all stated conditions within a stated period of time
Jitter	Variation of sub-task completion from precise periodicity
kofn	k out of n system
Logical architecture	Embodies commitments that can be made independently of the constraints imposed by the execution environment
MEDL	Message descriptor list
Merge function	Combines old population, P, mating pool, Q, and new population, R
Mutation	GA diversification operator
Neighbourhood	Set of solutions, S' , that can be reached from s by a simple operator, σ
NP	Nondeterministic polynomial: class of all problems that can be solved in polynomial time by a nondeterministic algorithm
NP-complete	Class of decision problems that have the best chances of being intractable problems in NP
NP-hard	Class of optimisation problems in NP
Periodic	Sub-task or message released at regular intervals by a timer event
Physical architecture	Takes the constraints embodied in the logical architecture and embraces non functional requirements
Peopleware	Users / operators of a system
Phenotype	Decoded version of the chromosome that is tested to determine the quality of the proposed topology
Platform	Set of hardware units used to implement a system
Primitive resource	A unit

Procurement	Decision to employ particular hardware and software resources
Reliability	Ability of a component or system to function correctly over a specified period of time
Resource	Type of unit
SA	Simulated Annealing
SAT	Satisfiability problem
SC-RT	Safety-Critical Real-Time
SDF	System Design Factor
Select function	Determines number of “fit” solutions which form the “mating pool”, Q
Shadow unit	Shared cold spares in a TTA system
Sporadic	Sub-task released by an originating event from either another sub-task or the environment of a processing task
Sub-task	Active process
Swap move	Transposes the values of two elements of a gene structure
Transaction	Sequence of precedence ordered sub-tasks that form paths through a system
TS	Tabu Search
TSP	Travelling Salesman Problem
TTA	Time Triggered Architecture
TTP/C	Time-Triggered Protocol
Unit	An indivisible item, such as a processor.
V-model	Design life cycle model
WCET	Worst-Case Execution Time
WCRT	Worst-Case Response Time

Acknowledgements

I would like to thank a multitude of people for their help and patience during the development and writing of this thesis. In particular, I would like to thank my supervisor Alan Burns for his guidance throughout the last six years. I would also like to thank my assessor Alan Frisch for his comments and suggestions during the internal assessment and review process.

I wish to acknowledge the EPSRC for providing financial support for the DESSERTS project, from which much of the material in this D.Phil emerged.

I wish to thank David Pumfrey whose collaboration on the Computer Assisted Breaking study in all its many disguises was invaluable. I would also like to thank Prof. John McDermid for his contributions to the Design Synthesis for Safety Engineered Real-Time Systems (DESSERTS) project, this thesis and continued support during the ‘writing-up’ period, and beyond.

I am thankful for the research environment I find myself in at the Department of Computer Science. I am especially thankful to all my colleagues in the HISE and RTS groups. Special thanks to John Clark, Tim Kelly, Stuart Mitchell, Divya Prasad and Andy Vickers.

Finally, and definately not least, I would like to thank my family and friends who have put up with my long absenses and distracted manner over the last year.

Declaration

I declare that this thesis is original work undertaken by me between the dates of registration for the Degree of Philosophy at the University of York, October 1994 to December 1997. During this period I was employed as a Research Associate and Research Student in the Real-Time Systems Group.

Chapter 2 is partially based on my qualifying dissertation entitled *Optimisation Searches Inspired by Nature* and on references [118, 119].

Chapter 3 contains material that has been published in [120, 121, 116, 115].

Chapter 4 is partly based on material published in [117].

The case study in Chapter 5 is presented and discussed in more detail in [106, 122, 133].

Chapter 1

Overview

“Knowledge is the enemy. Evolutionary algorithms can come up with systems that challenge our orthodoxies, defy our understanding, and still work” ¹

A key element in any control based safety-critical computer system is the hardware / software architecture as represented by a topology of configured resources. Thus a topology consists of a set of resource items (software, processors, buses, sensors and actuators) employed to fulfil a given set of control actions. A topology selection process aims to choose the set of software and hardware items and configure them so that a given set of goals can be met and constraints satisfied.

The functional attributes and system topology of a distributed control system should co-evolve, that is they should both emerge as design progresses. Unfortunately, the process by which an ‘optimal’ topology for a distributed Safety-Critical Real-Time (SC-RT) control system may be selected is poorly understood. Topologies are currently fixed very early in the design process, but only evaluated late in the process. Furthermore, it is not clear, nor is analysis carried out to determine, how the effectiveness of a given topology is affected by, for instance, a change in hardware procurement policies or a mid-life update of the control system.

Selecting a poor topology may have a number of effects on the implemented control system. The size and cost of the proposed hardware platform may be greater than necessary to provide an appropriate level of dependability and ensure that timing requirements are met. Alternatively, the proposed platform may be inadequate to provide the necessary dependability and timing attributes. The latter is more common in industrial practice. Maintenance of the system during its operational life may be more difficult and more costly than for a ‘good’ topology.

¹Comment by M.Kelly in Evonews Issue 3, March 1997

The primary side-effect of selecting a poor topology however is likely to be an increase in the complexity of other aspects of the system. For instance, a poor topology may impose severe restrictions on the Worst Case Execution Times (WCET) of particular software modules leading to a heavy expenditure of time and money to produce very fast code. The author has even seen code rewritten in assembler to overcome this problem and in the process circumventing the stated design process. Functional complexity may also be increased to provide extra protection against perceived deficiencies in the selected topology. It may have been more effective, and cheaper, to employ an alternative topology. This option is effectively denied the designer if the quality of a proposed topology is only evaluated late in the design process.

In this thesis a first attempt at producing an approach to help a design team select an ‘optimal’ topology for a distributed SC-RT control system is presented. In fact, in many cases a ‘sub-optimal’ feasible solution is considered to be ‘good enough’. Furthermore, the approach may be used to investigate the effect of a change in procurement policy or the evolution of a system through its operational life.

The approach treats a topology as an emergent property of a control system. An emergent property is a system property that can only be resolved by revisiting a design issue throughout the design process. Analysis of the effectiveness of a proposed topology is undertaken from an early stage in the design process. This allows a new topology to be selected at the cost of relatively little reworking of the design.

The topology selection process is split into two sub-problems, the architectural topology and allocation problems, that are addressed at different stages in the system design process. The chosen approach is hierarchical and iterative both within and between the sub-problems. Furthermore, it is updateable and upgradeable both within and between projects. The aim is to make the approach as “plug and play” as possible.

Automatic tool support is provided to aid the resolution of the topology selection problem in the form of guided searches. The architectural topology support tool, *X-Topmeter*, employs a Genetic Algorithm (GA). The allocation support tool, *X-Alloc*, employs a Simulated Annealing (SA) algorithm. These tools are adapted and re-run on a number of occasions during the topology selection and system design process. The user is required to intervene throughout the emergence process. This allows the impact of other design issues, interpretation of the results of an experiment, modifications to the search parameters and modifications to the system design to be incorporated into the topology selection process.

In the remainder of this chapter the background to the topology selection problem is introduced. Thus in Section 1.1 a set of issues that need to be addressed during the design process are introduced. In Section 1.2 the point in the design process at which a variety of issues can be addressed is investigated. In Section 1.3 an exposition of the

topology selection issue is presented. Finally, in Sections 1.4 and 1.5 the motivation for, and structure of, this thesis is given.

In short, the nature of this thesis is to review the elements of a topology, produce automatic software based tools to aid resolution of two topology selection sub-problems, evaluate the merits of the approach and present a number of conclusions. Appendix A shows the set of reliability models and data structures employed by the tools.

1.1 Safety-Critical Real-Time Control Systems

Software controlled systems are becoming increasingly common in applications that can cause *catastrophic* events, leading to either loss of life or significant economic harm. These systems often have timing, as well as safety, attributes. The applications of interest in this thesis are typically distributed among a number of hardware units. A unit is an indivisible item, such as a processor. SC-RT systems include flight control systems, nuclear reactor trip systems, satellite systems, and increasingly automotive applications.

The *risk* associated with deploying distributed SC-RT control systems is high. As a result, these systems are subjected to particularly stringent requirements. The challenge for the research community is to develop methods that can produce systems that meet these requirements.

Designers have to ask themselves a number of questions early in the design process; such as [159]:

1. How many local control loops are required?
2. What is the maximum possible sampling rate?
3. What are the high speed/high accuracy requirements for closed control loops?
4. What type of sensors and actuators are to be used?
5. How many variables are to be sensed?
6. If groups of sensors or actuators are situated remotely should data concentrators be used?
7. What degree of local autonomy is appropriate?
8. What degree of self-diagnosis, redundancy, or self-repair capability is required?
9. What degree of standardisation/modularisation of resources is required?
10. What is the operational environment?

11. What sequencing needs are present?
12. What safety features are needed?
13. What level of reliability / availability is required?

Answers to these questions form the context of the system development process. Other questions will also need to be addressed by the designer relating to the future needs of the system including [159]:

1. How can the system be extended in the future?
2. How can new actuators/sensors be interfaced?
3. How can the system be made flexible to allow modification?

These questions are inter-related in that the resolution of one issue will have an impact on plausible solutions to other questions. During the design process the postulated solutions to these questions are often taken as givens. A number of these issues however need to be addressed as emergent properties and therefore flexible support for their resolution is required.

The set of admissible answers to these questions form the *design freedoms* available to a designer of the system that is to provide the control services. For some questions it is clear what the alternatives are. For others, the set of alternative solutions may not be obvious or may conflict with the resolution of other questions. In some cases a set of alternatives can be automatically generated.

Evaluation of solutions to the design questions requires a classification of the characteristics that a control system provides. These characteristics are not *independent* and therefore trade-off based decision criteria are required. The primary characteristics of a SC-RT system are [120]:

- (i) **Functionality:** What actions the system must provide
- (ii) **Cost:** Cost of production and ownership of the system
- (iii) **Timing:** At least some of the functions of the service must be performed within specific timing constraints
- (iv) **Dependability:** Trustworthiness of a computer system such that reliance can justifiably be placed on the services it provides [92]
- (v) **Resource usage:** Hardware and software constraints placed on the system.

Control engineers often model closed-loop control systems using four tasks: the *process* (residing on a multiprocessor network), the *control component*, *sensors* and *actuators*, see Figure 1.1 [95]. The control component employed and the peopleware resources (operators) required are not considered in detail in this work. However, it can be shown that control [2] and peopleware [67] issues fit into the approach presented in this thesis.

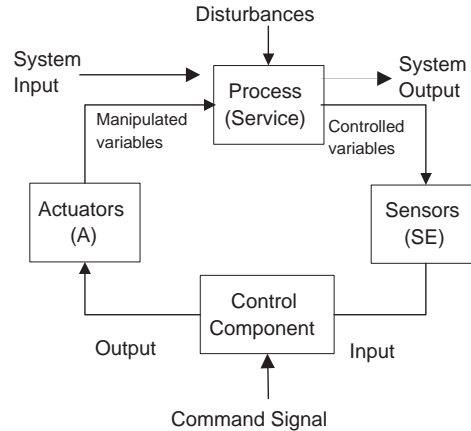


Figure 1.1: A Closed Loop Control System

Functionality and Cost provide the *goals* for the designers/maintainers of a system and therefore need to be considered at each level of the design process. *Constraints* on the choices available to a designer of a particular application are likely to be domain specific, but will inevitably include constraints on resource usage, dependability and timing. A set of *guidelines* for the production of the functionality of a system may be available. International consensus based standards [149] can be used as a general production framework. Specific standards, such as the ISO standard for CAN [76] may also be applied.

It is the contention of this thesis that architectural analysis can be employed inexpensively to reduce costs. Analysis can reveal the implications of design decisions. For instance, analysis can reveal the effect of a decision to employ extra functional units to improve fault-tolerance on the timing characteristics of a proposed system. For maximum cost savings architectural analysis should be first undertaken as early as possible in the design process [82].

1.2 Supporting SC-RT System Design

The analysis framework proposed in this thesis is deliberately closely linked to the design process. Evidence can be accrued during the development process and used to

make design decisions. A number of life cycle models can be accommodated within an emergence process. Two of the most popular, and simplest, are the *cascade model* and the *V-model*. As an example of the way in which a parallel analysis and design framework can be applied, consider the V-model shown in Figure 1.2. For clarity a number of design issues that form part of the emergence process have been omitted from Figure 1.2.

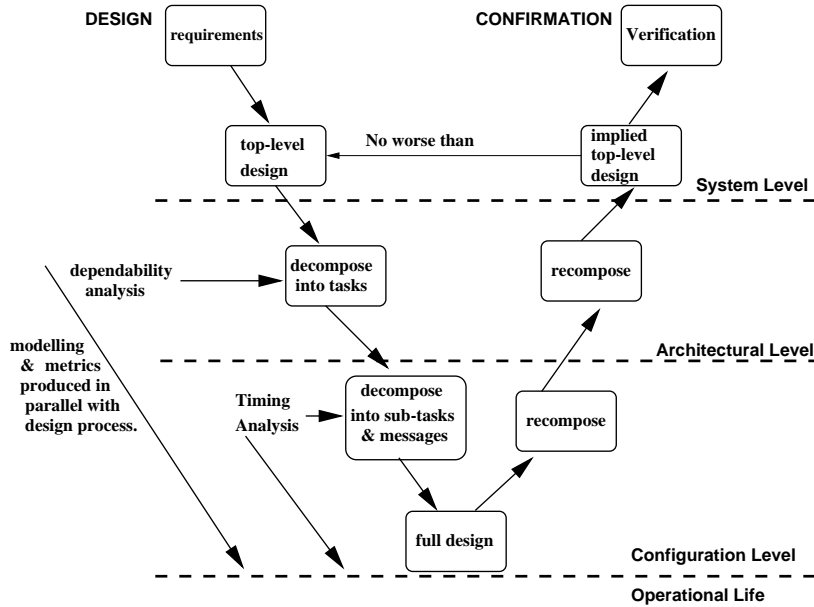


Figure 1.2: Simple V-model of the Design Process

On the left hand side of the V design activities are undertaken starting with the production of a set of requirements and a top-level design, including initial procurement decisions. The design may now be evaluated to determine some of its functional and non-functional attributes. This information is used as part of a decomposition into a more detailed design, which is then evaluated. This process continues until a full design emerges and the system is implemented. The point at which different attributes of a proposed design can be evaluated is also presented in Figure 1.2.

So, predictions are made during the design phase and are used to guide the design process. Once a full design has been produced a confirmatory process is applied, including verification and validation. On the right hand side of the V model shown in Figure 1.2 confirmatory analysis is undertaken to provide assurance that the characteristics of the system during its operational life will be *no worse than* that predicted during the design process. For the safety aspects of a design for instance, this implies that there are actually fewer, or less severe, output deviations, and more input deviations mitigated, than predicted.

This is a very simplistic model of the design process. However it contains the essential

interactions between modelling, metrics and the design process. Four distinct phases of the process are identified in Figure 1.2:

- (i) System level
- (ii) Architectural level
- (iii) Configuration level
- (iv) Operational life

Decisions taken at each of these levels have an impact on the validity of a selected topology. In Table 1.1 an overview of relevant issues at each level is presented.

Level	Design Issues	Topology Issues
System	IT requirements for the artefact (vehicle, plant, etc)	Setting of generic architecture and initial procurement decisions
Architectural	Decomposition into logical architecture. Setting of derived dependability, Worst Case Response Time (WCRT), functional and cost requirements	Selection of architectural component to be employed for each logical task in the system. Choice of resources to be employed in the system.
Configuration	Implementation of the software units and hardware platform	Assignment of software units to hardware units to ensure derived requirements are met. Software can be coded or logical units.
Operational Life	Evolution of the system (mid-life updates, maintenance)	Effect of design decisions on validity of existing topology

Table 1.1: Design and Topology Issues

(i) System Level Decisions

A requirements production process is undertaken at this level, before the nature of the control services to be provided is clearly defined. The set of possible alternatives is vast and the adoption of some requirements may preclude the attainment of others. Decisions are typically judgemental and place obligations on the computing system and therefore impact on the alternatives available at later stages in the design process.

Procurement issues are also addressed at this level. The procurement process determines the set of *off-the-shelf* resources (hardware, software and ‘peopleware’ units) that a designer may employ in a platform. An initial set of procurement decisions will usually be based on previous experience. The set of available resources may change as design progresses.

Suppose a firm wishes to produce and market a new vehicle, vehicle X. The firm must

determine the type of vehicle they wish to produce, is it to be a family saloon or maybe a four wheel drive vehicle for off-road use. They must determine its capacity in terms of the number of people and the amount of luggage it can carry. Of course these decisions are constrained by the potential cost to build X, against the price they believe it can command.

One important decision is how much information technology should be used in X. For instance, the next generation of vehicles could implement a whole range of features including ultrasound sensors for use during parking, wheel sensor systems to provide information for ABS, angle sensors for headlight throw control, pollutant sensors, rain sensors and sun sensors.

Decisions about which computer based facilities to incorporate are made on technical, safety and commercial grounds. The generic architecture to be used to support these facilities may also be determined at this level. A generic architecture incorporates a set of rules for configuring various instances of an architectural topology for different end-user requirements. In this thesis a bus based generic architecture, such as GUARDS (see Section 3.1), is assumed.

(ii) Architectural Level Decisions

A topology consists of a hardware platform and a set of software items allocated to it. The topology provides the required IT services. The basic elements and configuration of this platform is determined at the architectural level by the choice of an architectural topology, which is based on a generic architecture. An architectural topology consists of a set of resources employed to provide a dependable implementation of a set of control services.

The generic architecture to be employed is taken as a given in the topology selection process. Any challenge to the chosen generic architecture should be identified as early as possible in the design process. For instance, can the architecture provide sufficient support for the level of fault tolerance required by the proposed system? Can it provide sufficient computing power to fulfil the timing requirements of the system? If no variant of the generic architecture, that is an architectural topology, can be found to fulfil all these requirements, within given cost and size constraints, an alternate generic architecture will need to be chosen. Moving to a new generic architecture is potentially very costly both in discarded design work and time.

(iii) Configuration Level Decisions

By the *configuration* stage in the design process a set of sub-systems to provide the required functional and fault-tolerance properties of a system will have been identified.

An architectural topology for the system has been proposed. The form of the actual hardware platform still needs to be set.

The distinction between the configuration and architectural levels is blurred in practice. Some sub-systems may have developed to the point where coding has been undertaken. Others may only have WCET budgets set. The chosen topology selection approach must take this into account.

At this level a number of issues remain, including placement of software units onto the hardware platform. This placement will impact on the size of the required hardware platform and the timing characteristics of the system. These issues are relatively well defined with known alternatives, but providing solutions is very computationally expensive. The schedulability of the system is set at this stage and the sensitivity of allocations to changes in the WCETs of software units investigated.

(iv) Operational Life

A great many issues must be addressed in the selection of a topology. Some are highly visible such as meeting timing and reliability requirements. Others permeate the operation and evolution of systems such as a short system lifespan, difficulty in maintaining the system, and an inability to evolve within changing requirements. Evolvability may be defined as the ability of a system to adapt to or cope with changes in requirements, environment and implementation technologies.

Rowe [148] has produced an ontological framework to formalise the assessment of the evolvability of a computer system architecture. A similar ontological framework for a topology would allow the evolvability of a system to be included in a topology selection process. For instance, what would be the effect on the quality of a topology of a reduction in the Worst Case Response Time (WCRT) requirements for a service? Could this be accommodated by simply swapping to faster processors, or would a re-allocation of software units be required, or would this be possible only if a different topology was implemented? This enhancement is outside the scope of this thesis but could provide an important value-added element to the approach.

1.3 The Topology Selection Design Issue

In this thesis emphasis is placed on the resolution of the General Topology Problem (GTP) at the architectural and configuration levels of the design process, see Table 1.1. A Topology consists of a configured set of units employed to fulfil a given set of logical control actions. It may employ multiple, possibly diverse, copies of these units to ensure that dependability, timing, cost and functional requirements are met. Resolution of the

GTP requires the designer to select an acceptable topology with respect to a number of attributes that can be measured quantitatively.

Quantitative measures allow the quality of a proposed topology to be evaluated with respect to a topology model. Since the topology problem may be addressed at a number of points in the design process a hierarchical model is required, see Figure 1.3 and Chapter 3. The main feature of this model is that the set of fixed and variable elements in a topology change as design progresses. The variable elements constitute the degree of design freedom available to the designer at each point in the design process.

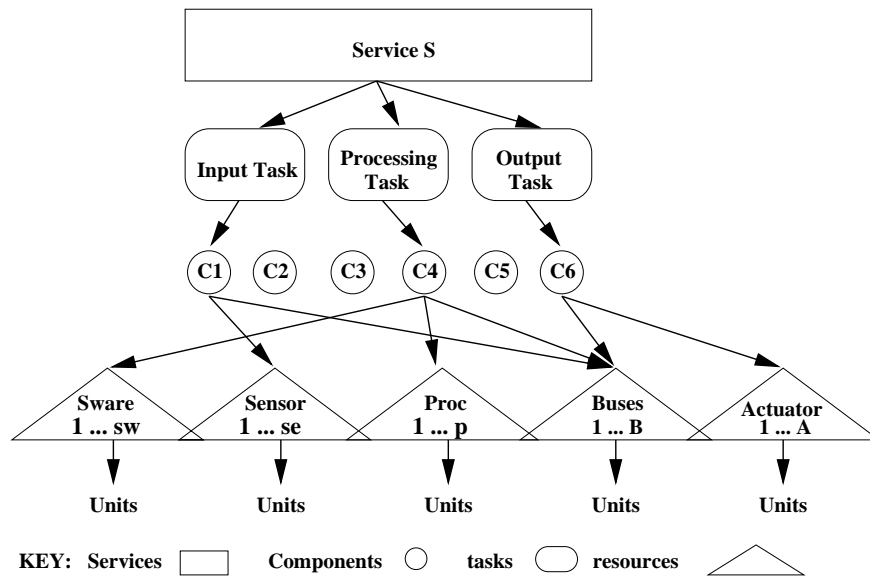


Figure 1.3: Hierarchical Model of a Topology

A control system must provide a number of control actions [23]. Control actions are implemented as a set of control Services ($s \in 1, \dots, S$) that form the logical architecture of a computer based distributed control system. A logical architecture is primarily aimed at satisfying functional requirements and embodies commitments that can be made independently of the constraints imposed by the execution environment. Services are the highest level of functional abstraction considered during the topology selection process. A single service is shown in Figure 1.3.

The logical architecture can be decomposed into a set of tasks ($t \in 1, \dots, T$). Each Service conceptually consists of three tasks; an input task, a processing task and an output task. Tasks form the second highest level of functional abstraction. A decomposition of the single service into three tasks is shown in Figure 1.3. The input task polls the environment for data pertinent to the required control action, the processing task determines the appropriate control action based on data provided by the input task, and the output task uses actuators to effect changes to the environment of the system

in an appropriate manner. At some later stage in the design process these tasks are decomposed into a set of precedence constrained sub-tasks (active processes).

The logical architecture for a control system also employs one or more communication networks. A network consists of one or more data buses which a given set of tasks have access to. Messages sent between, and within, tasks may be replicated for redundancy.

Designers need to physically implement the tasks in a dependable manner. One set of architectural components ($c \in 1, \dots, C$) can be employed to achieve the required level of dependability. A set of architectural components are shown in Figure 1.3. Note that not all of the components shown have been employed in the topology represented by Figure 1.3. The simplest component is known as a simplex component and consists of a single hardware unit. More complex components such as parallel, cold spare and triple modular redundancy can be implemented. The designer chooses exactly one admissible component for each task. This is a variable element in the selection of a topology at the architectural level and represents a degree of freedom to the designer.

Why should a designer choose to employ a hot spare *parallel* architectural component instead of a *simplex* component? The answer is influenced by dependability requirements. A task implemented as a parallel component can continue to provide a control action even when one hardware unit has failed. A task implemented as a simplex component ceases to provide a Service as soon as the single hardware unit fails. For control systems one important measure of the dependability of a Service is reliability. Reliability requirements are often couched in terms of the Mean Time to Failure (MTTF) of each Service in the system. This thesis concentrates on this reliability measure.

An architectural component may be physically implemented in a number of ways. It will however always require at least one hardware unit to implement a component and one hardware unit to give each task the ability to communicate with other tasks in the system. Thus, the size and form of each communication network is determined by the set of task components employed to implement the tasks that have access to the network.

Each unit employed in an architectural component is of a particular type, known as a resource. In this context a resource is a type of hardware, software or peopleware unit. Although the set of available resources is typically fixed the decision to use a particular resource to implement part, or all, of a component selected to implement a task is the job of the designer. Once a set of resources have been chosen the number of units of each resource employed can be calculated. In Figure 1.3 for instance, the component employed to implement the input task uses sensing units of sensor resource type 1 and bus units of bus resource type 1.

Thus, for a control system each task is implemented by picking one admissible component and choosing a set of admissible resources, and hence units, to implement this

component. For example, a processing task may be implemented as a simplex component that employs a single copy (unit) of an Intel 486 processing resource and a single copy (unit) of a software resource. The accompanying network may consist of a single copy (unit) of an Arinc 629 data bus resource. The overall set of hardware units employed by all the tasks in the system determines the hardware platform (set of configured hardware units).

Determining an optimal set of architectural components and resources to implement each task is the *architectural topology problem*. This problem is an architectural level issue and is investigated in detail in Chapter 3.

Software units employed in the proposed topology need to be placed onto an appropriate processing platform. At early stages in the design process during the resolution of the architectural topology problem a simplistic resource utilisation measure may be used to predict the number of units of each selected resource required in the hardware/software processing platform. At some point in the design process the set of architectural components and resources becomes fixed, or at least subject to little change. At this point the aim is to

1. determine an allocation of software units to individual hardware units, and
2. determine which sensors and actuators may be shared by which Services.

This is the *allocation* problem, which is a configuration level issue and is addressed in Chapter 4.

1.3.1 Audience for the GTP

The audience for the GTP partly dictates the appropriate formulation of the problem, and includes

- Designers of the distributed computer based control system
- Maintainers of the system.

Designers are responsible for producing the *best* control system given a set of constraints and system maintainers are responsible for the evolution of a system during its operational life. For this audience, there is also a desire to produce an optimal solution and so the GTP is formulated as an Assignment Type Problem (ATP), see Section 2.1.

There are other potential audiences for an approach to selecting a topology including:

- Equipment procurers

- Requirements engineers

This audience is more interested in the *feasibility* of a system. That is, given a set of constraints can a system be produced to meet these constraints. This audience wishes to consider different scenarios early in the design process, particularly of new and novel systems. For this audience the GTP can be formulated as a Satisfiability (SAT) problem, see Section 2.1, which are known to be NP-complete decision problems [52].

The exposition above has shown that the assignments to be made for the architectural topology problem are a single architectural component to each task and a set of resources to each component. A good architectural topology minimises cost and platform size while ensuring that Service reliability requirements are met. The assignments to be made for the allocation problem are a set of software units to hardware units of an appropriate type. A good allocation minimises cost and platform size while ensuring that worst case response time requirements (deadlines) are met.

1.3.2 Design (Artefact) Space

The ability to analyse, model and describe system design artefacts, see Figure 1.2, in a precise and unambiguous manner is vital to resolving the topology problem. Only then can the quality of a design be assessed accurately to determine whether the proposed system will meet the set of requirements on it.

A design artefact can be represented by a pair $\langle R, rel \rangle$ where R stands for the set of resources which the artefact is to be comprised of and rel denotes the set of relationships among these resources. Resources can be split into two categories: primitive resources that cannot be defined in terms of other resources and complex resources that can be defined in terms of previously described primitive resources.

A design (artefact) space [101] denotes the set of design artefacts that may be postulated for a particular design problem. The units used in a topology are particular instances of primitive resources that form one point in the topology design space. Sub-tasks, tasks and services are complex resources built from a combination of the primitive resources (units). Architectural components denote the relationships between primitive resources that form a complex resource (task). Thus, the set of topologies considered as part of a selection process may be represented as a design space.

The design space investigated by a design team is usually constrained to make trawling the set of available alternatives tractable. In fact, in industry the design space is often very heavily constrained [42]. The set of solutions to a design issue can therefore be split into three regions: typical solutions, feasible solutions and infeasible solutions (Figure 1.4).

The set of typical solutions may contain infeasible solutions, or at least very poor solu-

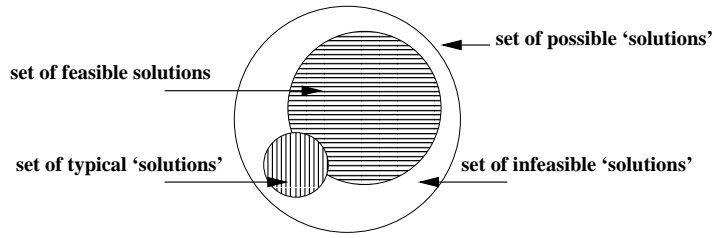


Figure 1.4: A Design (Artefact) Space

tions. This is because typical solutions are drawn primarily from past experience and domain knowledge. The rapid advancement in the complexity of system requirements for SC-RT systems may make such solutions infeasible. The aim is therefore to constrain the design space in such a way that the minimum number of infeasible solutions are investigated, while ensuring that the constraints do not inhibit investigation of good solutions.

For the topology problem a number of constraints are placed on the set of admissible topologies. These include the maximum number of diverse resources and the maximum number of hot/cold spares that may be employed per task. Other constraints may be placed on a topology selection space including resource usage, cost, reliability and timing. These constraints relate to the quality of a topology and indicate whether a topology is infeasible or feasible. Nevertheless the topology design space can become very large, see Chapter 3.

1.3.3 Analysis-Synthesis-Evaluation

The topology selection process can be viewed as a goal directed derivation process which starts with an initial set of design specifications and terminates with a topology. Topology selection can therefore be thought of as a process of synthesis with each synthesis step corresponding to a move from one synthesis state to another.

A synthesis state is described by a proposed design solution and the set of requirements that remain to be satisfied. After each synthesis step evaluation is undertaken. Evaluation attempts to test and verify the current proposed design against the available specification. Analysis is undertaken by the designer to determine whether a new design solution or a change in the specification is required in the next synthesis step.

A synthesis state is terminal if the specification is satisfiable by the design, in which case a design solution has emerged. A state is also terminal if neither the design nor specification can be further modified, in which case no feasible solution has emerged and the specifications have not been met.

This sequence of events is known as the Analysis-Synthesis-Evaluation (ASE) design

paradigm [101]. ASE is applied in the topology selection process presented in this thesis. A topology is selected by a synthesis step and evaluated with respect to a specification of a good topology. This process continues until an acceptable topology emerges. Analysis is then undertaken by the designer. If the topology is feasible then a solution to the topology selection problem has been found. If the topology is not feasible then the set of admissible synthesis steps is changed. That is either the specification of a good topology or the set of admissible topologies is changed. ASE continues until an appropriate topology emerges.

1.3.4 Definition of the GTP as an ATP

Three elements are required to define a problem as an Assignment Type Problem (ATP) [49], a set of items ($i=1,\dots,I$) to assign to a set of resources ($j=1,\dots,R$), an evaluation function to determine the quality of the assignments ($F(x)$), and a set of technical side constraints ($G_1(x), G_2(x), \dots, G_G(x)$). A solution can be represented by $X = [x_{ij}]$ where $x(i,j) \rightarrow (0,1)$ and $x(i,j) = 1$ implies item i is assigned to resource j . Thus, an ATP takes the form:

$$\begin{aligned} &\text{Minimise } F(x) \\ &\text{subject to} \\ &G_g(x) = 0 \quad 1 \leq g \leq G \end{aligned}$$

The set of resources item i can be assigned to is $R_i \in 1,2, \dots, R$, which is a sub-set of the set of admissible resources. The function $F(x)$ is not restricted to having any specific property other than being calculable. A side constraint, $G_g(x)$, is defined to take on a value of zero if and only if the constraint is met. If one, or more, constraints are not met $G_g(x)$ takes on a value greater than zero. The function $G_g(x)$ must be calculable.

The set of assignments employed in an ATP formulation of the GTP represent a single possible solution to the GTP introduced in this Section, see Figures 1.3 and 1.4. Designers need to generate a set of control services and tasks. Five sets of assignments are employed to produce the remaining elements of a topology:

A1: An architectural component to each task in the system, x_{ct} . The set of admissible components for task t is C_t .

A2: A set of resources to each component-task combination in the topology. The set of admissible resources for combination ct is $R_{(ct)}$.

A3: Units of one resource to units of a second resource, $x_{u\bar{u}}$. For instance, software units need to be assigned to processing hardware units. The set of admissible units for

unit u is R_u .

In Figure 1.3 the buses employed in a topology are represented as being part of an architectural component. However, since more than one task can use the same network two further assignments are required to determine the networks employed in a proposed solution to the GTP.

A4: An architectural component to each network in the system, x_{cn} . This assignment determines the number of copies of each message sent for fault tolerance purposes. The set of admissible components for network n is C_n .

A5: A set of resources to each component-network combination in the topology. The set of admissible resources for combination cn is $R_{(cn)}$.

To summarise, the particular values for these assignments form one possible solution to the GTP. Solution x can be formally defined as $X = [x_{ij}]$ where ij are the set of assignments represented by A1 to A5.

Solution x can be evaluated with respect to the goals of the designer, represented by $F(x)$, and the constraints on what constitutes a feasible solution, represented by $G_g(x)$. The goals for the GTP are:

F₁: Minimise cost of the topology

F₂: Minimise size of the platform

The constraints on the GTP are:

G₁: a set of Service WCRT targets, Max WCRT_s

G₂: a set of Service dependability targets, Min depend_s

G₃: a set of unit capacity constraints, Max num_{uū}

G₄: a set of restrictions on unit to unit assignments, Req_d parallel_u

For a solution, x , to be feasible all four sets of constraints must take on a value of zero.

A mathematical formulation of the GTP as an ATP is presented in Section 2.1 and the elements of a solution to the GTP are investigated in the remainder of Chapter 2. A satisfiability (SAT) formulation of the GTP is also presented.

1.4 Motivation for Approach Employed

The approach developed in this thesis is primarily aimed at providing the designer with automated help in resolving the topology problem. It can however be extended up the design life-cycle to be of assistance to procurers and requirements engineers. It can also be extended into the operational life of the system. Comments on these issues will be made as they arise.

Four characteristics of existing SC-RT control systems have led to the development of

the approach presented in this thesis:

1. Complexity of design solutions. Many SC-RT control systems are inherently complex. However, examination of systems produced by a number of industrial organisations has revealed extra complexity introduced to overcome problems identified late in the design process. Complexity may be introduced when a proposed system fails to meet timing and reliability requirements. This is a problem as complexity often increases the risks attached to a system.
2. Inter-dependence of attribute values. SC-RT systems require values of a number of attributes to be within a given range. Unfortunately, the values these attributes take are inevitably inter-dependent. Any approach to the topology problem must take this into account.
3. Failure to adopt quantitative reliability prediction techniques. A plethora of predictive reliability models have been produced. These models allow quantitative measures of the ‘likely’ reliability of a proposed system to be produced. They are not currently widely used as a design aid.
4. The mapping problem. The mapping problem is a well known problem that has historically focused on allocating processes so that timing requirements can be met. Mapping has generally not been done well in industrial practice with consequent inefficiencies and expensive reworking of the proposed design solution. However, mapping problems can be simplified by addressing elements of the problem earlier in the design process.

Previous work by the author has centred on the allocation of software processes to a given platform late in the design process. The result is *pass* if an allocation can be found that meets a set of timing requirements and *fail* otherwise. A fail result often leads to costly reworking and increased design complexity. Design complexity creeps in because it is too expensive to *start from scratch*. A trade-off between timing and functional complexity is made. The answer is to help the designer predict whether a valid allocation can be made earlier in the design process. Rework is cheaper the “higher up” in the design process any required functional reworking can be undertaken.

As a result emphasis shifted to answering two questions. First, which attributes of a system can be predicted before an artefact has been produced. Second, which elements of the allocation problem should be extended to facilitate better solutions to the mapping problem. The architectural topology appears to be the appropriate element in the allocation problem to address.

An architectural topology partly determines the set of software units to be allocated by setting the extra functional units to be employed for fault tolerance and reliability.

It also indicates the hardware resources to be used in the system platform. Software units are mapped to the processing elements of this platform.

A number of questions arise from a decision to address the architectural topology problem as a variable in the setting of a system topology:

- Is it possible to facilitate the emergence of a topology alongside the functional design process?
- Can an approach be devised that allows an architectural topology to be proposed and evaluated from an early stage in the design process?
- Will producing an architectural topology early in the design process allow a mapping of software units to hardware units to be proposed earlier in the design process than existing mapping techniques?
- Can the techniques employed to predict a topology during design be employed during the operational life of the system?

The resolution of these questions requires more detailed questions to be answered, such as:

- Can a set of quantitative measures be produced that allow a postulated topology to be evaluated to determine cost, reliability and timing attributes?
- Is it possible to produce tools to automatically search a set of possible architectural topologies and allocations to suggest a topology to the designer that exhibits appropriate characteristics?

This thesis contends that the answer to these questions is yes.

The topologies produced by the approach presented in this thesis exhibit fault-tolerance, a configured set of resources and an allocation of software units to hardware units. A physical architecture provides the link between the architectural topology and allocation sub-problems. The approach is new and novel. Evidence is provided to justify the approach. The utility of the approach is also shown in the form of illustrative examples and case studies.

Tool support is provided by guided search techniques that can be applied throughout the design process. Guided search techniques are employed because they

- are compatible with an ASE design process
- are able to search large design (artefact) spaces effectively

- use designer experience to guide the search during each experiment
- use designer knowledge to alter the artefact space during the design process
- employ quantitative measures of the quality of a topology that take into account the inter-dependencies of cost, reliability and timing
- are flexible in that there is a family of related techniques
- are powerful search techniques that are relatively easy to code, understand and extend

Two prototype tools, *X-Topmeter* and *X-Alloc* have been produced to demonstrate the approach. In particular the power and flexibility of the approach as a design aid is emphasised.

1.5 Structure of Thesis

In Figure 1.3 a representation of a topology was presented. Selecting a topology in the context of the design process for safety-critical real-time systems, and previous experience, forms the basis of the structure of this thesis. Thus, in Chapter 1 the elements of a SC-RT control system, the ASE design process and an informal formulation of the GTP have been presented.

The approach and information required to implement the approach are very entwined. Thus, a review of the literature is introduced, where appropriate, in Chapters 2 to 4. In Chapter 2 the elements of a topology and the techniques that may be used to select and evaluate a proposed topology are introduced. Structures investigated include the units employed in a topology (hardware and software) and, logical and physical architectures. Techniques investigated include designing for dependability and fault-tolerance, evaluation of the dependability and timing attributes of a topology, and guided search techniques.

In Chapter 3 a Genetic Algorithm is employed to aid selection of an architectural topology. The results of this selection process are then used in Chapter 4 as part of the resolution of the variable platform size allocation problem. A Simulated Annealing algorithm is employed in Chapter 4.

Interactions between the the architectural topology problem, allocation problem and the functional design process are discussed in Chapter 5. Supporting evidence for the quality of the topology selection process introduced in Chapters 3 and 4 is presented, including a Computer Assisted Braking system case study.

Finally, in Chapter 6 a summary of achievements, comparison with related research and proposals for future work are presented. Appendix A expands on the implementation of the prototype guided search tools employed in this thesis.

Chapter 2

The Elements of a Topology

Selecting a topology is an N-dimensional design issue. The aim of this chapter is to investigate whether two fundamental requirements for a topology selection approach can be determined for a wide range of potential control applications. Namely, the ability to determine an appropriate set of plausible topologies and the ability to evaluate the quality of each proposed topology. To set the scene for this investigation formal definitions of the GTP as an assignment type problem and a satisfiability problem are presented in Section 2.1.

The set of admissible topologies determines the design space for a topology selection process. That is it acts as a constraint on the form of the final topology selected by the process. The quality of a topology is dependent on the set of hardware and software resources available to a designer. It is also dependent on the functionality, timing and dependability attributes of the topological configuration employed.

A review of the units (resources) required in a topology and the attributes of these units is undertaken in Section 2.2. In Section 2.3 the dependability attributes of a proposed topology are presented. One method by which the dependability of a system can be improved is to introduce fault-tolerance and therefore a set of possible fault-tolerance techniques is investigated. Fault tolerance is the ability of a system to continue to perform its tasks after the occurrence of a fault. The literature on reliability modelling and evaluation is reviewed to show that predictions of the reliability of systems that employ a range of admissible topologies can be produced.

If a designer wishes to use a predictive reliability measure to aid the topology selection process an approach which takes into account the system design framework is required. The ASE approach introduced in Section 1.3.3 is employed in this thesis. This approach is compatible with the V-model introduced in Figure 1.2. Reibman & Veerarghavan's [144] six step approach would be an acceptable alternative.

These approaches allow the designer to change the reliability model employed as the

design process progresses. This is a very important ability, given the nature of the design issues to be supported. In particular it allows reliability measures to be employed at multiple levels in the design process. A reliability measure is proposed as one means of evaluating the *quality* of any topology proposed by the designers of a system in Section 2.4.

In Sections 2.5 and 2.6 the aim is to introduce a schedulability analysis that can predict whether the timing requirements for a control system can be met by a proposed topology. The review considers both uni-processor and distributed systems.

Finally, in Section 2.7 guided search techniques that may be used to traverse the topology design space are investigated. Particular emphasis is given to Genetic Algorithm (GA) and Simulated Annealing (SA) techniques.

2.1 Formal Definition of the GTP

Given the model and assignments presented in Section 1.3, the GTP can be formulated as an ATP. Thus, for a given generic architecture, the formulation for the GTP as an ATP is:

Minimise $F(x) = \sum_{u \in U} (num_u * W_u) + (\gamma_u cost_u)$		Size & cost goals
subject to		Constraints:
$\sum_{c \in C_t} x_{ct} = 1$	$1 \leq t \leq T$	Assignment
$\sum_{c \in C_n} x_{cn} = 1$	$1 \leq n \leq N$	Assignment
$\sum_{r \in R_{(ct)}} x_{r(ct)} \geq 1$	$1 \leq t \leq T$	Assignment
$\sum_{r \in R_{(cn)}} x_{r(cn)} \geq 1$	$1 \leq n \leq N$	Assignment
$\max WCRT_s - WCRT_s \geq 0$	$1 \leq s \leq S$	Timing
$depend_s - \text{Min } depend_s \geq 0$	$1 \leq s \leq S$	Dependability
$\max num_{u\bar{u}} - num_{u\bar{u}} \geq 0$	$1 \leq u \leq U$	Capacity
$parallel_u - \text{Reqd } parallel_u \geq 0$	$1 \leq u \leq U$	Fault Tolerance

The elements of a GTP formulated as an ATP are:

x	Set of proposed assignments, $X = [x_{ij}]$
s	Service: logical control action to be provided by the system.
S	Number of services in the system.
t	Task. Total number of tasks is T.
n	Communication network. Total number of networks is N.
r	Resource: type of unit that may be selected for a topology.
c	Architectural component
u	Unit: smallest indivisible item, such as processor.
U	number of units in the system.

$F(\mathbf{x})$	quality of proposed resource assignment
num_u	number of unit u employed in the system, 0 if non employed
W_u	penalty for using unit u
γ_u	penalty weighting factor for cost of unit u
$cost_u$	ownership cost of unit u
$\max WCRT_s$	maximum permissible WCRT of service s
$WCRT_s$	predicted worst case response time of service s
$\text{Min } depend_s$	minimum level of dependability required
$depend_s$	measure of the dependability of service s
$\max num_{u\bar{u}}$	maximum number of unit u employable on unit \bar{u}
$num_{u\bar{u}}$	number of unit u to be implemented on \bar{u}
$\text{Reqd } parallel_u$	number of units to be placed in parallel with u
$parallel_u$	number of units actually placed in parallel with u

These elements are discussed in some detail in this chapter. Furthermore, in Section 3.2.1 an ATP formulation of the Architectural Topology problem is presented. The resulting architectural topology, including a set of software units, provides data to the allocation problem, which is also formulated as an ATP (Section 4.1.5).

The General Topology Problem as SAT

Propositional Satisfiability (SAT) [155] is an NP-complete decision problem which poses real challenges for optimisation methods, see Section 2.7. In fact it has been stated that a problem is NP-complete if a good algorithm for it would entail a good algorithm for SAT. It can be informally defined as:

“Given a formula of the propositional calculus, decide if there is an assignment to its variables that makes the formula true according to the usual rules of interpretation [155].”

In other words define a set of boolean variables ($v=1, \dots, V$) that take on the value true if a specified constraint is met and false otherwise. One or more of these boolean variables can be combined by the or combinator to form a clause ($cl=1, \dots, CL$). If only a single boolean value in a clause is to take on a value of true the sum operator may be employed. This is because the sum of booleans equal to one is stronger than the inclusive or operator, and implies it is true. The aim in a SAT problem is to minimise the number of unsatisfied clauses. If the set of all clauses are true then a feasible solution to the SAT problem exists.

For the GTP the set of assignments that can be made is the same as the ATP formulation. The size and cost goals are reformulated as constraints. In total 11 clauses can be identified for the GTP. The formulation of GTP as a SAT is:

Minimise $F(x) = \sum_{cl \in CL} \epsilon_{cl} \text{ Clause}_{cl}$		Feasibility goal
subject to		Clauses:
$\text{Min } depend_s - depend_s \leq 0$	$1 \leq s \leq S$	Dependability
$WCRT_s - \max WCRT_s \leq 0$	$1 \leq s \leq S$	Timing
$\sum_{u \in U} cost_u - \max cost \leq 0$		Cost
$\sum_{u \in U} num_u - \max num_u \leq 0$		Size
$num_u - \max num_u \leq 0$	$1 \leq u \leq U$	Size
$num_{u\bar{u}} - \max \sum_{u \in R_u} x_{u\bar{u}} \leq 0$	$1 \leq u \leq U$	Capacity
$\text{Reqd } parallel_u - parallel_u \leq 0$	$1 \leq u \leq U$	Fault Tolerance
$\sum_{c \in C_t} x_{ct} = 1$	$1 \leq t \leq T$	Component-Task
$\sum_{c \in C_n} x_{cn} = 1$	$1 \leq t \leq T$	Component-Network
$\forall r \in R_{(ct)} x_{(ct)} \geq 1$	$1 \leq t \leq T$	Resource-component-task
$\forall r \in R_{(cn)} x_{(cn)} \geq 1$	$1 \leq n \leq N$	Resource-component-network

The elements of a GTP formulated as a SAT are the same as those employed for the ATP definition except:

cl	clause ($cl=1, \dots, CL$)
Clause_{cl}	0 if clause cl satisfied, 1 otherwise
$\max cost$	maximum acceptable cost for a topology
num_u	number of units in the topology
$\max num_u$	maximum number of units in the topology
$\max num_u$	max allowable number of unit u in the topology
$num_{u\bar{u}}$	number of unit u on unit \bar{u}

2.2 Topological Units and Resources

Units ($u=1, \dots, U$) are the indivisible items employed to form a topology. Examples of topological units include a software module, a processor, an actuator and an operator. A number of units are grouped together to form structures that provide particular functionality such as sensing, determination of an appropriate control action and implementation of an action.

A resource ($r=1, \dots, R$) is a type of unit, such as an Intel 486 processor or a software voter module. In a project to produce a distributed SC-RT control system the set of resources that may be employed by the system designers form one limitation on the system that can be produced. The set of resources available to designers, and maintainers, of a system may not remain constant through time. For instance, new versions of particular software modules or new types of processors may be introduced.

In this thesis it is assumed that a library of admissible resources is available. Some

of the resources in this library may be the product of previous projects or result from particular procurement decisions. Others will be produced specifically for the project. The approach presented in Chapter 3 may be employed to aid procurement decisions, as the effectiveness of topologies employing different resources can be predicted and compared. Furthermore, if necessary the approach can help determine a new topology for a system when a new resource is introduced.

2.2.1 Reliability of Units and Resources

One attribute of a non-repairable control system that is determined by the chosen topology is reliability. The time to failure of individual units fundamentally determines the reliability of a proposed topology. Stochastic modelling is used to produce a distribution function, $F(t)$, to characterise the dynamic behaviour of the time to failure of units.

Units are combined to form tasks and services and the time to failure of these complex resources can also be characterised by a failure distribution function. It can be shown that

$$\text{Reliability}(t) = 1 - F(t)$$

Many failure distribution functions have been used in the literature. In this thesis the Exponential Polynomial [152] (EP) family of distributions is employed. The EP family is closed under operators such as addition, multiplication, convolution, and probabilistic sum. Thus, a hierarchy of models may be constructed for the units, sub-tasks and tasks in the system. The distribution functions produced can then be combined to produce service failure distribution functions and hence a measure of the reliability of each service, *depend_s*.

The most commonly employed failure distribution is the exponential distribution, which is an EP distribution with a single parameter, λ , that characterises a constant failure rate per unit time. That is

$$F(t) = 1 - \exp[-\lambda t]$$

Other well known failure distributions, such as the Weibull, can be approximated using EP distributions.

Hardware

Failure rate predictions for hardware have been considered by many authors. Most large companies have their own standards, some of which are now in general use. Of

particular importance is the MIL-Hdbk-217 standard [166], although this has now been downgraded to advisory status. It is assumed in this thesis that the failure distribution of any hardware can be predicted. The interested reader is pointed towards Bowles [16] who considers the predictions of six hardware component reliability prediction procedures.

The bathtub curve is the most commonly employed hardware component life-time distribution model and consists of a burning-in section where failure rates fall; followed by a section with a constant failure rate; followed by a section of increasing failure rates. The exponential distribution is used to model the constant failure rate portion of this curve. More complex distributions, such as the Weibull distribution, can be used to model the full curve.

Work at the European Space Agency (ESA) [176] that considers both random ageing failures and design failure in complex hardware, indicates that the life-time distribution should be a roller-coaster curve. In the roller coaster curve an initial burning in is followed by an increase in the failure rate of the unit. This is caused by the exercising of design faults. After a short period the failure rate falls to a low and constant rate. There is little evidence of increasing failure rates due to age in modern electronic equipment. This failure curve can be modelled using EP distributions and therefore could be incorporated into the reliability model of a topology.

Software

Software does not fail in the same way as hardware. Hardware can fail on a random or a systematic basis, as shown by the bathtub and roller-coaster curves considered above. Software can only fail as a result of failures designed into it, either by an incorrect specification against intent or an incorrect implementation of this specification. Defects are exposed by triggering some deficiency in the software design.

A number of authors, including Arlat et al [4], Carmen et al [25] and Nikora & Lyu [123] have modelled the general behaviour of a software unit. Arlat, see Figure 2.1, talks in terms of a benign failure or a permanent malign failure. If a failure is *benign* the consequences of the failure are of the same order of magnitude as the benefit delivered in the absence of the failure. If a failure is *malign* the consequences of the failure are immeasurably disproportionate to the benefit provided by the system in the absence of the failure. Techniques to tolerate software failures are based on information about the state of the system, see Section 2.3.4.

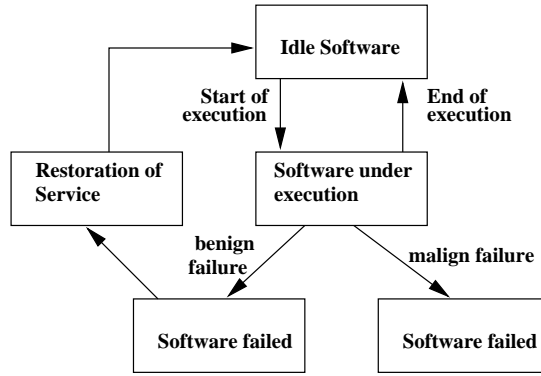


Figure 2.1: Software Failures

A control system employs software units, each of which is built to a particular *Safety Integrity Level* (SIL). A feature of the integrity level approach is the ability to make claims about the failure rate of software built to the development strategy mandated by the chosen integrity level. That is, the standards provide bounds on the reliability which can be claimed for software developed to each integrity level. Failure rate claims such as those produced by MISRA, see Table 2.1 [109], may be used in the early stages of the topology selection process.

SIL	Safety related continuous control systems (failures per hour)	Safety related protection systems (failures per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to 10^{-4}
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to 10^{-3}
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to 10^{-2}
1	$\geq 10^{-6}$ to 10^{-5}	$\geq 10^{-2}$ to 10^{-1}

Table 2.1: Failure Rate Claims for Software Units

Once a project has progressed to the stage where code is being produced, estimated failure rate data [124] [123] can replace failure rate claims. Carman et al [25] have produced a Software Reliability Engineering (SRE) method that is compatible with our approach. It should be noted however that such activities take place relatively late in the design process when a first version of the system has been produced. The approach presented in this thesis assumes that the point at which *real* software failure rate *estimates* are incorporated into the data sheet of a software resource is an application / company best practice decision [81].

2.2.2 Logical and Physical Architectures

The concept of a separate logical and physical architecture was introduced by Lister & Burns [23] as part of a framework for building dependable systems. A logical architecture introduces commitments that can be made independently of the constraints imposed by the platform the system is to be placed on. A commitment defines properties of the system design which designers operating at a more detailed level are not at liberty to change. A logical architecture is primarily aimed at meeting functional requirements. It thus forms the basis on which a topology selection is made by restricting the set of admissible topologies. For instance, the fact that a particular topology must employ a bus based communication system and support three control services are commitments on the topology.

A physical architecture forms the basis for asserting that non-functional requirements will be met once detailed design and implementation have taken place. It does so by refining the logical architecture in two ways:

1. it instantiates the elements of the logical architecture by mapping them to a target execution environment (platform) such that the set of given constraints are met. The platform for a system consists of a set of hardware and associated software units;
2. it annotates the elements of a logical architecture with non-functional attributes, such as dependability ($depend_s$) and timing ($WCRT_s$).

A topology is one possible implementation of the physical architecture of a system.

In Chapters 3 and 4 the resolution of the topology problem is guided by the ability of a designer to produce a logical and physical architecture for a proposed distributed control system. Chapter 3 focuses mainly on producing a physical architecture and determining the set of resources to be employed in the platform, for a given logical architecture. Chapter 4 focuses on producing a system platform, and allocating software to this platform, such that timing commitments are met, for the given physical architecture.

2.3 Designing for Dependability and Fault Tolerance

Computer systems that control real-world applications must have certain ‘desirable’ properties if they are to be relied on by both users and the public. One such desirable property is dependability. This is particularly evident for the type of control systems introduced in Chapter 1. For example, in a fly-by-wire aircraft the flight control system must have an acceptably low failure rate. Authors ranging from Hosford [72]

to McDermid [107] and Laprie [91, 92] have attempted to define dependability. The most widely accepted definition is from Laprie:

“Dependability is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behaviour as it is perceptible by its user(s)”

The literature does not agree on the terminology used to describe the attributes of a ‘dependable’ system. A great deal of confusion has arisen as a result. The dependability terminology proposed by Laprie, see Figure 2.2, is employed in this thesis.

The dependability of a computer based system is partly determined by the type of impairments that can arise as a result of the design employed, the means by which impairments are overcome in the form of a topology, and the requirements of a proposed control system. To produce a set of plausible topologies and analyse them for their dependability all three aspects must be addressed.

2.3.1 Impairments to Dependability

The dependability concepts of fault, error and failure are fundamental terms in fault tolerant design. Unfortunately, once again, no fully accepted set of definitions has been produced [77]. A fault may be defined as a defect, or flaw, that occurs within some hardware or software component that can give rise to an error under some circumstances. This defect may be a systematic or a random defect. An error is a deviation from accuracy or correctness that results from a fault. Finally, if an error results in the system performing one of its functions incorrectly, then a system failure has occurred.

The interaction between these concepts has been characterised by a number of authors in terms of a three-entity model. For instance, Prasad et al [132] talk in terms of types of system ‘badness’ that are linked both causally and chronologically:

1. A dormant ‘bug’ exists in either hardware or software. It may be activated at some point during the lifetime of the system or get fixed by some recovery mechanism.
2. The active manifestation of the dormant bug is an error.
3. The externally visible result of the active entity, when it has escaped to the ‘outside’ world, is a failure.

Prasad’s dormant, active and externally visible model is adopted in this thesis. The topology selection approach does not preclude other views on this issue.

The quality of a selected topology is partly determined by the ability of a system to cope with errors. A classification of the type of failures that can arise in a computer based control system is required. Once this set has been established means by which errors can be stopped by a topology from becoming failures can be addressed. A design philosophy can be adopted to enable a set of alternative topologies to be defined.

Not all systems may exhibit all types of failure as a result of the computational model, see Section 2.5, and safety kernel employed [22]. The computational model and safety kernel guard against, or transform, particular types of failures. For instance, an incorrect computation fault may be transformed to an omission failure. In this case no extra fault tolerance against value failures is required.

The literature has produced a number of fault classifications that appear appropriate for SC-RT control applications. For example, Ezhilchelvan & Shrivastava [48] introduce a five element classification. Barborak et al [9] present an ordered fault-classification taxonomy that has the property that a stronger class is a subset of a weaker class. Barborak's classification is:

- *Fail-stop fault*: hardware ceases operation and alerts other units of this fault.
- *Crash fault*: hardware loses its internal state or halts.
- *Omission fault*: hardware fails to complete a task
- *Timing fault*: hardware completes a task either before or after its specified time frame or never. Timing faults will be considered in Section 2.5.
- *Incorrect computation fault*: hardware fails to produce the correct result in response to the correct inputs.
- *Authenticated Byzantine fault*: an arbitrary or malicious fault, such as when one hardware unit sends differing messages during a broadcast to its neighbours, that cannot imperceptibly alter an authenticated message
- *Byzantine fault*: every fault possible in the system model. Solving this fault is the famous Byzantine Generals Problem, which requires $3f+1$ copies to tolerate f failures.

Software and smart hardware, such as smart sensors and actuators, may potentially exhibit the full range of faults classified by Barborak. A simple hardware unit cannot alert other units that it has failed. Therefore, the type of failures exhibited by simple hardware are a subset of those given here. For the purposes of this thesis Barborak's classification has been chosen because it is intuitive and complete. Other classifications are not prohibited. The particular types of failure that a system can exhibit is

important in that it affects the set of topologies that may need to be implemented by the designer.

2.3.2 Attributes of a Dependable System

Failure of a control system to provide the required control actions may be catastrophic. In Figure 2.2 Laprie indicates six attributes of interest to a designer of dependable systems. Although confidentiality may be important it is not central to this thesis. Thus, the quality of a topology is assumed to be a function of the reliability, availability, safety, integrity and maintainability of the proposed topology. A system is subjected to techniques designed to ensure fault tolerance in an attempt to provide acceptable values for these five attributes.

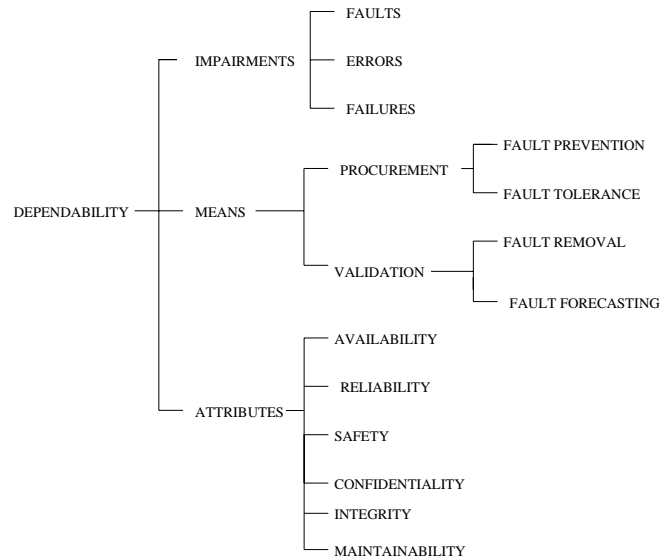


Figure 2.2: Laprie’s Dependability Terminology

The availability of a system is defined to be the probability that the system is working at time t , regardless of the number of times it may have failed and been repaired in the interval $(0,t)$ [152]. For control systems availability is often not an issue as the system is not normally subject to repair while in operation. It is not proposed to consider availability in the prototype tools introduced in Chapters 3 and 4.

The first failure to occur may cause the system to exhibit a catastrophic failure and therefore reliability is important to the system designer. Reliability may be quantitatively defined as the ability of a component or system to function correctly over a specified period of time [152]. It may also be defined qualitatively as no failures occur in a specified period of exposure, usually a mission duration or inter-maintenance period. Different fault tolerance approaches can guard against particular types of faults and hence failures thereby increasing the reliability of the system. The designer

chooses from a set of available fault-tolerance techniques to ensure acceptable system reliability.

Safety is the ability of a system to avoid causing unacceptable harm, that is loss of life or environmental damage [168]. Safety is not synonymous with reliability in that a control system may be operating but initiate erroneous control actions. If the failure is *benign* it may be deemed acceptable to put little or no effort into tolerating the failure. If the failure is *malign* then great efforts should be made to tolerate the failure.

Fault-tolerance can be employed to improve the safety of a system. Different fault-tolerance techniques imply different levels of safety. Choi et al [28] employ a Mean Time Between Hazardous Event (MTBHE) as a measure of the impact of a system architecture on system safety. They present a comparison of a number of dependable architectures. A measure of this kind could be employed in the architectural topology evaluation function introduced in Chapter 3.

The *Integrity* of a system is the likelihood that a safety related system will satisfactorily perform required safety functions under all stated conditions within a stated period of time. Safety Integrity Levels (SILs) provide an indication of the required level of protection against failures.

Maintainability is the ability of an entity to be maintained in, or restored to, a state in which it can perform a required function. Maintainability is an attribute of the system that is difficult to predict quantitatively. Decisions made in order to improve maintainability may have a beneficial or detrimental effect on reliability and safety. Conversely, the application of fault-tolerance may affect the maintainability of a proposed system. For instance, if a system employs a number of copies of a single type of processor it is likely to be easier to maintain than a system with a number of different types of processor. The ownership cost of hardware and software employed in a system will partly reflect the ease or difficulty of maintaining the system when particular fault-tolerance techniques are employed.

These five attributes provide the justification for employing a fault-tolerant topology. A topology employing different fault-tolerant techniques will improve one or more dependability attributes of the system, but may be detrimental to other attributes. Trade-offs of this nature form the core of the design decisions for a topology selection process.

2.3.3 Fault Tolerance Design Philosophies

Three primary philosophies have been employed to improve or maintain a system's normal performance in an environment where failures are of concern. These are fault avoidance, fault masking and fault tolerance. These philosophies can be linked to

Prasad's three-entity model. Fault avoidance techniques attempt to prevent the occurrence of dormant bugs in the first place. Techniques include testing, design reviews and techniques to overcome specification mistakes. Fault masking is any process that prevents dormant bugs in a system from introducing errors into the informational structure of that system [172]. The aim of fault-tolerance is to stop errors causing failures.

Perusal of the literature soon shows that these design philosophies are not applied in isolation. A single technique may be employed by a designer to both mask and tolerate a particular failure, such as an Omission failure in a processor. A set of techniques that may be employed to stop bugs becoming active or errors becoming externally visible are required. Fault Tolerance is used as a portmanteau term.

The common factor in all fault tolerance approaches is redundancy. Redundancy is simply the addition of information, resources, or time beyond what is needed for normal system operation. The set of admissible fault tolerance techniques for a particular system will be a subset of the techniques outlined in the following sub-sections.

2.3.4 Topologies for Dependability

System designers have a range of fault-tolerance techniques to draw on. These techniques can be employed at any level in the system, that is unit, sub-system or the system as a whole. In this Section work by Kazmann et al [82], Laprie [4], Kelly et al [84], Welke [172] and many others is drawn on to develop a set of topologies that exhibit the ability to tolerate different types of failure. The set of possible failures is presumed to be that surveyed by Barborak.

Of particular interest to a designer of a topology are redundancy techniques [27, 161]. A five element classification of the techniques that are most likely to be employed in non-repairable RT-SC systems is:

1. *Static redundancy* - Faults are masked through a majority vote involving a fixed group of redundant components.
2. *Dynamic redundancy* - Faults are not masked from causing errors at the output, but the faulty components are detected, isolated and reconfigured out of the system.
3. *Hybrid redundancy* - Faults are masked through a majority vote involving a group of redundant components which are reconfigured when spares are available.
4. *Adaptive voting* - Faults are masked through a majority vote involving a variable group of redundant components without spares.

5. *Adaptive hybrid* - Faults are masked through a majority vote involving a variable group of redundant components which are replaced when spares are available.

Many fault tolerance techniques use some form of decision mechanism. For instance, sensing tasks may employ sensor fusion mechanisms. Similarly, processing tasks may employ voters or acceptance tests to determine an appropriate control value. In Kelly et al [84] elements of the decision process for determining consensus are considered in some detail. Here a set of possible redundant topologies, some of which employ decision mechanisms, are presented.

Hardware Fault-Tolerance

Consider a very simple logical architecture consisting of a single hardware unit. The system may also support a single software unit, which is assumed to be 100% reliable. The system can fail due to a hardware failure. This is known as a simplex topology and can be represented diagrammatically as a single block. The topology can tolerate no failures and the reliability of the system is dependent on the reliability of the hardware employed.

The designer of a hardware system could decide to implement a hot spare topology, see Figure 2.3. In hot sparing the system has m extra components all of which are active. Data from whichever component completes first is taken as the correct output value. This topology allows fail-stop, crash or omission failures to be tolerated. The number of such failures that can be tolerated is $n-1$, where n is the number of hardware units employed.



Figure 2.3: Parallel Hardware Redundancy

The designer could decide that an omission failure is more acceptable than producing an incorrect value. Thus, an acceptance test is added to each unit in an attempt to guard against incorrect computation failures, see Figure 2.4. The reliability of this topology may be greater or less than that of the parallel design depending on the relative probabilities of a value failure and a failure in the acceptance test.

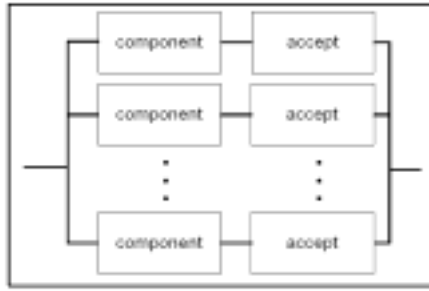


Figure 2.4: Acceptance Hardware Redundancy

A second extension to the hot spare topology, is to employ a decision mechanism in the form of a majority voting algorithm. This is known as N-modular redundancy (NMR). In an NMR topology n copies of the hardware component are employed in parallel and a majority voter used to identify an acceptable result (Figure 2.5).

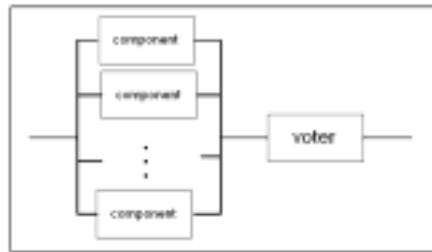


Figure 2.5: N-Modular Redundancy

One of the most common versions of this topology in common usage is Triple Modular Redundancy (TMR) which has three parallel copies of a component and a majority voter (Figure 2.6).

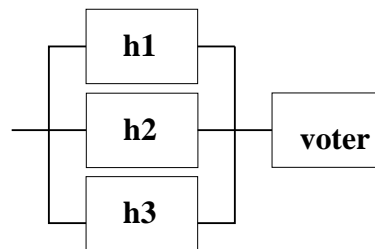


Figure 2.6: TMR Hardware Redundancy

A further extension is to employ two complete TMR topologies in parallel (Figure 2.7). The first result to emerge is employed downstream.

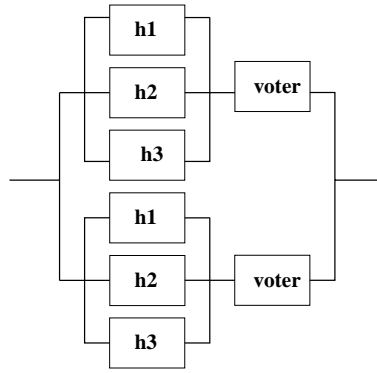


Figure 2.7: Twin TMR Hardware Redundancy

Cold spares can be used to implement a fault-tolerant topology in preference to hot spares (Figure 2.8). In cold (standby) sparing only one copy of each component is active at a time. The component is designed to be *fail-stop*. A switch determines whether the active hardware has failed and switches over to a spare hardware unit. It is assumed that the failure of each hardware unit is independent of the failure of other units. This assumption is not valid if common cause failures [80] can occur.

A cold spare topology can tolerate as many fail-stop and omission failures as there are spare hardware units. However, it does so at the price of extending the time taken to complete the task and may thus cause a timing fault. It may also lose data and produce an estimated result leading to a value failure.

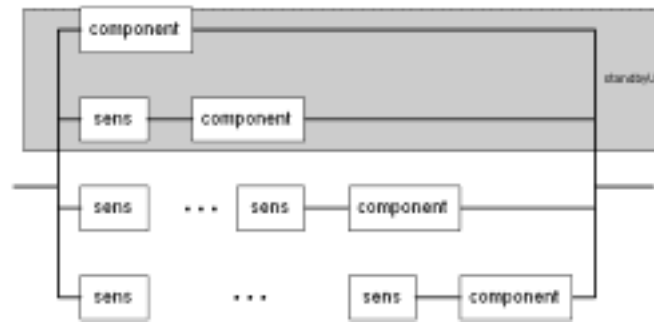


Figure 2.8: Cold Spare Hardware Redundancy

The topologies presented in Figures 2.3 to 2.8 assume independent failures. Techniques to overcome common cause failures may be employed to alleviate the symptoms of Byzantine failures. A Common-cause / Common-mode (CC/CM) failure occurs when multiple copies of a redundant component suffer faults near simultaneously, generally due to a single cause [88].

Kalbarczyk & Christmanson [80] have produced an excellent survey of the techniques applied to reduce the probability of a CC/CM failure. Three types of statistical de-

pendence involved in CC/CM have been identified:

1. dependence among failure events
2. dependence of failure events on conditions under which the component is expected to perform
3. dependence among the environmental conditions the system operates in

The main cause of common cause failures are design failures. Mistakes may have been made during specification, design or implementation that causes all units of a particular type to fail in the presence of a particular environmental circumstance. Design failures can occur in both hardware and software. In this case employing multiple copies of the same resource will not help tolerate the fault. A topology that employs resource diversity [84] can be selected as a means of tolerating CC/CM failures. In Section 2.4 reliability models for systems that exhibit CC/CM dependencies are introduced and used to analyse the quality of a proposed topology.

Topologies may exhibit redundancy in the form of multiple copies of the same hardware unit. This is known as *replication*. Diversity can be introduced by employing hardware units from multiple manufacturers in a topology. For instance, Boeing [43] have employed hardware diversity in the 777 project control system. Three lanes are used with each employing processors produced by independent manufacturers, in this case Intel, IBM and Texas Instruments. This approach has proved effective at detecting and tolerating physical faults.

Software Fault-Tolerance

Consider a logical architecture consisting of a hardware unit supporting a single software module that is not 100% reliable. In Section 2.2.1 the differences in the failure characteristics of software and hardware were introduced. Equivalents to hardware fault tolerance techniques can be employed to tolerate software faults. The software fault tolerance equivalents to standby sparing and N-modular redundancy are *Recovery Blocks* [71] and *N-Version Programming* [18]. A third hardware fault-tolerance technique, active dynamic redundancy, has a software equivalent in the form of N-self Checking Programming (NSCP) [4].

In the Recovery Block (RB) approach each program is divided into a primary block, an acceptance test and a set of alternate blocks. Error detection is therefore by exception test and backward recovery. Thus if the results of the primary alternative do not pass the test the secondary alternative is executed, and so on. RB can tolerate f failures by implementing $f+1$ alternatives. The RB approach is an extension of the topology

shown in Figure 2.8. Each component exhibits design diversity. The delivery of a result is suspended during error processing and a timing failure may be introduced. In the worst case all alternatives must be assumed to run. The ability of a task subjected to RB techniques to meet its timing requirements is considered in Chapter 4.

Problems arise in the RB approach when concurrent processes are implemented on distributed platforms as the ability to provide exactly the same environmental conditions for each alternative may not be guaranteed. Conversations [138] and recovery caches have been proposed to overcome this problem. The use of conversations does not affect the structure of the topology employed, but does affect timing.

N-Version Programming (NVP) is defined as the independent generation of two or more functionally equivalent programs from the same initial specifications. Each program is executed in parallel and a voter employed to identify an acceptable result. This is the software equivalent of NMR, see Figure 2.5. Wherever, possible different algorithms and programming languages should be employed in each variant. The approach taken by Boeing in the 777 project was to write a single logical variant in Ada but compile each variant using a compiler supplied by a different vendor. Thus, both replicated and diverse variants of the NMR/NVP topology can be employed.

An extensive analysis of faults in NVP is presented by Brilliant et al [18]. NVP can tolerate unanticipated errors, time overheads are negligible and the problem of CC/CM failures is reduced. However, implementation costs are very high, related failures can occur and the voter can be very complex. Processor assignment and Mean Time to Completion measures for NVP have been investigated by [94]. In this thesis a measure of the WCRT of a task or service that employs NVP is employed.

A self-checking program results from adding redundancy to a program so that it can check its own dynamic behaviour during an exception. A self-checking software topology consists of either an alternative and an acceptance test or two variants and a voter [4]. Thus NSCP can be represented by the topologies shown in Figures 2.4 or 2.5.

Joint Software-Hardware Fault Tolerance

Hardware FT techniques have typically assumed perfect software and software fault-tolerance techniques perfect hardware. This is not sufficient for a topology selection process. The ability to postulate and analyse a variety of topologies that employ both hardware and software is required. In Section 2.4.2 a unified reliability model and accompanying reliability evaluation techniques are presented. The aim being to evaluate each alternative topology with respect to its predicted combined hardware and software reliability.

The first step in such an approach is to choose an architectural component. The pieces for an architectural topology are provided by the topologies investigated in this Section and the units introduced in Section 2.2. By way of a round up of the set of possible topologies consider Figure 2.9. This is a hybrid topology consisting of a diverse TMR topology with cold spare.

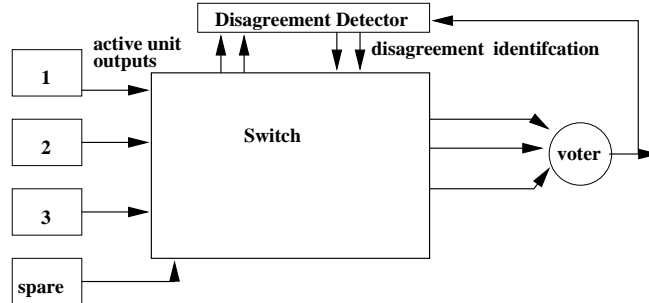


Figure 2.9: TMR plus Spare Hardware Redundancy

The three redundant components designated 1,2,3 may be three sensors or three software programs. They may be replicated or diverse units. A switch mechanism is employed to determine if one of these units has suffered a crash failure. Cold spare standby hardware units are used to replace the faulty component in the next activation of the system. A voter is used to determine an appropriate output each activation. The voter results are used by the disagreement detector to determine if one of the components is exhibiting an incorrect computation failure. This topology can guard against multiple failures and a number of different failure types. It is able to do so at a high cost in redundancy, diversity and voting.

2.4 Analysing the Quality of a Dependable System

The aim of this Section is to show that the literature facilitates prediction of the quality of a proposed topology with respect to the dependability of the services provided by a control system, $depend_s$. A number of different measures based on the characteristics discussed in Section 2.3.2 could be employed. Assuming that a SC-RT control system is not repairable during operation, reliability is the most indicative quantitative measure of the dependability of the system.

The reliability of an artefact is the probability that it can perform a required function under given conditions for a given time interval $[0,t]$. This interval could for instance represent a mission time. From a mathematical viewpoint reliability, $R(t)$, can be expressed by:

$$R(t) = \Pr\{A \text{ is fully functional in } [0,t]\}$$

where A is a unit, sub-task, task, service or system. In a continuous system $R(t)$ is usually considered to be a measure of the time to failure.

To produce a reliability measure for a range of topologies that may be selected to provide a dependable control system a Reliability Model Generator (RMG) is required. RMGs attempt to produce system level dependability models that incorporate both hardware and software reliability models and evaluate them to produce a predictive reliability measure. The elements of a RMG are:

1. A system description
2. A reliability model specification
3. Automatic generation of a combined model
4. Automatic solution of a combined model

Tools to produce elements one, two and four of a RMG have been produced by numerous authors. The difficult part appears to be the third element. The guided search approach put forward in Chapter 3 employs a library of combined system and reliability models, see Section 2.4.3. The iterative nature of the design process allows models to be updated as a design emerges. Methods exist that allow information from a model to be used in later, more complex, models. The author is aware of only two attempts to produce a full RMG [97, 173].

The NASA [97] approach is to present the designer with a Graphical User Interface (GUI). The main input categories to the GUI are requirements, architectural components and system parameters. A block diagram reliability model is then produced using this information and a reliability measure calculated.

The White RMG [173] generates all possible operating configurations of a block diagram representation of a system. These configurations are then combined into a set of states, which are evaluated to produce a reliability measure. White identifies nine distinct types of design information for his block structure RMG. Not all of this design information is required for non-repairable control systems.

The remainder of this Section is split into five sub-sections, one for each step in the production of an RMG and a sub-section for a collection of related issues. The ability to predict the relative reliabilities of a range of topologies is a pre-requisite for a topology selection process. Control systems are assumed to employ both hardware and software units.

2.4.1 System Description

A system description takes the form of a set of descriptions of the characteristics of the selected topology for a system. Two main types of system description can be employed, *combinatorial* and *state based*. To illustrate these two approaches block diagrams and Markov models are introduced. Both models will be employed as part of the selection process for the Architectural Topology problem in Chapter 3. The allocation problem will employ a system description in the form of a Directed Acyclic Graph (DAG) [8].

Block diagrams, see Figure 2.3, can be used if the assumption that system components exhibit stochastic independence is valid. In other words the failure of one hardware or software unit does not affect the operation of other hardware or software units. In a block diagram model, components are combined in series, parallel or in k-out-of-n configurations. Components can be units, sub-systems (tasks), services or systems. Units of the same type that appear more than once in a diagram are assumed to be copies.

Markovian models can be employed if the stochastic independence assumption is not valid. A stochastic process is a family of random variables $X(t)$ defined on a sample space. The values of $X(t)$ are called *states*, and the set of all possible states is the state space [152]. Thus, Markov models represent a system as a set of states and transitions between states.

Consider a Simplex Topology comprising a single task for a single control action. This can be represented as a two state Markov chain, see Figure 2.10. Obviously more complex topologies can be represented using this state based approach.

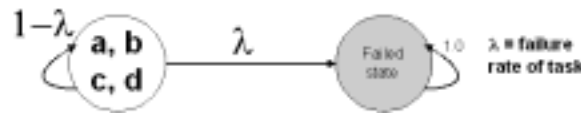


Figure 2.10: Simplex Markovian Model

The system is assumed to be operational when the task is first activated. The values of a,b,c,d represent the number of on-line units the topology ‘thinks’ it has, the number of units the topology actually has, the number of spares the topology ‘thinks’ it has and the number of spares it actually has respectively. This representation allows topologies with spares and *coverage* factors less than one to be investigated. Coverage is the probability that an artefact can automatically recover from a permanent fault (hardware or software) given that a permanent fault has occurred.

Each time a task is activated the system may undertake a state transition to the failed state with probability λ . In a non repairable system once the system has entered a failed state it cannot exit. The system may of course remain operational with a

probability of $1-\lambda$ each activation.

2.4.2 Reliability Model Specification

A plethora of reliability models have been put forward [57, 58, 144]. These studies informally discuss various model types and the kinds of dependencies that can be modelled by each type. Reliability models aim to allow *prediction* or *estimation* of the reliability of an artefact to be undertaken. Predictive techniques use parameters and developmental environment to predict dependability, while estimation techniques apply statistical techniques to observed failures during testing to forecast reliability. In fact many of the reliability models are applicable to both measures.

The reliability model employed by a designer is dependent on factors such as:

- Familiarity of the designer with the model
- Ease of use in a particular application
- Kind of system and system behaviour to be modelled
- Measure of system behaviour to be computed
- Conciseness and ease of model specification
- Availability of an appropriate modelling toolkit

Corporate best practice guidelines within which the designer works will also guide the models to be used. The use of simple reliability models implies a restricted view of the reliability characteristics of a component. Reliability predictions made over such models are likely to be somewhat naive and will not accurately predict the actual differences in reliability of different topologies.

In Chapters 3 and 4 automated tools are employed to choose between a set of alternative topologies. Thus, the *granularity* of a reliability measure is an important issue. The granularity of a measure is its ability to distinguish between two alternatives that have similar characteristics. If a measure is able to distinguish between very small changes in the characteristics of a component, in this case reliability, it is a fine grain measure. If it is only able to distinguish large changes in reliability it is a coarse grain measure. Allowance must therefore be made for the reliability model employed to predict the reliability of a topology to change as the design process progresses. Otherwise, an appropriate granularity for the selection process cannot be maintained.

Malhotra [102] has produced a formal comparative evaluation of the modelling power of various types of reliability model, see Figure 2.11. Modelling (expressive) power is defined by the kinds of system behaviour that can be modelled and measures that can

be computed. A *hierarchical composition* [45, 152] can be employed. In hierarchical composition an overall model is not generated. Instead, a set of smaller models whose solutions are combined to yield the overall model solution, are produced. This approach is used extensively in Chapter 3 and is supported by the SHARPE tool [152].

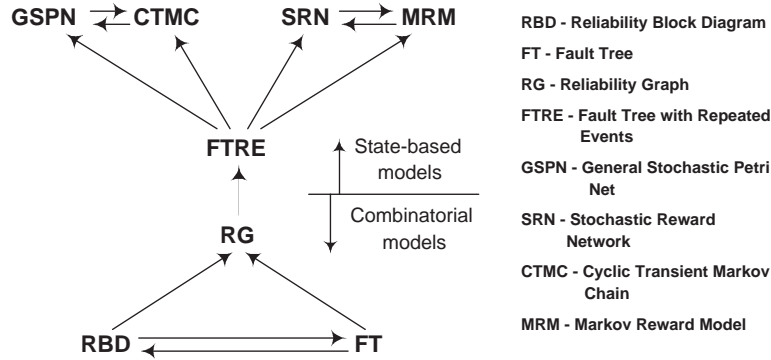


Figure 2.11: Hierarchy of Reliability Models

Among the combinatorial models Fault Trees with Repeated Events (FTREs) are the most expressive. Reliability graphs are less expressive than FTREs but more powerful than reliability block diagrams (RBDs) and Fault Trees (FT) [47] without repeated events. The construction of Malhotra’s hierarchy allows automated translation between some models to be undertaken, represented by the arrowed lines in Figure 2.11.

State-based models include Continuous Time Markov chains (CTMC) and General Stochastic Petri-Nets (GSPN) [29]. These models are only tractable under certain assumptions, such as exponential failure rates. It is well known that CTMC and GSPN are equivalent and therefore automated translations can be undertaken.

This hierarchy is a vital part of the topology selection approach presented in this thesis. In the early stages of design simple combinatorial models can be employed. As design progresses, or the complexity of the proposed topology increases, more powerful Markovian models are required. This hierarchy allows data to be re-used between models and more importantly refinement of an appropriate reliability model to mirror the system design process.

A set of combined system and reliability models have been produced to predict the reliability of an RT-SC control system for a variety of topologies. These combined models are introduced in Section 2.4.3 and summarised in Appendix A. Failure rates and failure distributions for individual units were introduced in Section 2.2.

Hardware Reliability Models

Topologies that employ only hardware units can be produced. First, consider a hardware topology consisting of a single hardware unit, i , that fails at a constant rate, λ_i . The reliability of the unit can be estimated / predicted using:

$$R_i(t) = 1 - e^{-\lambda_i t}$$

If n independent units are employed in series then the reliability of the (sub)-system can be produced in a simple combinatorial manner:

$$R_{series}(t) = R_1(t).R_2(t)...R_n(t)$$

Now suppose replication is introduced to facilitate data diversity, there are n units producing the same result in parallel, with no decision mechanism employed. The reliability of the (sub)-system is given by:

$$R_{parallel}(t) = [1 - (1 - R_1(t)).(1 - R_2(t))...(1 - R_n(t))]$$

As an example, suppose there are B buses each of which exhaustively connects a set of processors. Each bus has a constant failure rate and there is no voting on the messages arriving. The bus structures contribution to reliability [36] is the probability that at least one bus is operational at time t .

$$R_{bus}(t) = 1 - (1 - R_b(t))^B$$

A more general combinatorial reliability model for a hardware topology considers the probability that m or more out of n components are working at time t . A vote is taken on the result. The simplest approach is to consider the voter (decision mechanism) to be a separate unit executed in sequence. Reliability of a general NMR hardware topology [57, 36] can be calculated using the formula

$$\sum_{k=0}^{m-1} \binom{n}{k} (R(t))^k (1 - R(t))^{n-k}$$

The reliability of series connected components is produced when m equals n . The reliability of parallel components is produced when m equals 1. The reliability of a TMR component is produced when m equals 2 and n equals 3. Note that $R(t)$ assumes identical components in this formulation. This is not realistic if diverse spares are employed.

Standby redundancy employs cold spares. Assuming that cold spares do not fail when powered down and that switching is perfect the reliability of an m cold spare topology is:

$$\sum_{k=0}^{m-1} R(t)_k$$

A reliability model for a cold spare non-repairable topology using a CTMC approach [98] can also be produced.

If Common-cause/ Common mode (CC/CM) failures are deemed to be a problem fault tolerance techniques to reduce their likelihood or effects, and hence increase the reliability of the topology subject to such failures, are required. For instance, the CTMC model for a cold standby non-repairable topology can be updated by adding extra states to the model to show the effect of near coincident faults.

Unified Hardware and Software Reliability Models

Extra modelling complexity can be added by introducing an integrated software and hardware reliability model. Why consider hardware and software integration issues? Well, interaction of hardware and software is realistic, particularly when the designer needs to distinguish between the reliability provided by topologies using different Fault Tolerance protocols.

Markov models can be produced that represent hardware failures, software failures, and the coverage of hardware failures [24]. This is accomplished by determining a transition probability for a software failure and then incorporating the software failure into the hardware reliability model. The assumptions of Welke's [171] combined hardware and software model are:

- Software failure is described by a Goel-Okumoto Non-Homogeneous Poisson Process (NHPP)
- Time between hardware failures is an exponentially distributed random variable
- Probability of failure occurring in a period is approximated by λ
- Failures are permanent

Consider a topology which has no fault tolerance; a single software module and n hardware units of the same type and implementation. Any single hardware failure causes the system to fail. The simple Markov model for this task has just two states, operational and failed, see Figure 2.12. The probability of being in a failed state by

the end of a mission can be computed relatively easily as the sum of the hardware and software failure probabilities per unit time.

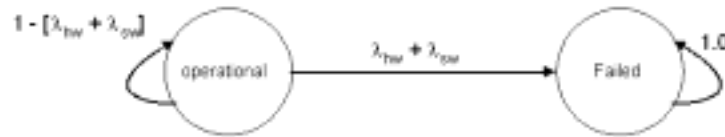


Figure 2.12: Unified Simplex Model

This model may be extended to consider a TMR topology, see Figure 2.13. Voting is via majority vote and spare processors may be used. This model allows both reliability and safety characteristics of the TMR topology to be investigated. The states of the model have the following form: Failed Unsafe (FU), Failed Safe (FS), (a,b,c,d) represents the values introduced in Section 2.4.1. That is, the number of on-line units the topology ‘thinks’ it has, the number of units the topology actually has, the number of spares the topology ‘thinks’ it has and the number of spares it actually has respectively.

In Figure 2.13 state transitions are marked with three elements, hardware failure rate (λ_{hw}), software failure rate (λ_{sw}) and the coverage factor (COV). Note that software can cause a complete failure from any state. This model also assumes that fault masking is applied so that state 3,2,0,0 is an operational state. Setting λ_{sw} to zero gives the standard hardware Markov model for a TMR topology. The reliability of the topology is given by the sum of the probabilities of being in states FS or FU by the end of the mission.

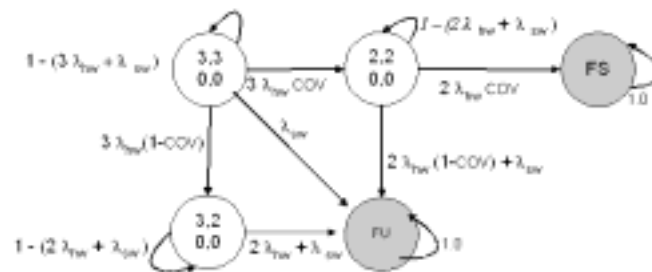


Figure 2.13: Unified Model for a TMR Hardware/Software System

The two models shown above could have been considered either as hardware tasks or software tasks alone. However, if diverse standby sparing is employed all the interactions cannot be modelled accurately by considering hardware and software separately. For instance, the software in the on-line hardware component may be different from that in the spare component. Welke [172] has produced a model similar to Figure 2.13 for such topologies.

Issues remain in reliability modelling. Trivedi & Geist [58] have produced a list of current research areas including:

- integration of system design and evaluation
- integration of measurement data and modelling
- integrated evaluation of hardware and software, and
- multilevel, hierarchical model decomposition.

All of these issues are addressed to some degree within the topology selection approach presented in this thesis. The aim is to use existing techniques in a more constructive manner. It is noted that theoretical shortcomings limit the granularity and expressive power of the models available. Work on incorporating performance measures within a reliability modelling framework, such as Performability, and modelling common cause failures, such as joint reliability importance [5] indicate that existing models are open to improvement. Research into Markov Reward Models also holds out the chance of being able to consider risk as part of a topology selection approach [119].

2.4.3 Generation of a Combined Model

There is little tool support for combining a system description and reliability model. This process remains very much in the expert domain. Therefore, the approach employed in this thesis uses a set of libraries.

Three libraries are required, a reliability model library, a hardware library and a software library. The contents of these libraries are considered in detail in Appendix A. Here an overview of the features of the reliability model library is presented. The hardware and software libraries will include failure rate data for individual hardware and software units, see Section 2.2.

A reliability model library contains a set of models for the different topologies to be investigated as part of a topology selection process. These models can initially be imported from similar projects or the designer could produce a set of simple reliability models. As design progresses and hence knowledge about the system increases, the reliability models employed will need to develop.

Results of testing of completed (sub)-systems or more complex modelling can be used to upgrade the reliability models employed. Automatic translation algorithms produced by Malhotra ensure that minimum information is lost as the modelling exercise progresses. The hierarchy of reliability models allows appropriate models to be employed throughout the design process.

2.4.4 Automatic Solution of a Combined Model

A number of powerful general purpose solution engines have been produced. These tools take a specification of a topology model and allow a set of dependability measures to be produced. In this Section six automatic evaluation tools are introduced, each of which has features of interest to the designer selecting a Topology. The choice of computing engine is partly determined by the required expressive power of the topology reliability model employed by the model specification part of the RMG. The main comparative work on reliability tools is by Trivedi & Geist [58].

HARP (Hybrid Automated Reliability Predictor) [44, 11] provides a general Markov modelling capability. The main features of HARP are:

- Model specification is via dynamic FTREs [46]
- Sequence dependent failures are modelled as dynamic FTREs and evaluated using Markovian techniques.
- Hot and cold spare systems can be evaluated.
- Produces guaranteed parametric bounds on system reliability
- Solves truncated models for very large systems

The NASA RMG employs an evaluation tool called SURE (Semi-Markov Unreliability Range Estimator) [24]. This tool produces reliability measures for fault-tolerant systems that have slow fault-arrival processes and fast system recovery processes. It employs a semi-Markov technique and a path pruning algorithm to reduce the computational complexity of the specified reliability model.

SURE has a number of features that make it attractive for large complex systems. The use of paths and the production of reliability bounds make it relatively fast, an important feature for a tool employed as part of a topology selection strategy. SURE's ability to deal with hierarchical models is however somewhat limited and Semi-Markov models have not proved attractive in industrial practice.

SPNP (Stochastic Petri-Net Package) [29] is a tool that evaluates General Stochastic Petri-Nets. This tool can evaluate the complex system models required late in the design process. Petri-nets are only required for detailed analysis.

UltraSAN [153] allows model-based evaluation of systems represented as Stochastic Activity Networks. A GUI allows easy specification of models in an hierarchical fashion. Reward structures can be used to define performance, dependence and performability measures. Analytic models are solved using Markovian techniques. Discrete event simulations can be used to obtain transient results.

As well as the ability to perform evaluation of instantaneous dependability models UltraSAN also allows timed activities to be explicitly modelled. This tool may be used as a link between the dependability and timing characteristics of a system.

SDAT (System Dependability Assessment Tool) [17] automatically generates mathematical dependability models based on simple parametric representations of system component characteristics, maintenance and logistics capabilities, sub-system redundancy structures and critical mission functionality. Specification in SDAT is oriented towards capturing design information that is relevant to system reliability model analysis (RMA). The design elements that can be defined by the user are shown in Figure 2.14:

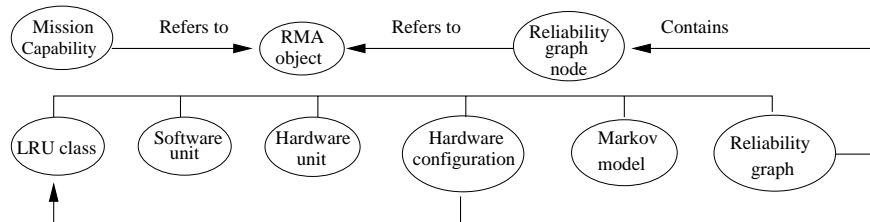


Figure 2.14: SDAT Dependability Model Object Types

Physical hardware units are modelled at the level of *line replaceable units* (LRUs) that are replaced when a failure occurs during system operation. This tool has features employed in Chapter 3.

Finally, SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator). SHARPE [152] provides a specification language and solution methods for the following model types:

Combinatorial Models	State Based Models
Series-parallel block diagrams	Acyclic Markov Chains
Fault Trees	Cyclic Markov chains with absorbing states
Reliability Graphs	Irreducible Cyclic Markov chains
Series-parallel directed graphs	Acyclic semi-Markov chains
Product-form Queueing Networks	Irreducible Cyclic semi-Markov Chains
	Generalised Stochastic Petri-nets

Input/Output distribution functions are exponential polynomials that are symbolic in time t . A hierarchical combination of different model types is used to derive overall models for a system. SHARPE was chosen as the solution engine for the reliability models employed in this thesis for a number of reasons including:

- Ability to specify a set of generic models that can be instantiated with different values. For instance, the generic RBD model for a parallel topology can be used

if two copies of processors of type 1 are used or if two processors of type 2 are used. The failure rate of each processor determines the reliability of the parallel sub-system.

- Text based input/output makes the integration of SHARPE into a topology selection tool relatively easy.
- All main model types can be evaluated
- SHARPE is inherently hierarchical because of the nature of the exponential family of distribution functions employed by the tool.
- SHARPE was available to the author.

A library of SHARPE reliability model specifications has been produced, see Appendix A. This library forms an important element in the topology selection process.

Extension to RMGs

Attempts have been made to extend the use of dependability measures of a proposed topology into the design process. For instance, Tillman [161] considered N-stage parallel-series systems and attempted to optimise the reliability of the system subject to constraints such as cost, and weight. This paper employed a combinatorial model and dynamic programming.

More recent work has concentrated on optimisation using guided search techniques. Coit & Smith [31] produce a problem-specific GA to:

“ analyse *series-parallel* systems and to determine the optimal configuration when there are multiple component choices available for each of several k-out-of-n subsystems.”

They call this configuration problem the *redundancy allocation problem* for a series-parallel system.

Painton [125] presents an optimisation model to:

“ ... identify the types of component improvements and the level of effort spent on those improvements to maximise one or more performance measures subject to constraints.”

Painton employs a GA and concludes that the integration of GA capabilities with reliability analysis software can provide a robust and powerful design-for-reliability tool.

Results from these studies are promising. These papers along with previous work by the author have informed the decision to use guided search techniques to aid the designer resolve the Topology issue. In particular, a GA will be used to aid resolution of the Architectural Topology Problem. This GA improves on the existing techniques by providing a much larger set of system topologies and explicit assignment of resources to functional elements of a topology. An interface with the allocation problem is provided.

2.5 Designing Hard Real-Time Systems

A second non-functional attribute of a distributed control system must be addressed when selecting a topology: timing. The need to produce a model for the real-time characteristics of an application stems from the need to ensure efficient resource sharing and to meet Worst-Case Response Time (WCRT) requirements. The timing characteristics of a topology are determined by four factors:

1. Computational model
2. Allocation of functional units (software units and messages) to hardware units
3. Worst-Case Execution Time (WCET) of functional units when allocated to a particular hardware unit.
4. Worst-Case Response Time of tasks, in the topology.

The interface between software and hardware is the *computational model*. In this Section a simple computational model along the lines discussed by Burns [21] is presented. The model is compatible with the work on dependability introduced above and includes sub-tasks, messages, transactions, a communication model, and a scheduling model. The assumptions made in this model are also assumed in the *X-Alloc* tool.

The allocation problem is considered in detail in Chapter 4. In Section 2.6 techniques to calculate the WCETs of units and WCRTs of transactions are introduced. Transactions are sequences of precedence ordered sub-tasks that form paths through a system. That is they take data from the environment, process it in some way, and produce an output to the environment.

Consider a system architecture that employs Preemptive Priority Based Scheduling (PPBS). It employs a fixed set of processors onto which a fixed number of processing sub-tasks (software units) are placed. Each sub-task has a unique priority which determines the order in which it is processed. That is, if two sub-tasks want to use the same hardware resource at the same time the unit with the highest priority is executed first. Scheduling analysis produces a guaranteed WCRT for each software unit, transaction,

and hence processing task, in the system. Priorities are set so that timing requirements are met. Thus the priority of each software unit is a degree of design freedom.

Restrictions are placed on the number of sub-tasks that may reside on any single processor. Furthermore, each sub-task resides on exactly one processor and all sub-tasks may be modelled as *periodic* events. That is, events released at regular intervals. All sub-tasks in a single transaction have the same period. The usefulness of these restrictions will be assessed in Section 2.6 and Chapter 4.

In Hard Real-Time (HRT) systems each transaction has a WCRT requirement placed on it, which is expressed as a *deadline*. A deadline is the maximum time allowed from the expected transaction release until execution is completed. In Figure 2.15 a transaction with three precedence ordered periodic sub-tasks is presented. There is a flow of data, but no flow of control, between these sub-tasks. Sub-tasks are *offset* in time. That is, sub-task 2 cannot be processed before sub-task 1 has completed, as it requires data from that sub-task.

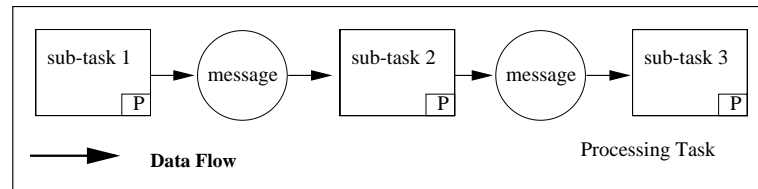


Figure 2.15: Periodic Transaction

Each sub-task is assumed to be a permanent element of the system. It is placed on the run queue by an invocation event. It gains access to the processor it resides on when it is the highest priority sub-task wishing to run. Each processing sub-task has a potentially infinite number of such invocation requests. A cold spare is modelled as a hot spare with the WCET of the sub-task adjusted to take into account the ‘warm-up’ time of a cold spare.

Extensions to this simple model are possible. Sporadic sub-tasks are released by an originating event from either another sub-task or the environment of the processing task. Sporadic sub-tasks arrive at irregular intervals, but with a minimum inter-arrival time between triggering events. Bate [10] shows that sporadic sub-tasks can be analysed as if they were periodic with offset in order to predict their WCRTs. Mixed sporadic and periodic transactions are often used in control applications to reduce *jitter*. Jitter is the allowed variation of sub-task completion from precise periodicity.

Thus the elements of the computational model used in this thesis are:

- Entities: Periodic or sporadic software units (active objects) and messages (passive objects).

- Communication via shared data areas. These areas may be implemented using buses or shared memory units.
- Precedence relationships and priority orderings define the order in which sub-tasks run.
- Entities are allocated to a single processor or shared data area.
- Transaction deadlines capture end-to-end timing requirements.
- Pre-emptive Priority Based scheduling.

2.6 Analysing the Quality of an HRT System

The quality of an HRT system is partly determined by how quickly it can perform a required set of actions. The system must produce results within a maximum response time. Off-line analysis techniques can help predict whether a proposed topology for a system will allow these requirements to be met. The results of this analysis are employed in the evaluation function for the GTP, $WCRT_s$.

Software units consist of a number of instructions that must be executed by a processing hardware unit. The time required to execute these instructions typically varies between invocations of the unit. The execution time of a unit may also change if the type of processing hardware (resource) the unit is assigned to changes.

Messages also take a non zero time to complete transmission. In the case of a bus communication system the time for a message to be sent between sub-tasks on different processors is the time for a message to be encoded, transmitted down the bus and decoded at the other end. In this thesis it is assumed that the coding and decoding of messages is factored in to the WCET of the sending and receiving sub-tasks. The worst case communication time (WCCT) of a message is therefore a function of the number of bytes to be transmitted and the speed of the bus. Intra-processor communication is thus assumed to take zero time.

The central goal of a timing analysis system is to determine a **safe** and **tight** bound on an active object's execution time by a purely static analysis of the program's source code, object code or both [26]. The analysis is static since it does not involve running the program. This is not currently possible even with a state of the art static code analyser. Thus bounds are placed on the execution time of a piece of code.

A safe result is one where the predicted WCET is strictly larger than or equal to the actual WCET of an object. A tight result is safe, but also within a reasonable margin of the actual WCET. The conservatism of the predicted result is often known as the **pessimism** of the predicted WCET. An overly pessimistic result will lead a designer

to believe that a system does not have sufficient capacity to run a given set of software units when, in fact, it does.

A number of tools have been produced to determine the WCET of active objects. Chapman et al [26] present an excellent survey which includes work by Park & Shaw [126], the MARS group [86] and York [6]. The designer of any timing analysis system faces several problems that are independent of programming language and target hardware, but are beyond the scope of this thesis [26]. Assuming that these problems may be addressed the designer of a topology may assume that WCET data is available. In the early stages of the design process ‘time-budgets’ in the form of maximum instructions per invocation, are employed. As coding progresses WCET bounds based on program code analysis are produced.

One important factor for this thesis is the effect of placing software units onto different hardware resources. How is the WCET of an active object affected by being placed on a 486, rather than a 68040 processor, for instance? Traditionally, CPU manufacturers have been unwilling to release detailed information about their products, so worst case timing analysis remains a problem [117].

Assume that a safe overall worst case number of instructions independent of processor allocation for a software unit, can be produced. Thus the effect of assigning a sub-task to a different processing resource on the WCET of a sub-task is purely a function of the relative speeds of the processors. The WCET of a software unit is given by

$$\frac{\text{fixed worst case number of instructions}}{\text{speed of processing resource in instructions per unit time}}$$

Similarly, the WCCT of a message is given by

$$\frac{\text{fixed worst case number of bits}}{\text{speed of bus resource in bits per unit time}}$$

This assumption is pessimistic and may lead to rejection of valid allocations. If data is available a table of WCETs for a software unit can be produced for the set of admissible processing resources. Once an architectural topology has been selected a much tighter bound on the WCET of a sub-task can be produced, as detailed analysis of the chosen software / hardware interface may be undertaken. Manipulation by the designer of the interface between the architectural topology and allocation problems is therefore essential, see Chapter 5.

Worst Case Response Time

If each sub-task is placed on a dedicated processor and each message on a dedicated bus the WCRT of a transaction is calculated by simply summing the WCET of each software unit and WCCT of each message in the transaction. This is unrealistic for industrial systems. Typically, more than one sub-task is placed on each processor and more than one message per bus. In this Section calculation methods for determining the WCRTs of sub-tasks/messages, distributed transactions and processing tasks are presented. The exposition is based on the work undertaken in the Real-Time Systems Group [6] and Rolls-Royce UTC [10] at the University of York.

Assume that an assignment of sub-tasks to processors and messages to buses has taken place. Tindell et al [163] introduce an holistic approach to the prediction of WCRTs for bus based systems. A system is assumed to be implemented as a set of transactions, with the first sub-task being periodic, and subsequent sub-tasks/messages being sporadic. Sporadic sub-tasks are triggered by the arrival of the preceding sub-task/message. For modelling purposes sporadics are given a fixed minimum inter-arrival time.

Timing analysis is performed for each processor in the selected hardware platform. Variability in the arrival time of consecutive invocations of a sporadic sub-task is accounted for by a factor known as the *release jitter*, J . The WCRT of sub-tasks allocated to a processor is predicted using equation (2.1) taken from [162]. This equation assumes all sub-tasks have a fixed priority, zero offset and deadlines are not greater than period.

$$R_i = C_i + B_i + I_i \quad (2.1)$$

where i ranges over the sub-tasks for a given processor, R_i is the WCRT of sub-task i , C_i is the WCET of sub-task i , and B_i is the blocking time of sub-task i .

The interference a sub-task suffers is the total computation time higher priority software units can generate in any arbitrary time interval. Interference is calculated using equation (2.2).

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i + J_j}{T_j} \right\rceil C_j \quad (2.2)$$

where J_j is the release jitter of sub-task j and $hp(i)$ the set of higher priority sub-tasks than sub-task i .

The blocking time a sub-task suffers is the total computation time a lower priority sub-task can stop the sub-task from executing and is calculated using equation (2.3).

$$B_i = \max_{k \in lp(i)}(C_k) \quad (2.3)$$

where $lp(i)$ is the set of lower priority sub-tasks than sub-task i .

Equation (2.1) is solved by forming a recurrence equation as shown in equation (2.4).

$$R_i^{n+1} = C_i + \max_{k \in lp(i)}(C_k) + \sum_{j \in hp(i)} \left\lceil \frac{R_i^n + J_j}{T_j} \right\rceil C_j \quad (2.4)$$

which terminates when $R_i^{n+1} = R_i^n$ or $R_i^{n+1} > D_i$. D_i is the deadline of sub-task i . When the analysis converges the WCRT is expressed as:

$$R_i = R_i^{n+1} + J_j \quad (2.5)$$

These equations can be used to perform timing analysis of the data bus, with WCET being interchangeable with WCCT and i denoting a message.

Analysis of this approach by Bate & Burns [10] has focused on convergence. Convergence is defined to have occurred when $R_i^{n+1} \geq R_i^n$ for all sub-tasks and $R_m^{n+1} \geq R_m^n$ for all messages. Bate presents a revised schedulability analysis, which is designed to overcome the perceived convergence problems with the analysis presented in equations (2.1) to (2.5).

In the revised approach all sub-tasks are modelled as periodic events. Offsets are employed to enforce the correct precedence ordering in transactions. The offset of a sub-task is set such that it is always greater than the WCRT of the event that triggered it. A change in the offset of a sub-task on one processor may lead to a change in the response time of a sub-task on another processor. Convergence occurs when no further increase in response time occurs for any sub-task, or message, during an iteration of the schedulability algorithm. This new analysis is applied in Chapter 4.

The existing uni-processor timing analysis is extended to account for offsets. A composite sub-task with zero offset is used to represent sub-tasks with non-zero offsets. In this case equation (2.6) may be used directly.

$$R_i = C_i \max_{k \in lp(i)}(C_k) + \sum_{j \in hp(i)} \left\lceil \frac{R_i - C_i + \delta}{T_j} \right\rceil C_j \quad (2.6)$$

where i is a sub-task, $T_i(T_j)$ is the period of sub-task $i(j)$, δ is one clock cycle, $hp(i)$ is the set of higher priority sub-tasks than sub-task i and $lp(i)$ is the set of lower priority sub-tasks than sub-task i .

Equation (2.6) is a reworking of equation (2.4) with the jitter factor set to 0. The algorithm terminates when $R_i^{n+1} \geq R_i^n$ for all sub-tasks and $R_m^{n+1} \geq R_m^n$ for all messages.

On termination the WCRT of sub-tasks and messages can be extracted directly. If some, or all, sub-tasks or messages, fail to meet their deadline this can also be determined. The WCRT of a transaction is given by the WCRT of the last sub-task in the transaction. The WCRT of a processing task can be determined by using the formula

$$WCRT_t = \max(WCRT_{tr} * Tr_{t,tr}) \text{ for all } tr \in t.$$

where $Tr_{t,tr} = 1$ if transaction tr forms part of processing task t and 0 otherwise.

Given a complete allocation of sub-tasks and messages the schedulability of each processing task can be predicted, along with the WCRT of schedulable tasks. This analysis has been extended by the author to investigate shared memory systems [120].

2.7 Search Techniques for Combinatorial Optimisation

In Figure 2.16, based on work by Ribeiro et al [50], three classes of search technique that may be employed to solve optimisation problems are shown. *Calculus* based techniques employ a set of necessary and sufficient conditions that have to be satisfied. *Enumerative* techniques investigate every point in the search space, they are exhaustive. *Implicit enumerative* techniques are based on enumerative methods, but use extra knowledge to guide the search process.

The Topology Problem may be formulated as an NP-complete decision problem (SAT) or an NP-hard combinatorial optimisation problem [56, 27] (ATP), see Section 2.1. The important features of an NP-hard optimisation problem, with respect to producing a solution for non-trivial sized problems, are:

1. checking whether a proposed solution is “acceptable” with respect to a given set of criteria can be undertaken in polynomial time.
2. finding the optimal solution, with respect to a given set of criteria, is non-polynomial (NP). That is, calculus and enumerative searches cannot in general be guaranteed to find an optimal solution in polynomial time.

This implies that an implicit enumerative (guided) search, or some other form of problem specific heuristic, is required in order to find good solutions within a ‘usable’ time period. In this thesis elements of a variety of guided search heuristics are used to find acceptable solutions to the topology selection problem. All these guided search techniques fit within a single unified search paradigm, see Section 2.7.1. A GA has

been chosen to address the Architectural Topology problem, see Section 2.8. A second technique, Simulated Annealing has been chosen to address the Allocation Problem, see Section 2.10.

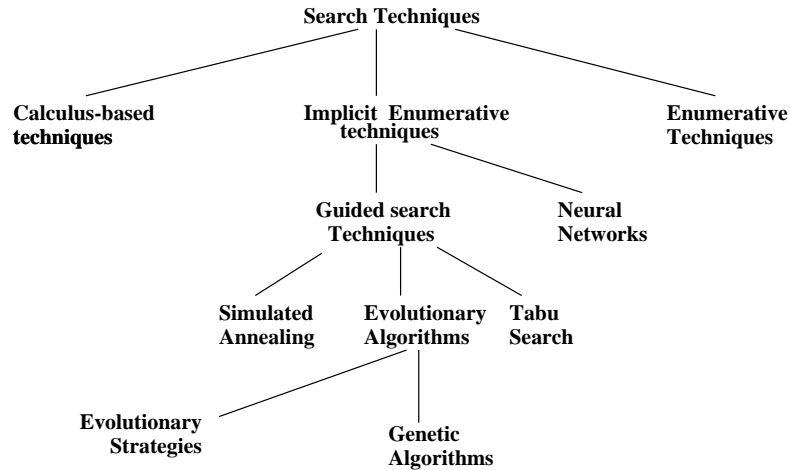


Figure 2.16: Search Techniques for Optimisation

2.7.1 A General Search Paradigm

Consider the *neighbourhood* of a solution to a combinatorial optimisation problem. A neighbourhood $N(s, \sigma)$ of a solution s , is a set of solutions, S' , that can be reached from s by a simple operator, σ . These operators are often called *moves*. If a solution $\bar{s} \in S'$ is better than any other solution in its neighbourhood $N(\bar{s}, \sigma)$, then \bar{s} is a local optimum with respect to this neighbourhood. In some cases it is possible to find a move σ such that a local optimum is also a *global* optimum, s^* . A *unified* approach to the resolution of combinatorial optimisation problems through the use of guided neighbourhood search techniques has been postulated by Rayward-Smith [139], see Figure 2.17.

The pool of solutions, P , contains at least one potential solution drawn from a set of admissible solutions, S . In some cases, such as simple GAs, S may contain multiple copies of a given solution forming a multiset. In other cases S may contain only a single copy of any particular solution, forming a set. Q contains a selection of solutions from P that are manipulated to create a (multi-)set of new solutions, R . Finally, (multi-)sets P , Q and R are merged to produce a new pool of solutions each iteration.

To instantiate this general paradigm as a guided neighbourhood search each solution is associated with a set of neighbours which can be reached directly by an operation, or set of operations. Thus in the general search paradigm *Initialise* selects some starting set of solutions and assigns them to P . The *Select* function picks members of the population to form Q . *Create* applies a set of neighbourhood operations to the members of Q to

```

P, Q, R: (multi-)set of solutions  $\in S$ ;
Initialise (P);
while not finish(P) do
  begin
    Q:= select(P)
    R:= create(Q)
    P:= merge(P,Q,R)
  end
end

```

Figure 2.17: General Search Paradigm

produce a set of new solutions, R , in the neighbourhood of P . *Merge* selects the subset of the solutions in $S \in (R \cup P)$ with the lowest evaluation function values. *Finish* ensures termination, either after a given number of iterations, or when a stopping criterion is met.

To implement a guided neighbourhood search, the heuristic designer must decide:

1. how solutions are to be represented, i.e. specification of S ,
2. the cardinality of P and starting set $P_0 \subset S$,
3. the set of totally ordered values (C, \geq) and the evaluation function: $S \rightarrow C(S)$,
4. what constitutes a “neighbour” (i.e. the definition of *neighbourhood*),
5. the definition of the *select* function used to pick an appropriate subset of neighbours,
6. the defining properties of the subset $R \subset Q$, constructed by *create*,
7. the definition of *merge*, which determines the starting point for the next iteration, and
8. the termination criteria (i.e. the definition of *finish*).

The simplest form of neighbourhood search has only one solution in P . The algorithm moves from this solution of value $c(s)$, to a neighbouring solution of value $c(s')$, only when $c(s) > c(s')$. The algorithm terminates when there is no neighbour with a smaller value, in a minimisation formulation. This is **Hill Climbing**, or more appropriately *Hill Descending*. The main problems of this approach are that it tends to find a local, rather than global, optimal solution and the number of solutions to be evaluated can be very large. More complicated neighbourhood searches can escape from local optima.

The unified search approach allows a suite of tools to be produced to aid resolution of the topology selection problem in which the designer's experience can be harnessed, while allowing analysis to be undertaken throughout the system development process. It provides a framework to produce heuristics to tackle both the Architectural Topology and Allocation problems.

2.7.2 The Beauty Contest!

In this section a comparison is made between the search techniques employed in this thesis. Heuristics considered include Genetic algorithms, tabu search and simulated annealing. In many ways such a comparison resembles that of a *beauty contest* in that the criteria by which the decision is made are not very well defined and the results are often highly subjective, with different groups of judges each having their own preferred candidate. Thus, problems arise in making comparisons between heuristics, mainly because of the amount of domain knowledge being used to produce the algorithm. Heuristics that have domain knowledge applied to them will generally produce better results than the 'Vanilla Flavoured' (standard software) versions of other techniques.

The performance of one heuristic, relative to another, can be measured against two main criteria: the quality of the final solution produced and the computation time required to converge to good solutions. As in many aspects of the class of heuristics considered in this thesis a trade-off between these two criteria is usually required.

The choice of a heuristic for a given problem is not easy either. Factors that should be considered include; *problem size, problem structure, cost of using a sub-optimal solution, data relating to performance on similar problems, and data relating to computational cost on similar problems*. In general no one technique will allow a definitive decision to be made as to which heuristic is most appropriate. For many problems it may be that different elements of the problem need to be solved using different, but related techniques.

In the early stages of this DPhil project Tabu Search (TS) was seriously considered as an appropriate search mechanism for the general topology problem. It became apparent however that sufficient data on the set of alternatives and knowledge of the set of appropriate tabu restrictions were not available given the current state of industrial practice. However, TS has a number of elements that may well add considerably to the quality of tools to aid resolution of the topology problem. TS hybrids are considered in Section 2.9, Chapter 3 and Chapter 4.

The topology problem is addressed as two inter-related problems in this thesis. Why wasn't a single formulation produced and a single search engine employed? First, there is insufficient data available during some phases of the design process to address allocation along with the architectural topology. Second, previous experience with

GAs has shown that they are not good optimisation techniques [63, 54].

The split is also made on philosophical grounds. Human intervention is vital in a tool produced as a design aid. It is not clear that a fully automatic tool to resolve the topology problem is plausible in the near future. It must be stressed that the aim of this thesis is to produce an ability to search the design space rapidly in an attempt to indicate *good* topologies that can be assessed by the designer. Using two searches allows the user to change the moves allowed and the parameter settings of the tools. What is effective at one stage in the process is unlikely to remain effective throughout the design process.

The architectural topology problem is computationally very expensive because evaluation of each alternative can take a long time. State-based reliability models in particular are prone to very long computation times. However, it is likely that computationally expensive alternatives will employ many spares and decision mechanisms making them large and costly. Thus, the search is likely to reject such alternatives early in the search process. GAs have been employed on control problems for which each evaluation may take several hours [127]. In these cases GAs have been modified to converge to *acceptable* results in a very small number of generations.

GAs are considered to be robust, but weak optimisation methods [63]. The robustness of GAs comes from the ability to work on extremely *fuzzy* problems, in which little information is available to help traditional methods [51]. The ability of GAs to work on such problems is one reason why they are used to aid resolution of the Architectural Topology problem in Chapter 3. In the early stages of the design process the available information about a proposed system is limited, and the issues being addressed are often somewhat fuzzy in nature. GAs have previously been employed to address simple variants of the architectural topology problem, see Section 3.3.

Previous work by the author [117] has shown that SA performs poorly when:

- the search space is *ill-formed*. This may be due to solutions that lie close to local optima being given high penalty values. This may also occur if the evaluation function is sensitive to small changes in the values of some parameters.
- moving from a poor solution to a good solution requires multiple moves, some of which reduce the quality of the solution
- easily *side-tracked*. This may occur if a number of parameters are heavily inter-related

GAs can overcome these problems at the price of a tendency to find a non-optimal solution.

The tool to aid resolution of the allocation problem in Chapter 4 employs an SA algorithm. There are a number of reasons why this change in search approach was made. The first two are the flip side of the reasons why GAs are employed in Chapter 3 namely, lack of data and quality of optimisation. The allocation problem is essentially a well defined optimisation problem. The architectural topology and set of sub-tasks to be placed on the hardware platform have been chosen. The exact number of each hardware resource is not fixed. This reduces some of the problems found with the search space in previous attempts to use SA for allocation [117].

A third reason for the change is that the author has previous experience with SA for the allocation problem and can therefore employ a non-*vanilla flavoured* algorithm. Finally, since the environments for GA and SA are compatible, it is easy to move from one to the other. This compatibility means that users can attempt to employ a GA for the allocation problem if they desire.

In conclusion, a GA is used for the architectural topology problem because of the fuzzy nature of this problem and the computational complexity of function evaluation. SA is used for the allocation problem because it is a better optimisation technique. The decision to split the problem was made on computational and user involvement grounds.

2.8 Genetic Algorithms

A GA is a guided search technique, based on models of Darwinian [165] and Lamarkian [73] evolution. Solutions are represented by fixed length strings over some alphabet. The value of a solution is a measure of its “fitness for purpose”. In terms of the algorithm presented in Figure 2.17 the general search paradigm becomes a GA search when:

- (1) P is a multiset and contains more than one element.
- (2) The *create* function employs genetic operators to generate new solutions.
- (3) Both *select* and *merge* functions depend on the evaluation of solutions and together favour the preservation of higher *value* solutions.

A number of overviews on GAs have been produced, including work by Whitely [174] and Beasley et al [12] and the following review draws heavily from these papers. A number of bibliographies of GA references have also been produced [1, 62].

Representation

GAs use a direct analogy of natural behaviour. The salient features of each individual population member are represented by a string, usually of fixed length and referred to

as a *chromosome*¹ or a *genotype*. The components of the strings are often called *genes* and the possible values they can take *alleles*. The chromosome is decoded to form the *phenotype* and this is the incarnation that is tested for fitness.

In GAs it is often assumed that System Design Factors (SDF) can be represented by bit strings. Genes that represent a parameter are discretised to take on values that correspond to a power of 2. A 10 bit gene could therefore take on values between 0 and 1023. Problems arise if a parameter has an exact finite set of values that is not a power of two, as a number of unnecessary bit values may occur. This can be tackled by using default worse possible evaluations, representing some parameter settings twice, or other similar techniques.

Non-binary codings can be used. Higher cardinality alphabets such as integers and non-negative reals have been considered. One advantage of using such codings is that it is easier to define meaningful, problem-specific genetic operators. In this thesis both binary and integer coding is employed.

Initialisation

In simple GAs the initial population is often created by randomly generating the values of genes and subsequently chromosomes. Domain knowledge can be used to perform heuristic initialisation so that the search begins with some reasonably good population members.

The appropriate population size and mix is determined during initialisation using existing population sizing theory [61]. The bounds produced indicate the size to which a population can be increased before convergence may no longer occur.

Select

The *Select* function determines a number of “fit” solutions which form the “mating pool”, Q . In order to produce this (multi-)set of solutions an ability to determine which solutions are “fit” and to select from these fit solutions is required.

In GAs a *fitness (evaluation) function* assigns a score to each individual in a population that indicates the “quality” of the solution the individual represents. The fitness function is often given as part of the problem description². In combinatorial optimisation problems, most points in the search space are not feasible. A fitness function is produced where the fitness of an invalid chromosome is viewed in terms of how good it is at leading the search *towards* valid chromosomes.

¹Chromosome stands for “coloured body” after the colouring of nuclei in early experiments to identify DNA.

²Multi-Criterion Decision Analysis (MCDA) can be employed to determine a fitness function [83].

Once a population has been generated a mating pool is produced. In simple GAs the probability that a chromosome in the current population is copied into the mating pool is proportional to its' fitness. There are many ways of accomplishing this selection:

- (1) *Roulette wheel sampling*: Each individual in the population is represented by a space proportional to its fitness. By repeatedly spinning the wheel, individuals are chosen using random sampling with replacement.
- (2) *Implicit fitness remapping*: The most common variant is *K-tournament* selection where K individuals are drawn from a population with replacement. The most fit individual of the K selected is copied into Q. The process is repeated until Q is full.
- (3) *Exponential sampling*: Selects strings at random using an exponential distribution random number generator so as to favour the best solutions in P.
- (4) *Steady state (proportional) replacement*: The maximum proportion of individual chromosomes to be replaced is limited. For example, GENITOR [175] selects parents according to ranked fitness scores, and the offspring replace the two worst members of the population.

The tournament approach can be applied in both the *select* and *merge* functions. At the selection stage a single acceptance/rejection tournament is often held. By adjusting the tournament size or win probability the tournament can be made arbitrarily large or small. Other tournaments can be held, such as double acceptance/rejection (both parents compete against both offspring) tournaments.

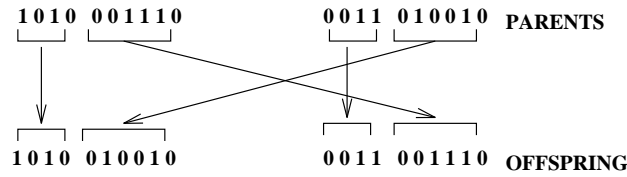
Create

The *create* function applies genetic operators to the mating pool, Q, to generate a new set of chromosomes, called *offspring*. The operators used can be categorised by whether they *intensify* or *diversify* the search space being considered. The standard intensification operator is *crossover* and the most common diversification operator is *mutation*.

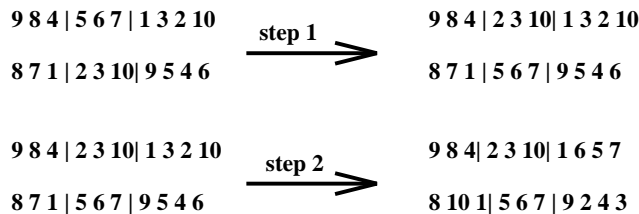
Crossover takes two individuals and cuts their chromosome strings at some randomly chosen position(s). The substrings produced are then swapped to produce two new full length chromosomes. Crossover techniques include:

- (1) *Single point crossover*: A single random cut is made, producing two “head” sections and two “tail” sections. The two tail sections are then swapped to produce

two new chromosomes:



- (2) *Two point crossover*: Chromosomes are regarded as loops formed by joining the ends together. Segments of these loops are exchanged to form offspring. Researchers generally agree that 2-point crossover is better than 1-point because it samples uniformly across the full length of a chromosome [141].
- (3) *Uniform Crossover*: Each gene in the offspring is created by copying the corresponding gene from one or other of the parents. The selection of the donor parent is undertaken via a randomly generated crossover mask. Where there is a 1 in the mask the offspring gene is taken from the first parent; where there is a zero, the gene is taken from the second parent.
- (4) *Partially Mapped Crossover (PMX)*: This operator has been proposed for permutation problems to allow sequences in the parents to be maintained. Two crossover points are picked uniformly at random. The substrings within the marked points are exchanged. Then, multiple copies of alleles outside the substring are eliminated by substitution:



Domain knowledge can be used to design *local improvement operators* [142] which allow more efficient exploration of the search space around good solutions. For instance, knowledge could be used to determine the appropriate sites for crossover points.

Diversification schemes allow jumps to different areas of the search space. The basic mutation operator for binary coded problems is *bitwise mutation*. Mutation occurs randomly and very rarely with a probability p_m (per bit). Typically the mutation rate is less than one percent. In some cases mutation is interpreted as generating a new bit and in others it is interpreted as flipping the bit. In high order alphabets mutation takes the form of replacing an allele with a randomly chosen value, in the appropriate range, with probability P_m .

Merge

The *merge* function is used to create the next generation of chromosomes. It thus combines the old population, P, mating pool, Q, and new population generated by intensification and diversification operators, R. Possible merge mechanisms include:

- (1) *Replace All*: New P is simply R.
- (2) *Best fit*: Best solution in R is used to replace worst solution in P until either R is empty or all solutions in P are better than those left in R.
- (3) *Tendency*: Same as best fit, except that all chromosomes in R that exist in P are discarded beforehand.
- (4) *Tournament*: Competitions are run between sets of strings from R and P, with the winners becoming part of the new population [100].

2.8.1 Hard Problems for GAs

There are a number of problem characteristics that make it difficult for GAs [54] to find a “good” solution including **epistasis**, **genetic drift**, **deception**, and **premature convergence**.

Many problems that GAs are used to tackle are non-linear in nature, which often indicates that parameters cannot be assumed to be independent. That is, the contribution of a gene to the fitness of a chromosome depends on the value of other genes in the chromosome. Three levels of interaction between genes are put forward by Beasley et al [13]. Level zero indicates no interaction. Level one indicates mild interaction where a particular change in one gene always produces a change in fitness of the same sign. Level two indicates *epistasis* [14] where a change in a gene causes a change in fitness that varies in sign and magnitude, depending on the values of other genes.

Epistasis may be tackled in two ways: as a coding problem or as a GA theory problem. If it is a coding problem then a different representation and decoding method must be found that does not exhibit epistasis. For many difficult problems a non-epistatic representation may involve a great deal of effort. A hybrid that takes account of epistasis may be more appropriate.

One of the fundamental principles of GAs is that chromosomes that include elements of the global solution increase in frequency as the algorithm progresses. Eventually these elements combine, via genetic operators, to create the global solution. However, if elements that are not in the globally optimal solution increase in frequency faster than those that are, the GA is misled. This is a form of epistasis and is known as

deception. Deceptive problems are difficult to solve. Reordering operators, such as inversion [63], may be one way of coping with such problems.

If by chance a gene becomes predominant in the population it can eventually be copied into every member of that population; it becomes fixed. The convergence of this gene causes a ratchet effect and other genes begin to converge. This process is known as *genetic drift*. Higher mutation rates reduce the effects of genetic drift, but if rates are too high the search effectively becomes random.

If a GA converges as a result of genetic drift before a globally optimal solution has been reached, it is said to have *prematurely converged*. Premature convergence can be blamed on the loss of critical alleles due to selection, schema disruption due to crossover; and parameter settings such as mutation rate, crossover rate, and population size. A critical allele is lost whenever no chromosome in a population has that allele value [129].

Research indicates that the best parameter settings for online performance are population size (20-30), crossover rate (0.75-0.95) and mutation rate (0.005-0.01) [13]. These values are used as a starting point in Chapter 3.

2.8.2 Smorgasbord of GA Topics

GAs are inherently parallel, that is the degree of parallelism is directly proportional to the population size [66]. Parallelism will not be employed in the proof of concept tool, *X-Topmeter*. It is however an obvious method of speeding up the tool for industrial use.

In simple GAs it is assumed that attributes can be combined linearly. In reality techniques used to produce appropriate fitness functions, such as Multi-Criteria Decision Analysis [83], are unlikely to produce functions that exhibit such a restrictive characteristic. Horn *et al* [70] have considered ways of using methods from decision analysis to allow characteristics of a solution to be applied multiplicatively within a GA framework. Their approach also accommodates uncertainty in the attribute values, although it does not give the full flexibility of MCDA.

During a search run the optimal value for each operator probability may vary. Linear variations in crossover have been used and been found, in many cases, to be effective. A dynamically variable crossover rate, depending on the spread of fitness, allows a more flexible approach. Davis [37] describes another technique which is based directly on the success of an operator at producing good offspring. These approaches have “hints of” the *Tabu Search* approach.

A number of formulations of guided search techniques for SAT problems have been presented by Ferland & Fleurent [52]. SAT GA solutions are encoded as bit strings

and therefore the standard bit-string genetic operators can be employed. Ferland has produced a crossover operator specifically for SAT that hybridises a GA with a local search. This operator attempts to find clauses that are satisfied in one parent, but unsatisfied in the other. The corresponding solution variables are assigned values according to the parent satisfying the identified clause. Values of the remaining variables are then generated as per the standard uniform crossover operator. The mutation operator is the classical bit-wise operator.

2.9 Search Space Evolution via Tabu Search

Tabu Search (TS) [60] is a neighbourhood search technique that starts from some initial feasible solution. It escapes local optima by imposing restrictions, based on a short-term memory of recent solutions and strategies that mimic long-term memory processes, to guide the search process. Reeves [141] indicates that

“The notion of exploiting certain forms of flexible memory to control the search process is the central theme underlying Tabu Search”

TS has been used to solve combinatorial optimisation problems [15], including scheduling [169, 99]. Each admissible solution s is evaluated according to the cost function $c(H,s)$, where H is the history record of the states encountered during the search.

During each iteration of a TS an admissible move is applied to the current solution, transforming it into its neighbour with the smallest cost. In terms of the unified approach (Figure 2.17) the *initialise* function produces both the starting solution and the history of that solution. The *select* function chooses the current solution. The *create* function uses $c(H,s)$ instead of $c(s)$ to generate new solutions. The *merge* function uses $c(H,s)$ to determine which solution survives to the next iteration. The algorithm *finishes* after a given number of iterations.

The admissible set of neighbours each iteration is determined by a *Tabu list*. A potential solution can become *tabu* for a number of reasons, such as the move has been tried previously. In its strictest form a tabu move can never be made. However, in the ‘long run’ this is not the case because of three features of the list:

1. Items on the list are not permanently tabu (recency). One possible static rule is choose a move to be tabu for t moves. One possible dynamic rule is choose t to vary randomly between bounds t_{min} and t_{max} ; t is an integer.
2. Frequency based memory. History is used to create a modified evaluation of currently accessible solutions. Residence and transition frequency measures can be used.

3. Aspiration level conditions may override the Tabu status of a move. Consider a Tabu move m on solution s . This move may be accepted if it has an aspiration level, $a(s,m)$, greater than a threshold value $A(s,m)$. Aspirations can be move or attribute based.

Memory in TS is also used as a kind of learning process. Information on the characteristics of moves that produce better results is built up and an *intensification* scheme developed to exploit this information.

GAs and TS

The designer of a Topology is likely to rule out a number of alternative topologies as the design process continues. These restrictions are made as a result of derived requirements, previous experience and previous iterations of the Topology selection process [42], see Section 3.2.5.

The more *fractured* a design space becomes the more likely a GA search engine is to suffer *premature convergence*. A fractured search space has many solutions that are deemed to be *infeasible* dotted around it. TS systematically imposes and releases constraints to allow exploration of otherwise forbidden regions of the search space. Implementing elements of TS within the GA framework for the Topology selection problem may therefore be appropriate.

Reeves [140] suggests that *recency* can be adopted at a variety of levels within a GA. For instance,

- treat a crossover point as an attribute so that the GA is encouraged to explore the effect of crossing over at different places.
- make a chromosome tabu, effectively forbidding it to reproduce until its tabu tenure has expired.

Reeves also suggests that frequency measures can be employed in a GA context to guide mutation of particular bits of a chromosome and promote fitness values of chromosomes based on their frequency in the population.

2.10 Simulated Annealing

Simulated Annealing (SA) is a variant of the general search paradigm presented in Section 2.7.1, and has been applied to a wide range of practical problems [40]. A simple sequential SA algorithm for a minimisation problem, is presented in Figure 2.18. Maximisation problems can be addressed as the dual of this algorithm.

Problem: minimise $P(s)$ such that $s \in S$

Algorithm:

```
Select an initial Solution  $s_0$ ;  
Select an initial temperature  $t_0 > 0$ ;  
Select a temperature reduction function  $\alpha$   
Repeat  
  Repeat  
    Randomly select  $s \in N(s_0)$ ;  
     $\delta = P(s) - P(s_0)$ ;  
    if ( $\delta > 0$ )  
      then  $s_0 = s$ ;  
    else  
      generate random  $x$  uniformly in the range (0,1);  
      if ( $x < \exp(-\delta t)$ ) then  $s_0 = s$ ;  
  Until iteration_count = nrep;  
  Set  $t = \alpha(t)$ ;  
Until stopping condition = true;
```

s_0 is the approximation to the optimal solution.

Figure 2.18: SA Minimisation Algorithm

SA was initially inspired by the *laws of thermodynamics* which state that at temperature, t , the probability of an increase in energy of magnitude, δE , is given by

$$P[\delta E] = \exp(-\delta E/kt) \quad (2.7)$$

where k is the physical constant known as *Boltzmann's constant* and t can be considered to be a parameter of the process. In a *simulated* version this equation is used within a system that is 'cooling' towards a steady (frozen) state.

In the author's paper [117], the *initialise* function picks a solution s_0 , with random characteristics. The value $P(s_0)$ of this solution is calculated. The *select* function used is the identity function, that is the current solution is placed in Q . A point in the neighbour space, s , is chosen using the *create* function and $P(s)$ calculated. A logistic acceptance criterion is applied [79] and the *merge* function is simply to make P the new solution if it is accepted.

In Figure 2.18 the energy change resulting from a potential move from s to s_0 is calculated ($\delta = P(s) - P(s_0)$) and the move is accepted if δ is greater than 0. However, if δ is 0 or less the move is only accepted with the probability given by equation (2.7).

The initial temperature, t_0 , is chosen automatically by the algorithm [85] so that virtually all proposed jumps are taken.

Once a suitable starting solution and temperature have been produced, the main iterative cycle is started. The temperature is *reduced* by a factor α , where $0 < \alpha < 1$. Values of α between 0.9 and 0.99 are common in the literature. The number of downward (penalty decreasing) jumps are counted and equilibrium is said to be achieved when the count exceeds a threshold. This approach leads to potentially infinite chains (especially at low temperatures) and so an upper bound on the number of trials is enforced. Once equilibrium has been achieved the temperature is reduced and another set of solutions tried. The *finish* function for the algorithm is also determined automatically; the search is terminated when no upward or downward jumps have been taken over a number of reductions in temperature.

2.10.1 Extensions to SA

The standard SA algorithm uses uniform random sampling with replacement of a neighbourhood. Other schemes can be envisaged [64]. The number of neighbours considered before an accept/reject decision is made can be increased as the temperature falls so that the local area is searched more intensively. More complicated strategies for local searching have been put forward. For instance, Localised SA (LSA [96]) contains a local and a global annealing schedule, allowing a more extensive search of particular neighbourhoods.

Researchers have created schemes to alter the accepted to generated ratio dynamically. For instance, Adaptive SA (ASA) produced by Ingber [75] allows an exponential temperature annealing scheme to be applied. Other researchers have produced schemes with *constant* [33] and increasing temperatures. *Tempering* employs a parameter that specifies the threshold percentage, which when reached reheats the annealing algorithm.

In cases where the evaluation (cost) function is necessarily computationally complex it may be desirable to use an estimate for most moves and only apply the full cost function at specified intervals. Estimation can be undertaken by simulation to provide a strategy for the use of estimates [164]. Cost functions for which the calculation of the *difference* between old and new solutions is quick can significantly speed up the search. Such functions are often referred to as *delta* functions.

2.10.2 Applying SA

As for GAs an evaluation function can be formulated as a *penalty* function that gives high scores to ‘bad’ attributes of a solution and low scores to ‘good’ attributes. The

penalty function can be formulated as a linear combination of non-linear factors. For instance, in the allocation problem the square of the number of excess sub-tasks on a processor is penalised. Considerable effort is often expended to choose weighting factors and cooling rate, so that an appropriate mix of intensification and diversification operations is achieved.

Comparisons have been made between SA and other techniques. For instance, Crabtree [34] compares SA and the state based Constraint Programming approach for the optimisation of a general resource allocation problem. He concludes that there is a smooth transition between problems for which constraint programming is dominant to those for which SA is dominant. The most influential factor is the number of precedence constraints, with SA being dominant for resource allocation problems with a high proportion of precedence constraints. In Brind et al [19] five stochastic techniques (GA, hill climbing, random search, SA, and TS) are applied to a vehicle routing problem (VRP). In a 10 minute run (on a sparc IPC) SA dominates the other techniques, and performed well on all problems. With more time available the GA outperformed other techniques for under-resourced problems. SA performed well on both under- (and over-) resourced problems.

The application of SA to allocation and scheduling [135, 41], has been investigated by many authors, see Section 4.2. SA has also been applied to synthesis problems. Smith & Setliff [157] produced *RT-Syn* a low-level software synthesis architecture targeting real-time software design, that searches the implementation design space. RT-syn generates functional C code from high-level algorithmic descriptions.

SA and TS can be hybridised [55]. Temporal memory can be exploited by employing a tabu list for particular moves. That is, once tried a move may not be tried again for n moves. Spatial memory may also be incorporated to pick candidates from the neighbourhood of the current solution. The penalty function is chosen to inhibit the selection of undesirable candidates. In order to ensure the convergence-in-probability feature of SA with the Tabu element added, the Tabu element must not employ incentives, time-dependent penalties and neighbourhoods, and unbounded memory [55].

2.11 Discussion

In this Chapter a number of essential attributes of a topology for a real-time safety-critical control system have been introduced. Anyone wishing to select a new topology, or evaluate an existing topology, needs to consider the following:

- units employed,
- dependability characteristics (selection and analysis),

- timing characteristics (selection and analysis), and
- search techniques to determine an ‘acceptable’ topology.

The notion of selection is however more multifaceted, representing as it must academic and industrial experience of best practice. Many of the elements of a topology presented here appear to be prescribed. For instance, thou shalt use this computational model and employ pre-emptive priority based scheduling that should be analysed using these schedulability techniques. This exact set of implementations is very unlikely to be appropriate for all industrial applications. They are however, one set of valid techniques and topological alternatives that could be employed as part of a topology selection process. This, or a similar, set of techniques are required to show the effectiveness of the approach presented in this thesis.

As a result of this insight it became obvious early in the project that a set of flexible topology selection techniques were required. The techniques also needed to be generic in nature to minimise the amount of rework for each application. A guided search and data/technique library based approach has been adopted in the next two chapters. A fully automated topology selection process is not possible, partly because of the need for flexibility in the techniques employed. User intervention, whether by designers, procurers or maintainers is vital if an appropriate topology is to be selected.

The user should ensure that the WCRT and dependability prediction techniques employed to evaluate the quality of a distributed control system are pessimistic and safe. The user should also bear this pessimism in mind when the approach finds it difficult to select a valid topology. For instance, are the assumptions and prediction techniques employed too pessimistic?

Finally, the user should consider the validity of the independence assumptions made during the architectural topology selection process. Independence between transactions and Services is unlikely to hold throughout the design process. This does not necessarily invalidate the topology selection decisions made. However, careful analysis and investigation should be undertaken before the timing attributes of a proposed topology are investigated to ensure that the user is satisfied with the proposed architectural topology. This is one of the factors behind splitting the topology problem into two distinct sub-problems. These sub-problems are addressed in the next two chapters.

Chapter 3

The Topology Problem at the Architectural Level

The aim of a topology selection process is to compare the relative merits of a set of alternative topologies, not provide precise MTTF predictions. Thus, an iterative multi-level approach is required so that a topology can emerge during the functional design process. The first level at which it is practicable to address topology selection is the architectural level. At this level the generic architecture to be employed by the system, along with a set of admissible resources, has been determined. Furthermore, a set of control services have been postulated. The design issue addressed in this chapter is: given this architectural, resource and control information, determine an architectural topology of minimum complexity that meets a set of dependability requirements. Complexity is measured in terms of the size and cost of the platform of hardware and software units implied by the selected architectural topology.

Thus in Section 3.1 a generic architecture and a *straw-man* topology development process are presented. In Section 3.2 the architectural topology problem is formulated as an ATP. In Section 3.3 the use of GAs to aid resolution of the architectural topology problem is discussed. A tool, X-Topmeter, that employs a GA is presented in Section 3.3.1. A number of illustrative examples have been investigated, see Section 3.4.

Finally, in Section 3.5 a discussion of the remaining issues surrounding the prototype tool, as currently implemented, are presented. In particular the effect of using a different generic architecture is discussed. Essentially, existing results are invalidated, but re-implementation is often relatively straight-forward.

3.1 The Architectural Framework

The architectural topology chosen to implement a distributed control system is based on a generic architecture. A plethora of architectures have been put forward for this class of problem. Essentially they fall into two categories: bus (e.g. IMA [78]) and shared memory (e.g. DIA [177]) architectures. In this thesis the prototype tool, X-Topmeter, is based on one of the latest generic architectures, GUARDS (Generic Upgradeable Architecture for Real-Time Systems) [130]. GUARDS was chosen purely as an exemplar. There is no reason why the approach employed cannot be adapted to investigate shared memory architectures, or other bus based architectures.

The GUARDS generic architecture consists of a set of hardware units and software mechanisms, together with rules for configuring various instances of the architecture for different end-user requirements. An instance of a GUARDS architecture forms an architectural topology and can be defined using three concepts (see Figure 3.1):

- the number of channels (C),
- the multiplicity of resources within a channel (M), and
- the number of integrity levels supported (I).

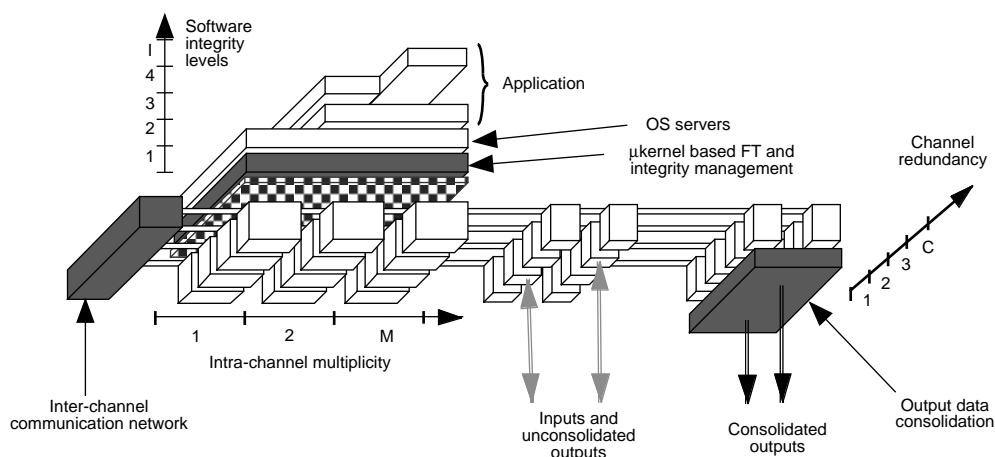


Figure 3.1: The GUARDS Generic Architecture

If two channels are implemented ($C=2$) a *fail-safe* topology can be produced by using duplication and comparison to guard against physical faults, and/or diversification and comparison to guard against design faults. If three channels and a decision mechanism are implemented, a TMR topology that enables faults in one channel to be masked is indicated. If four channels are implemented a single Byzantine failure can be guarded against. A four channel topology also allows off-line testing of one channel while

a degree of fault tolerance is guaranteed through voting on the remaining on-line channels. The topologies introduced in Section 2.3.4 can be accommodated within the GUARDS framework.

The Multiplicity axis, M , allows secondary fault containment regions to be defined within a channel. In other words individual tasks, such as Input, Processing and Output can be subjected to different levels of redundancy. Multiplicity can be employed to improve both fault-tolerance and real-time performance. For instance, the Boeing 777 [108] employs an $m=3$ architecture for the processing tasks; that is it employs diversely implemented processing *lanes* in each channel.

Integrity levels [110] were introduced in Section 2.2.1. Assume that the applications of interest employ four levels of software integrity. In the GUARDS architecture a number of protection features are also provided, such as memory protection and temporal firewalls. Any protection measures that operate at *run-time* are assumed to be the responsibility of the real-time safety-kernel.

To produce a topology based on an instance of this architecture two requirements need to be considered [159]:

- amount of data to be transmitted around the bus-based system and
- best way of exploiting redundancy in the distributed system.

An architectural topology based on GUARDS must therefore contain:

- inter-channel communication network(s),
- an output data-consolidation system used to ensure physical outputs are *safe*. The chosen system topology may employ a number of decision mechanisms,
- the basic operating system kernel needed to manage the underlying FT hardware.

An Integrated Modular Avionics (IMA) [78] network has significant advantages over topologies employing multiple networks, including reduced development costs, reduced cost of spares and the possibility of sharing resources. An IMA architecture in which groups of modules form a Topology for a task is shown in Figure 3.2. Each module within the group performs one function and is connected to other modules by a single logical data network. Multiple physical data buses may be employed to increase dependability. The GUARDS generic architecture can be formulated as an IMA by implementing an appropriate inter-channel communication network.

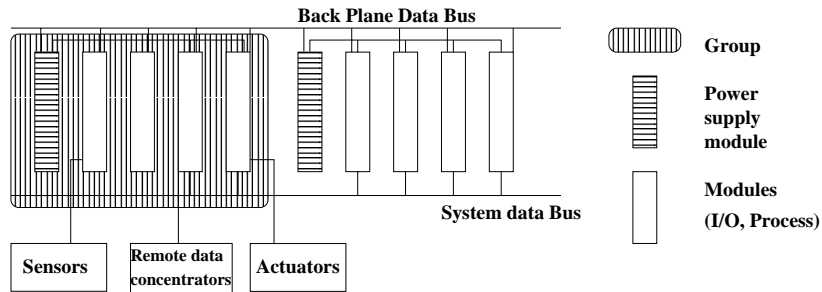


Figure 3.2: An Integrated Modular Architecture

So to summarise, a control service may employ an architectural topology drawn from the set of topologies made possible by the use of the GUARDS architecture. Single or multiple network communications may be employed. The software used is designed to a particular integrity level and decision mechanisms may be employed to produce a consolidated output. Redundancy is either at the channel (input, processing *and* output) level or at the task level (input, processing *or* output).

Straw-Man Development Process

Thompson [159] has proposed a development process for the production of a distributed fault-tolerant topology. This work is based on experience from industrial sponsors, as well as work in academia, and attempts to push forward best practice in the production of architectural topologies. The thirteen elements of the approach are:

1. Propose architecture
2. Initial control law design
3. Coarse selection of a topology
4. Choice of a topology
5. Evaluation of reliability and availability
6. Positioning for safety and environmental considerations
7. Modelling of fault-tolerance mechanisms
8. Modelling of databus delay
9. Control law redesign to tailor to chosen topology
10. Scheduling of control functions on the processing platform
11. Power requirements and heat dissipation

12. Full Markov modelling of refined processing topology
13. Full linked series of models of the proposed system topology.

The approach can be split into three phases. First, an analysis phase (steps 1-3) that sets the form of the architecture to be used and the control laws to be implemented by a distributed computer based system. Second, a synthesis phase (steps 4 to 9) in which the topology and system to be implemented emerge. Finally, an evaluation phase (steps 10 to 13) in which the distributed control system is evaluated relative to the proposed topology. Step 10 of this cycle is considered in detail in Chapter 4. This approach follows the general Analysis-Synthesis-Evaluation (ASE) methodology for the design of an artefact proposed by Maimon & Braha [101] (see Chapter 1).

In industrial practice the synthesis phase consists of the designer proposing a topology from experience. Future demands on the system [147] are then considered. Changes are made to the proposed topology and an evaluation of characteristics, such as reliability and availability, is undertaken. This process continues until an acceptable topology is found.

The ASE method appears to be a reasonable approach to the production of an architectural topology. However, a number of ways of enhancing this approach are apparent. These enhancements focus on the synthesis and evaluation phases of the development cycle and form the main improvements to the distributed control system development process proposed in this chapter. The approach improves the synthesis phase by employing predictive reliability modelling and evaluation techniques, together with automated search techniques, to investigate and evaluate a much greater range of topologies. Thus reliability is explicitly introduced into the topology selection process.

3.2 The Architectural Topology Problem

In Thompson's approach it is implicitly assumed that the functionality of a control system can be produced to fit any one of a set of topologies in a simple manner. To stretch a point, a set of control laws are determined, a topology is produced with these laws in mind and the control laws are then tweaked to ensure that the functionality works. This is almost never the case in practice because of three interacting factors: system complexity, novel aspects and people factors. In this chapter therefore an architectural topology is deemed to be an emergent property of the system design process. That is design for functionality and selection of a system topology occur together, rather than separately or in parallel.

To aid resolution of the architectural topology problem the following information must be obtained:

- representation of an architectural topology
- measure of the quality of a proposed topology
- operations to produce new topologies
- restrictions on the set of admissible topologies

3.2.1 Formulation of the Architectural Topology Problem

In Section 1.3 the topology selection design issue was introduced and in Section 2.1 a definition of the problem as an ATP was presented. In this Section a formulation of the Architectural Topology problem as an ATP is presented. That is assign a set of architectural components and resources to produce a topology. It is important to keep the flexibility of a formulation that allows both goals and constraints to be expressed in a single formulation. The goals can however be transformed into constraints, allowing formulation as a satisfiability problem, see Appendix B.

A solution to the architectural topology problem, x , requires four assignments of items to resources:

1. exactly one architectural component, from the set of components $\{c \in 1, \dots, C\}$, is assigned to each task $\{t \in 1, \dots, T\}$ (see Figure 1.3). Thus $x_{ct} \rightarrow (0, 1)$, where $x_{ct} = 1$ implies component c is employed by task t .
2. Exactly one component per network. Thus, $x_{cn} \rightarrow (0, 1)$, where $x_{cn} = 1$ implies component c is employed by network n . In fact the set of components each task (network) can employ is restricted to $C_t \subseteq C$ ($C_n \subseteq C$) because some tasks (networks) may not employ particular architectural components.
3. A set of resources to each component-task combination. Thus, $x_{r(ct)} \rightarrow (0, 1)$, where $x_{r(ct)} = 1$ implies resource r is employed by component-task combination ct . The set of resources that can be employed by any component-task combination is restricted to $R_{(ct)}$ for each task in the topology.
4. A set of resources to each component-network combination in the topology. The set of admissible resources for combination cn is $R_{(cn)}$.

The subsets C_t , C_n , $R_{(ct)}$ and $R_{(cn)}$ may change as the design process progresses and are used to ensure that incompatible resources are not employed in a topology. Multiple copies (units) of any resource can be employed in an architectural topology. The number of units required is calculated from the set of resource assignments. The architectural topology problem formulated as an ATP is:

$$\begin{array}{ll}
\text{Minimise } F(\mathbf{x}) & \\
\text{subject to} & \mathbf{Constraints} \\
\sum_{c \in C_t} x_{ct} = 1, & 1 \leq t \leq T \quad \text{- Task} \\
\sum_{c \in C_n} x_{cn} = 1, & 1 \leq n \leq N \quad \text{- Network} \\
\sum_{r \in R_{(ct)}} x_{r(ct)} \geq 1 & 1 \leq t \leq T \quad \text{- Component-Task Resources} \\
\sum_{r \in R_{(cn)}} x_{r(cn)} \geq 1 & 1 \leq n \leq N \quad \text{- Component-Network Resources} \\
\min mttf_s - mttf_s \leq 0, & 1 \leq s \leq S \quad \text{- Reliability}
\end{array}$$

where

$F(\mathbf{x})$	function indicating quality of proposed resource assignments
s	service
t	task
n	network
x_{ct}	1 if task t employs architectural component c , 0 otherwise
x_{cn}	1 if network n employs architectural component c , 0 otherwise
$C_t \subseteq \{1, 2, \dots, C\}$	set of admissible components for task t ($\{t=1, \dots, T\}$)
$C_n \subseteq \{1, 2, \dots, C\}$	set of admissible components for network n ($\{n=1, \dots, N\}$)
$R_{(ct)}$	set of admissible resources for combination ct
$R_{(cn)}$	set of admissible resources for combination cn
$mttf_s$	predicted mean time to failure for service s
$\min mttf_s$	minimum acceptable mean time to failure for service s

The value of the reliability side constraint is measured using the predicted MTTF of each Service. The objective function, $F(\mathbf{x})$, employs cost and size measures. Timing properties are used to indicate the minimum number of processing hardware units of each type in a proposed topology. The evaluation function for an architectural topology is a combination of the objective and side constraint functions. The set of architectural components will be discussed in Section 3.2.2.

The assumptions required to address an architectural topology problem are common to both SAT and ATP formulations. They reflect the desire to keep the approach simple and hence tractable. They are however realistic and can in most cases be relaxed if desired. In Section 2.3.4 the effect of common-cause / common-mode (CC/CM) failures on the reliability models are discussed. CC/CM failures break independence assumptions. The following set of assumptions are employed in the illustrative examples in this thesis:

1. Resources, Components, Tasks and Services are all 2-state (good, bad)
2. Failures of individual units are independent.
3. Failed units do not damage the system and are not repaired.

4. Failures are due to fail-stop, Omission, Timing or Incorrect computation faults, see Section 2.3.1.
5. Hardware failure rates are known, deterministic, and are constant per unit time.
6. Software failure rates are known and follow a Non-Homogeneous Poisson Process [114].
7. Four software Integrity levels may be employed, $\max(I) = 4$
8. Maximum of three channels may be employed, $\max(C) = 3$
9. Maximum of three units per task may be employed, $\max(M) = 3$
10. Four combinations of sparing and decision mechanisms may be employed per task (network).
11. All communication networks are assumed to be implemented as buses.

These assumptions are compatible with the GUARDS architecture introduced in Section 3.1. They are also compatible with the V-model and ASE approaches to resolving design issues.

3.2.2 Representing a Topology

In Figure 1.3 a solution to the topology problem, x , was shown to consist of a set of configured hardware and software units. The type of units employed is determined by selection from the set of available resources, R , and the topological configuration is determined by the set of architectural components, C , employed. The number of units required is inferred using assumptions about independence and the maximum utilisation of resources.

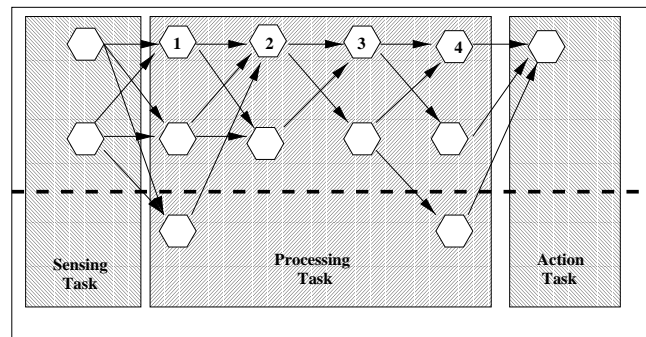


Figure 3.3: Logical Architectural for a Single Service

In Figure 3.3 a representation of a single service is shown. This representation consists of three tasks, all of which must work correctly for the system to apply appropriate

control activities. Each task can be further decomposed into sub-tasks, shown as hexagons. These sub-tasks provide the functionality and fault tolerance capabilities of a distributed control system. In Figure 3.3 for instance, the sensing task employs two sensors in parallel. This may be either to provide values for two different environmental variables or to provide redundant sensing of a single environmental variable. Results from these sensors are employed by three processing sub-tasks in parallel resulting in three control values. An action sub-task takes the value of the first of these results to arrive and provides the appropriate control action.

The graph of sub-tasks forms a precedence ordered Directed Acyclic Graph (DAG), which can be analysed to produce a measure of the reliability of a Service [8]. Different DAGs are produced for different configurations of sub-tasks generated by the use of different functional designs and fault-tolerance techniques.

For the architectural topology problem, setting a topology at the sub-task level is inappropriate. For instance, assume that in Figure 3.3 four logical sub-tasks are employed to implement the processing task. These sub-tasks are represented by the four hexagons in sequence along the top of the processing task section of the diagram. All other sub-tasks in this processing task are employed to provide fault tolerance.

The desirability of this processing task topology is not obvious. Neither is it obvious how to implement such a topology in architectures such as GUARDS. Furthermore, at early stages in the design process, where the architectural topology problem should first be addressed, the design will not be decomposed into sub-tasks, indicating that redundancy should be implemented at a coarser level.

In this thesis therefore architectural topologies are set at the task level and a mapping onto sub-tasks made when the design has been decomposed to an appropriate level. For example, suppose the processing task in Figure 3.3 is to be implemented as a dual-replica task topology. Each of the four functional sub-tasks employs a single replica. This dual-replica topology consists of the sub-tasks above the dotted line in Figure 3.3.

In Figure 3.4 a simple model of a control action implemented as a single control Service is presented. Extra services can be added to the system simply by repeating the structure shown here, with the task id's changed as appropriate. In this case, data is transmitted around the system via three bus-based networks;

1. between the sensors and the computer system,
2. within the computer system, and
3. between the computer system and the actuators.

The amount of data transmitted around each network is determined by the number of copies employed for fault-tolerance as well as the functional requirement for data

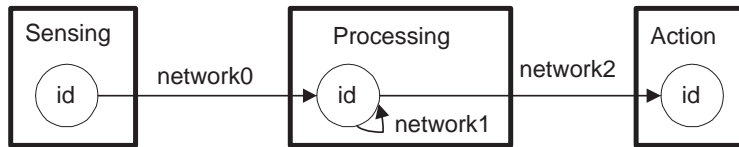


Figure 3.4: Three-task and Three-network Service

transmission indicated by a proposed design. An alternative service structure that employs a single IMA network, see Figure 3.5, may be used.

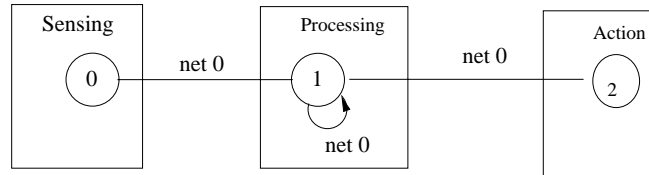


Figure 3.5: Three-task and One-network Service

Task, network and hence service topologies, may be represented as block diagrams. In Figure 3.6 a selection of architectural topologies based on topologies presented in Section 2.3.4 are shown.

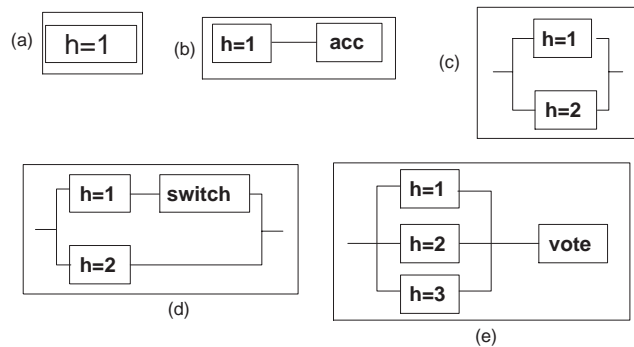


Figure 3.6: Sample Topologies

Task topology (a) represents a simplex topology that uses hardware of type (resource) one. In task topology (b) resource one is employed, along with a software based acceptance test to transform value failures into omission failures. In task topology (c) a redundant set of sub-tasks are employed in parallel. That is the task runs the required functionality twice, once on units of resource type one and once on units of resource type two. In task topology (d) the task is implemented as a redundant cold-spares topology. Finally, in task topology (e) the task is implemented as a redundant TMR with voter topology. In this case each sub-task has three copies implemented on different hardware resources.

Network topology structures can be represented in a similar manner. It should be noted however that neither action tasks nor networks may employ decision mechanisms. Furthermore networks share resources among a number of tasks. For instance, all sensing tasks are linked to all processing tasks via a single network. Reliability and system models must take this into account. In the GUARDS architecture ensuring that the failure of one element of a network cannot affect other elements is the responsibility of the real-time kernel and the underlying operating system.

A numerical, as well as a graphical representation, can be produced. An integer representation of an ATP formulated architectural topology employs three sets of values:

- number of copies of each admissible resource employed $[0 - max_1)$ per task
- combination of sparing and decision mechanisms employed $[0 - max_2)$ per task
- number of copies of each admissible resource employed $[0 - max_3)$ per network

For an architectural topology problem formulated as a SAT a solution may be encoded as a bit-string of size n . Each element in the bit-string takes on a value of one if the constraint governed by the accompanying system variable is met and zero otherwise. A clause is true if a given set of constraints are met. For instance, if the MTTF for task i is greater than the required target MTTF then the string element for the MTTF of task i takes on a value of 1.

3.2.3 Evaluating an Architectural Topology

Determining an appropriate evaluation function is vital as it encapsulates the quality of a proposed topology in a single measure. For instance, suppose the reliability of a proposed topology is to be used as part of the evaluation function. To produce such a measure reliability models of the proposed task, network and service topologies need to be produced and the MTTF of each task, network and hence service calculated. The value used by the evaluation function to encapsulate all this information could then be based on the number of services that fail to meet their MTTF requirements. The quality of the evaluation function and the information it encapsulates to a great extent determine the quality of the results from the search process.

To produce an evaluation function proposed design decisions need to be evaluated with respect to a variety of System Design Factors (SDFs) [143] such as reliability, cost and performance. There is no general agreement on either the set of SDFs to be employed nor on models for evaluating any given SDF. There have however been a number of decision support tools that allow MCDA [20] to be undertaken. Thus for a given design issue the ability to evaluate a proposed solution is dependent on three elements;

1. set of SDF's chosen to characterise a particular design decision,
2. measures used to produce a quantitative evaluation of each chosen SDF,
3. combination of measures of the SDF's into a single evaluation function.

A set of topology SDFs have been identified, see Table 3.1. There are guidelines available for checking whether this set has certain desirable properties, such as completeness and non-redundancy [146]. These properties are meta-evaluators for the design decision. A balance needs to be struck between considering all SDFs of a decision and keeping the problem simple to analyse.

Attribute	Description
hardware	Hardware resources employed
software	Software resources employed
size	Number of units employed
dependability	Dependability of control services
cost	Cost to implement topology

Table 3.1: Task Attributes

Glocker et al [59] introduce a two stage system architecture analysis (SAA) method developed to allow evaluation of proposed system architectures at Siemens. In stage one a set of evaluation criteria are produced based on customer requirements, project requirements and quality attributes. A team is formed which includes experts from marketing, sales and servicing. The team identifies a set of evaluation criteria. Requirements are ordered according to their level of abstraction. Then, requirements at a certain hierarchy level are selected as evaluation criteria. Finally, requirements are weighted by assigning weights in a pairwise comparison procedure. In stage two system implementations are proposed and evaluated using the criteria produced in stage one. Stage two for the architectural topology problem has already been covered in Section 3.2.2. Having canvassed widely with industrial partners, MSc students from industry, the research literature, and the METHOD project personnel the following three SDFs are used to evaluate a given architectural topology in the prototype tool, X-Topmeter.

1. Cost of the proposed topology
2. Size of the proposed topology
3. Reliability of the proposed topology

Once a set of SDF's have been identified measures to produce a quantitative evaluation of a topology can be addressed. The effectiveness of a measure of an SDF is determined by the granularity of the measure and expressive power of the models (see Section 2.4.2) used to produce the measure. Furthermore, the speed with which a measure can be calculated must be borne in mind if the measure is to be used as a design aid.

Minimising the value taken by the first two SDFs identified above form the goals of the architectural topology problem and are therefore incorporated into the optimisation function ($F(x)$). The required level of reliability imposes a constraint on the set of feasible architectural topologies and is therefore incorporated into the constraint function ($G_d(x)$). The proposed measures are:

- cost of ownership of a proposed topology,
- size of the hardware platform implied by a given topology, and
- predicted Mean Time to Failure of each service in the topology.

Finally, the three measures need to be *combined* to form an evaluation function. This function determines the ability of the selection process to choose an appropriate architectural topology. The functional form of the evaluation function could be additive, multiplicative, polynomial, etc. *Corporate knowledge* of designers on how to approach particular design issues may be employed to determine the functional form.

In the case of an architectural topology a fairly naive additive evaluation function can be employed to encapsulate the relevant information about a topology. Further industrial based research is required to produce a more accurate evaluation function. Results with this additive function are sufficiently encouraging to support the viability of the approach. The evaluation function proposed for an architectural topology is:

$$V_{topology} = F(x) + G(x) = (k_{cost} * cost) + (k_s * size)^2 + (k_{rel} * mttf)^2 \quad (3.1)$$

where

- k_x = weighting factor for element x determines the change in the evaluation function value of an incremental change in factor x .
- $cost$ = production and ownership cost of resources in the topology
- $size$ = number of units of each hardware resource used in the topology.
- $mttf$ = difference between predicted mean time to failure (MTTF) and required MTTF. If predicted MTTF is greater than required MTTF then $mttf$ equals zero.

The most complex element of the proposed evaluation function to evaluate is the *mttf* element. This requires the production and evaluation of a number of task, network and service reliability models. The MTTF of each Service is predicted using the formulae proposed in Section 2.4. If a service is predicted to meet its MTTF target then a value of 0 is added to *mttf*. If a service is predicted to fail to meet its MTTF requirement a value equal to the number of time units it is predicted to fail by is added to *mttf*.

For an architectural topology formulated as a SAT the solution string may be evaluated with respect to the number of satisfied clauses. If the SAT is formulated as a *minimisation* problem the aim will be to minimise the number of system variables that fail to meet the constraints that define the set of clauses [51].

In conclusion, whichever evaluation function is chosen the results of a search for a topology is determined by this function. If the evaluation function is a poor representation of the actual value of a proposed topology then the chosen topology will not provide the required characteristics. Explicit justification for the form of evaluation function used is required. This justification may take the form of *best practice*, company standards or explicit evaluation using MCDA. In this thesis the form of the evaluation function is chosen from informal perusal of academic papers and discussions with industrial partners. In an industrial setting work like that presented by Glocker et al [59] is more appropriate.

3.2.4 Producing New Solutions

A set of alternative topologies can be generated by altering a small number of attributes of task and network topologies. In an ATP formulation each task (network) is assigned to exactly one architectural component. The elements of this component are then assigned to a set of primitive resources. So the algorithm is; determine an architectural component and then choose a set of primitive resources to implement the component.

A systematic approach could be employed with each possible combination of architectural components and resources being taken in turn, evaluated, and the best result used. The theoretical basis of this systematic approach is a *design space* [89] in which possible topologies for a system are modelled as points in a space spanned by so-called design dimensions [59]. A design dimension corresponds to exactly one SDF. Each possible design decision corresponds to a co-ordinate in the design space. The concept of a design space was introduced in Section 1.3.2.

The set of possible alternatives is vast. Suppose that the design space for a single service system, represented as three tasks and three networks, is to be determined. The set of admissible architectural components is restricted to those using up to three copies of three different resources. This restriction applies to both tasks and networks. Four combinations of sparing and decision mechanisms are allowed: none, cold spare

and acceptance test, hot spare and no decision mechanism, hot spare and voter. Thus, the design space is defined by seven restrictions

1. maximum of three copies of resource A_i per task, i
2. maximum of three copies of resource B_i per task, i
3. maximum of three copies of resource C_i per task, i
4. four combinations of sparing / decision mechanism per sensing and processing task
5. maximum of three copies of resource D_j per network, j
6. maximum of three copies of resource E_j per network, j
7. maximum of three copies of resource F_j per network, j

More complex search spaces are possible. However, bounding the design space in this manner appears reasonable. The total number of potential service topologies is 1.02×10^{12} . This design space is far too big to address using a brute force enumerative search approach. In Section 2.7 methods by which large design spaces can be traversed were discussed in detail.

The design space for an architectural topology problem formulated as an ATP, is generated by setting the value of four characteristics of each task, and three characteristics of each network, within defined limits. Production of these values may be accomplished by a systematic sequence on a very restricted design space or by a set of search operators.

For a SAT formulation a new solution can be generated by taking a bit and swapping its value. The underlying topology is changed to ensure that the corresponding system attribute either fails (swapped to zero) or succeeds (swapped to one). So for instance, if a dependability measure for a service should have a MTTF of at least 3000 units the set of resources and decision mechanisms used are altered to ensure that the system either fails or meets this target, as required by the appropriate binary value. In practice this design space is also be subjected to a set of operators.

3.2.5 Topology Evolution via Search Space Restrictions

The architectural topology issue is addressed more than once during the design process. Elements that may be changed during the topology selection process include:

- functional design for which a topology is to be produced

- set of admissible resources
- set of admissible components for given tasks, networks or combinations of tasks and networks
- set of SDFs and measures of SDFs
- formulation of the combination of SDF measures
- set of requirements to be met

Changing the functional design is a fundamental part of the design process. As the design process progresses attributes of the design, such as the predicted (estimated) failure rate of resources and coverage factors, may change. The designer must ensure that any restrictions on the design space are still relevant when the proposed functionality of the system is changed.

Restrictions to the design space can be employed to allow the designer to play *what-if* games. Scenarios are invented and the approach applied to indicate a good topology given the scenario. Thus, a designer may undertake multiple iterations at a single point in the evolution of a topology in order to settle on a design decision. This is particularly useful if meta-characteristics, such as maintenance or changes made during the operational lifetime of a system, are to be taken into account during a topology evolution process.

Two sets of restrictions can be employed: those that restrict the design space and those that change the evaluation function. The design space is inherently restricted by the minimum and maximum number of units from a library of resources that may be employed to implement tasks and networks. The designer may decide to change, or restrict, the set of admissible resources in the library. For example, reducing the number of admissible resources increases homogeneity but may also increase the level of replication required to provide appropriate levels of dependability.

The evolution of a topology may require changes in the evaluation function. For instance, an increase in the dependability requirement for a topology will change the set of feasible solutions, but not the overall set of topologies in the design space. A feasible solution is one where all side constraints are met.

Design Space Restrictions

Restrictions on admissible task and network components arise for a variety of reasons including rejection in previous iterations of the approach, domain knowledge of the designers and corporate knowledge from previous projects. Physics may also play an important role in restricting the set of alternatives. One combination of task topologies

that may be deemed unacceptable is sensing and action tasks employing different levels of redundancy. For instance, a designer may wish to declare that a dual redundant sensing task may not be employed with a triple redundant processing task.

Restrictions on the set of admissible topologies may allow an exhaustive search of the remaining design space [42]. In the early stages of the selection process this is unlikely. Therefore methods to implement searches over restricted search spaces need to be employed. One method is to employ a hybrid technique, such as local search with guided search techniques, see Section 2.7. A second approach is to transform inadmissible solutions into admissible solutions. A third approach is to heavily penalise inadmissible points in the design space.

Changes to the Evaluation Function

The set of SDFs may change as design progresses as different elements of a topology become amenable to evaluation. For instance, introducing a *maintenance* SDF early in the design process may not be possible due to the granularity of the data required. It can be introduced once the design has developed to an appropriate extent.

The way in which SDFs are combined may also change as the interactions between characteristics of the system become clearer and the goals of the designer change. Finally the weighting factors for each SDF may be subjected to evolution.

Future Systems

A log of the restrictions made can be kept and an investigation of their impact on the resulting topology undertaken at each level of decomposition. For instance, would the appropriate topology become clearer earlier in the selection process if certain types of restrictions were made before other restrictions? This log provides supporting evidence for decisions made by the designer. The designer also needs to consider which restrictions should be *preserved* into the next level of decomposition. This allows decisions to be maintained but reduces the scope for innovative changes in the topology.

The changes made and the order in which they occurred is an important piece of information to future designers that is currently lost for the topology problem. It allows future projects to attempt those changes that produced good results, including the best weighting factors. This data will hopefully allow designers to converge to a good solution more quickly and hence less expensively. For instance, there may be particular design steps that should always take place before others.

3.3 Resolving the Architectural Topology Problem

The GA literature abounds with attempts to apply GAs to combinatorial optimisation problems, with varying degrees of success. However, work on the use of GAs to solve mapping problems is relatively scarce [158, 170, 69].

Painton & Campbell [125] employ a GA to investigate how a system can be improved during its lifetime by adding to, or changing, hardware components in the system. A component may employ a number of hardware units in parallel. The problem formulation is both combinatorial and stochastic: optimise the reliability of a system design by identifying the best combination of component improvements to yield the highest 5th percentile Mean Time Between Failures (MTBF), while remaining within a budget constraint.

A number of features of Painton's approach are of interest. First, the approach uses a series-parallel reliability model. Second, three improvement options, based on the effect on the failure distribution of each component, are produced for each component. The improvement options are "no improvement", a "low cost improvement" and a "high cost improvement". Third, the GA uses tournament selection, two-point crossover and mutation. The coding of the chromosomes is integer; each allele is the number of the improvement option employed for a component. Fourth, the GA fitness function has two parts; a cost constraint and a MTBF goal. An example is given and favourable results presented. The GA finds a best fitness value within 0.0006% of the best fitness value in the search space. This approach is encouraging as the results it produces show the efficacy of the approach over existing Integer Programming and Hill climbing approaches.

Coit & Smith [31] produce a GA to address the redundancy allocation problem for a series-parallel system. They present an overview of previous research in this area which has mainly employed dynamic programming, Integer Programming and heuristics. They expand the type of series-parallel systems that can be investigated to include k-out-of-n redundancy. The GA has an integer coding based on the number of redundant units employed per sub-system. The fitness function is a penalty function based on the squared constraint violation in the system. Uniform crossover and a predetermined number of mutations per generation are employed as the genetic operators. The merge function employs a form of elitism. Coit & Smith's exemplar GA converges to a 'good' solution having traversed only 0.0021% of the search space. Overall, their work appears to imply that GAs are an appropriate tool to address the architectural topology problem.

Draber [42] has produced a "consistent strategy for the configuration-variant dependability problem." Draber's approach employs a set of restrictions on the configuration solution space based on the results of a Failure Modes and Effects Analysis (FMEA),

and a set of rules determined by the implicit interdependencies between task configurations that must hold if the global configuration is to make sense. The evaluation criteria for each proposed global configuration is based on a set of Markovian reliability models and the cost of hardware per unit.

The approach by Draber is interesting in that it allows certain intra-task and inter-task configurations to be explicitly ruled out. Together these configuration restrictions define the admissible search space. However, Draber’s approach relies on the ability to restrict the search space sufficiently so that an exhaustive enumerative search can be employed to determine the optimal solution amongst the remaining admissible configurations. If this assumption holds this appears to be an appropriate approach. It is easy to envisage circumstances during the design process under which it is not possible, or desirable, to constrain the solution space to this extent. In this case some form of implicit enumerative search technique is required. Elements of Draber’s work can be applied during a GA search.

Programming Environments

Filho et al [50] consider a number of different GA programming environments, which they break down into three categories. Application oriented systems are essentially “black boxes” that hide the activities of the GA solver from the user. Algorithm-oriented systems support the solution of particular GAs. They are either algorithm specific or library based. Finally, toolkits provide many GA programming utilities.

A toolkit, *GAmeter* [105], has been chosen for this thesis. *GAmeter* provides an environment for the development of GA based optimisation tools. Each potential solution in *GAmeterV1.5* is characterised by a bit-string. The *GAmeter* environment has three operators: select, create and merge. All the standard variants of the genetic operators discussed in Chapter 2 can be employed in *GAmeter*. More than one stopping criteria may be applied. A sister tool *SAmson* allows the same problem to be investigated using Simulated Annealing.

GAmeter provides a set of ingredients from which a recipe to solve a particular problem can be chosen. It also provides the ability to ‘cook’ the chosen recipe to select the best architectural topology, in the form of a GA. However, producing a particular recipe from the given ingredients is non trivial. Not only do the ingredients have to be included in the correct proportion (in the form of an evaluation function and accompanying weighting factors) but herbs and garnish (in the form of problem specific elements and heuristics) added. For the architectural topology problem the ingredients and garnish that need to be coded for each problem include:

- set of tasks for which an architectural topology is to be selected

- connections between tasks in the system
- set of admissible topologies
- how to produce new topologies from existing topologies
- set of dependability models and which topologies they ‘go with’
- dependability measures
- evaluation function and weighting factors

The recipes employed for different problems are likely to be similar in form. To extend the analogy: many variants on pasta in a red sauce can be envisaged. The basics however remain the same. All recipes are likely to require considerable coding effort. This effort falls as the design process continues. Re-evaluation of the appropriateness of the current ‘recipe’ is required each iteration.

Search Space Evolution

In Chapter 2 the concept of a hybrid GA and TS was introduced. The aim of such a hybrid is to allow restrictions to be applied to a search space in a structured manner so that disruption to the search space is minimised.

A tabu list can be applied to determine when to penalise particular alternatives and when to make them inadmissible. The evaluation function is extended to *penalise inadmissible alternatives*. The GA is amended as follows:

1. Each restricted alternative has a tabu list element indicating the iteration it will next be admissible.
2. The tabu status of each alternative is determined
 - If a restricted alternative is chosen and the iteration number is less than that shown in the tabu list the penalise inadmissible alternatives element is employed.
 - If a restricted alternative is chosen and the iteration number is **greater** than that shown in the tabu list the penalise inadmissible alternatives element is not employed.
3. The set of tabu restrictions is updated by setting the tabu value for the selected element to be the current iteration number plus *tabu*. *Tabu* can take on any positive integer value up to the maximum number of iterations allowed in the search. It is fixed for any particular run of the algorithm.
4. Repeat steps 1 to 3 for each iteration of the GA.

3.3.1 X-Topmeter Tool

A tool, X-Topmeter, that supports resolution of the architectural topology problem during the topology selection process has been produced. This tool is based on the X-GAmeter environment. It presents a set of possible topologies based on the data given by the designer. It takes note of any restrictions to the quantified design space and the form of the evaluation function. An ATP formulation is assumed. However, a SAT based GA could be produced within the environment.

Elements of the Painton & Campbell [125] paper are used in X-Topmeter. For instance X-Topmeter uses an approach based on the failure distribution function of combinations of units in series and in parallel. X-Topmeter however is more flexible in that it provides many more options such as Markovian reliability models, decision mechanisms and up to 27 different combinations of resources to implement each task.

The assumptions employed in Coit & Smith [31] are very constrictive to the designer. For instance all redundancy is active (hot sparing). In X-Topmeter a library of reliability models is employed to allow the designer to overcome these limitations by constructing appropriate reliability models for the search space options under consideration.

This Section is divided into the following topics:

1. Prototyping Process
2. Input data
3. Search space
4. Traversing the search space
5. Evaluation function and extensions
6. Results from X-Topmeter

The exposition is based on the concepts introduced earlier in this thesis. The aim is to focus on implementation issues.

Prototyping Process

The prototyping process for the baseline X-Topmeter tool was split into two phases: production of an architectural topology “*example generation program*” and incorporation of this program into a GA. The example generation program employs seven steps to produce an example architectural topology for a given control problem:

1. Produce a DAG representation of the logical architecture of the system

2. Decompose this DAG into a set of service DAGs
3. Determine a set of hardware and software resources
4. Decompose the service DAGs into a set of task DAGs and determine the set of resources that can be used in each task
5. Determine a set of architectural components and assign one to each task
6. Determine a set of network topologies
7. Produce an example architectural topology by combing results of steps 2 to 6.

Previous work by the author [117] employed a DAG of precedence ordered processes. This work was extended to incorporate sensing and activation elements of a system. At the architectural level, when the architectural topology problem is first addressed, such a detailed DAG is unlikely to be available. A number of different abstractions of this DAG were formulated and attempts made to automatically produce example services. In the end a model incorporating a hierarchy of three levels (tasks, services and system) was chosen, see Figure 1.3. Thus, in step 2 a simple service DAG consisting of three tasks and one (or three) network(s) is produced.

A topology employs a set of hardware and software units. Given the simple representation of a topology to be used at the architectural level it was not possible to accurately predict the required number of processing hardware and software units. In order to accurately predict the size of a topology an allocation of software units to hardware is required. This allocation problem is addressed in Chapter 4. At the architectural level a set of resources, employed to produce a reliable system, can be determined. A simple utilisation measure may be employed to indicate the minimum number of processing units required in a proposed architectural topology.

In step 3 the example generation program produces a set of hardware and software resource data sheets. Particular attention was given to producing plausible failure rates, speed and cost values. For instance, industrial partners indicated that sensors tend to be an order of magnitude less reliable than processors or actuators. Despite extensive literature searches accurate failure rate figures remain elusive. However, industrial partners have indicated that they keep such information.

In step 4 service DAGs are decomposed into three tasks and a set of admissible resources for these tasks produced. This proved to be a relatively easy process. Particular care was taken to ensure that processing resources could not be *assigned* to sensing or action tasks.

Step 5 proved to be one of the most difficult. A set of architectural components that could be automatically produced and evaluated for any task was required. A number of

different components were formulated. The approach used in the generation program was to have a look-up table for each task that determined the set of components it could employ. A component was chosen at random and a second table used to assign resources to it.

In step 6 a set of network topologies are produced. Modelling networks caused some difficulty as the independence assumption used for tasks does not hold. Multiple services can employ the same communication network. The final version of the generation program built network topologies in a similar manner to that employed to produce tasks. Each network is modelled as a separate component that forms part of the service model. For simplicity it was assumed that any software used by a network formed part of the processing tasks in the system. It is clear that modelling networks and incorporating them into a more expressive service model is one of the key areas of future work for X-Topmeter.

Once the ability to generate an example problem and a solution had been sufficiently refined only two elements of X-Topmeter needed to be produced. First, an ability to evaluate each proposed topology. The example generation program was used as a basis for this extension. Each element in the look-up table of architectural components for a task was extended to point to an appropriate reliability model to allow task reliability predictions to be made. Similar extensions were made to networks. A wide variety of models were employed. In the end RBD and Markov models were chosen. RBDs are used to model simple architectural components and Markovs to model complex components. These models were then composed into a simple service reliability model to produce predictions of service reliability.

Second, an ability to generate a set of new solutions in order to traverse the design space for an architectural topology. A GA has been chosen to accomplish this. Thus, the current X-Topmeter tool employs a GA, elements of the example generation program and a set of reliability models to aid resolution of the architectural topology problem. In the remainder of this Section more details on the baseline version of X-Topmeter are presented and illustrative examples produced. This version is the result of extensive prototyping. A number of further extensions can be envisaged, see Chapter 5.

Input Data

X-Topmeter makes explicit use of data libraries that contain data on a set of admissible resources. Data libraries facilitate reuse of data about the dependability and cost characteristics of resources within and between projects. Reuse is a feature of modern control systems.

The input file for X-Topmeter contains five sets of data structures: Hardware resources, Software resources, Tasks, Services, and Networks, see Table 3.2. Details of these

structures are introduced in Appendix A and the exposition in the remainder of this chapter draws on these structures.

Structure	Elements
Hardware resources	id, type, failure probability per unit time, speed in items (instructions or bits) per unit time, cost of ownership for first unit and % marginal cost of extra units, capacity constraints
Software resources	id, type: 0=logical, 1=variant, 2=decision, 3=comms, failure probability per unit time, computation time (in instructions), period, ownership cost per unit
Tasks	id, number of time units task must remain operational, tabu value, type of task: 0=sensing, 1=processing, 2=communicating, 3=action, 4=decision, number of each of three named hardware resources employed, number of each of three named software resources employed, service task belongs to, bytes of messages sent from task per activation, number of bits used to represent a task in GA
Services	id, number of tasks in service, set of tasks in service, period of service, number of time units service must remain operational
Networks	id, proportion of multiprocessor load inter-processor, type of network, number of three named hardware resources employed, number of time units network must remain operational, number of bits to represent networks in GA

Table 3.2: Data Structures

In Table 3.3 initial values that define the way X-Topmeter traverses the design space are introduced.

GA Parameters	Value
Task (network) search space restrictions employed, 5 available	0 or 1 (0 or 1)
Weighting factor for predicted reliability of tasks in $f(x)$	1
Weighting factor for predicted number of hardware units in $f(x)$	10
Weighting factor for predicted cost of topology in $f(x)$	0.1
Penalty for attempting to choose tabu task (network) configuration	100000 (1000000)
Iterations each inadmissible task (network) configuration is tabu	20 (20)
Number of iterations each inadmissible network is tabu	20
Multiplier factor for processor load. A 2 indicates each processor has a Maximum utilisation of 50 %	2

Table 3.3: Search Space Data

Five search space restrictions are available in the prototype X-Topmeter:

1. At least one admissible resource per task. Maximum number of units is six.
2. Redundancy must be employed in all tasks. Inadmissible components are transformed to admissible components.
3. Redundancy must be employed in all tasks. Inadmissible components are penalised.
4. Redundancy must be employed in all tasks. Inadmissible components are made *tabu* for tabu evaluations each time they are encountered.
5. Sensing tasks are restricted to be identical in form to action task topologies for each service. Inadmissible topologies are transformed to admissible topologies.

Network topologies can be restricted in a similar manner. The evaluation function elements determine the set of weightings to be employed.

Search Space

X-GAmeterV1.5 is only able to manipulate a problem formulated as a bit-string. So, in X-TopmeterV1.5 eight bits are employed to represent the search space for each task

- two bits to determine number of units of hardware R_1 used (0-3)
- two bits to determine number of units of hardware R_2 used (0-3)

- two bits to determine number of units of hardware R_3 used (0-3)
- two bits to determine combination of sparing and decision mechanisms employed

Bits 1-2, 3-4 and 5-6 indicate that between zero and three units of resources R_1 , R_2 and R_3 can be employed per task. A bit string of 00 00 00 ** is always transformed into a bit string of 01 00 00 **. A * indicates that a 0 or a 1 may be present. The combination of values the first six bits can take is limited by the requirement that a maximum of six units be employed for any task. Any bits outside this range are transformed to the string 10 10 10 ** indicating two copies of each task resource are employed. The bit strings deemed to be outside the quantified design space are

111111 ** | 111110 ** | 111011 ** | 101111 ** | 111101 ** | 111010 ** | 101011 ** | 101110 ** |

Bits 7 and 8 indicate whether no sparing and decision mechanism (00), cold spares and switch (01), hot spares and no voter (10), or hot spares and voter (11) are employed. The total number of options per task is therefore $2^8 - 8 = 256 - 8 = 248$.

The design space for each network is limited to six bits as decision mechanisms are not employed; that is:

- two bits to determine number of units of hardware N_1 used
- two bits to determine number of units of hardware N_2 used
- two bits to determine number of units of hardware N_3 used

The number of units per network is restricted to six. Thus the total number of network topologies in the quantified design space in X-Topmeter is $2^6 - 8 = 56$. The total number of alternatives per service, with a single network, is $248^3 * 56$, that is 8×10^8 . A representative bitstring for a single task and network is shown in Figure 3.7.

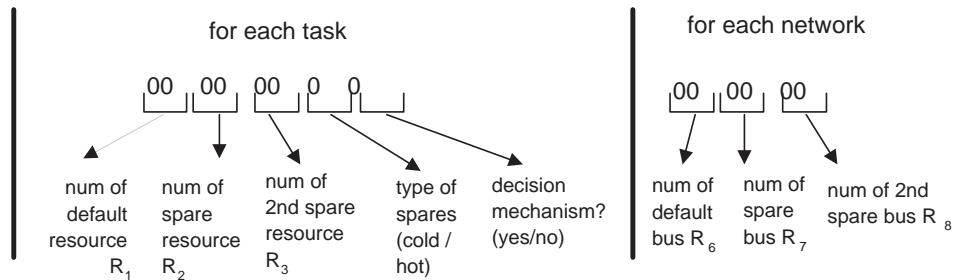


Figure 3.7: Bit String for a Task and a Network

To represent a single service and single network topology ($3 * 8 + 6$) 30 bits are required. If three networks are employed this rises to ($3 * 8 + 3 * 6$) 42 bits.

Traversal of the Design Space

In Table 3.4 the parameter set for the default X-TopmeterV1.5 search is presented. An initial pool, P, of 10 solutions is generated randomly. The four genetic operators used to produce new solutions from the members of the population pool are starred. For more details see the GAmeter manual [104] and Section 2.7.

Parameters	Value
Crossover (Mutate) Probability	60 (1.00)*
Pool Size	10
Minimum (Maximum) number of parents	2 (10)
Best fit inaccuracy	3
Selection mechanism	tournament*
Crossover mechanism	2-point*
Merge mechanism	best fit*
Number of Crossover points	2
Maximum time, generation, no change limits	7200,1200,200

Table 3.4: X-TopmeterV1.5 Parameter Set

A crossover probability of 60% implies that each solution in P has a 60% chance it will undergo crossover with another solution per iteration. The best-fit operator used is slightly non-standard in that a mechanism to introduce ‘noise’ is employed. The *inaccuracy percentage* mechanism employs a 3% chance that a solution will be allowed to enter the solution pool regardless of its evaluation function value. This noise mechanism is used to help maintain diversity and resist the effects of deception. The standard bit-flip mutation operator is employed.

Evaluation Function

The default X-Topmeter evaluation function is:

$$V_{topology} = F(x) + G(x) = (k_c * cost) + (k_s * size)^2 + (k_{rel} * mttf)^2$$

This evaluation function encapsulates information about a number of characteristics of each topology in a single measure.

The reliability of each service is measured in terms of the predicted service MTTF. A positive value of mttf indicates that one or more services have failed to meet their reliability target. Note, the reliability term is a square term. It is generally accepted in the GA community that a square term for constraints is appropriate. The set of

reliability models employed is based on the exposition in Chapter 2 and Appendix A. The SHARPE tool is called by X-Topmeter to determine the predicted MTTF of each task, network and hence service in a topology.

The *cost* of a topology is determined by the number of units of each type of resource employed in a proposed topology. Both hardware and software may be subjected to replication and diversity of implementation. The cost of a topology is given by:

$$Cost_{topology} = Cost_{hardware} + Cost_{software}$$

The cost of the hardware units employed in the minimum size platform implied by a given architectural topology is calculated as the sum of the number of each type of hardware resource employed times the purchase plus ownership cost of each hardware resource. That is:

$$Cost_{hardware} = \sum_h n_h * (Cost_{h,purchase} + Cost_{h,ownership})$$

where h is a hardware resource and n_h is the number of units of h employed.

Ownership costs may include the cost of maintenance and replacement of the unit during the lifetime of the system. It reflects an interaction with another design issue, that of supporting the system during its life time. Thus upgrade costs may also be factored in. In X-Topmeter ownership costs have been formulated as the average cost per unit time of activation of the system it is incorporated into.

The cost of the software resources employed must also be calculated. The cost of multiple copies of the same software is the same as that of a single copy. However, software that employs software based fault tolerance will cost more to develop than software that is not fault tolerant. For instance, production of diverse variants of software incurs extra design or purchase costs. Laprie [90] has investigated the effect on cost of employing a range of fault-tolerance techniques, see Table 3.5. The overall cost of the software employed in a topology is (Laprie [90]):

$$Cost_{software} = \sum_{SW} C_{FT}/C_{NFT} \times N_{sw}$$

where

C_{FT} = cost of fault-tolerant software,

C_{NFT} = cost of non fault-tolerant software

N_{sw} = number of variants of a software resource

SW = number of software resources

The minimum size of the computing platform is calculated using a *parts count* formula:

$$size(x) = \sum_{se=1}^{SE} N_{se} + \sum_{p=1}^P N_p + \sum_{b=1}^B N_b + \sum_{a=1}^A N_a$$

No. Faults	Component	N	Min	Max	Average
			C_{FT} / C_{NFT}	C_{FT} / C_{NFT}	C_{FT} / C_{NFT}
1	Recovery block	2	1.33	2.17	1.75
1	Acceptance check	2	1.33	2.17	1.75
2	Comparison	4	2.24	3.77	3.01
1	NVP	3	1.78	2.71	2.25

Table 3.5: Cost of N-Version Software

where sensors= s , actuators = a , buses= b and processors= p .

Calculation of the number of sensors and actuators employed is simply a summation of the number of units used in each task topology. Calculation of the number of processors, N_p , and buses, N_b , is a function of the utilisation of these resources. Utilisation figure thresholds are taken from existing scheduling theory. The utilisation of processing resource p is:

$$U(p) = \sum_{sw=1}^{SW} I_{sw} / (R_p * T_{sw})$$

where

sw = software module

R_p = speed of processor p in instructions per unit time

T_{sw} = period of software module

I_{sw} = number of instructions per iteration of software module.

The utilisation of a bus is calculated in a similar manner. Instead of instructions, bytes of data per second are used. The processor utilisation threshold is 50% and bus threshold 33%. So for instance, a total utilisation of a processing resource of 126% indicates that at least three units of the resource will be required in the hardware platform. These utilisation figures are taken from industrial standards and existing scheduling theory.

The weighting factors are given default values of $k_c=1$, $k_s=100$ and $k_{rel}=0.1$. These values provide a starting point and have been found by the author to be effective over a number of problems.

Additional elements can be added to the default penalty function. For instance, suppose that the designer restricts the search space. Undesired bitstring values can be penalised by adding the following generic function element

$$k_{red} * (num_{inadmissible})^2$$

This element can be used in conjunction with TS elements. The weighting factor for

this element, k_{red} , aims to ensure a low probability of a restricted solution joining the solution pool.

Results from X-Topmeter

X-Topmeter produces three primary forms of output. First, a graph window that displays a plot of the statistics stored during an experiment. The default setting is best fitness on the y-axis against number of generations on the x-axis. Second, a statistics window is used to display data on the GA in progress. Finally, at the end of the experiment an accumulated set of statistics can be saved to a file. The designer can also save details of the best solution in a file. The data placed in this file is a user defined function coded, in this case, by the designer.

3.4 X-Topmeter Illustrative Examples

In this Section three examples are presented to give a flavour of the approach employed to aid resolution of the architectural topology problem. These examples are also employed in Chapter 4 to illustrate the allocation tool. Example one is a simple one Service system. Results of this example are contrasted against a brute force search undertaken on the same example. Example two is a two service example taken from [116]. Example three is a five service IMA network. A set of design scenarios is introduced for this example.

3.4.1 Illustrative Example One

In Figure 3.8 a DAG represents the logical architecture of a single service control application. The parameter set employed for the GA search is also presented. For more details see Table 3.4 and Appendix A. This example consists of three tasks and a single IMA network.

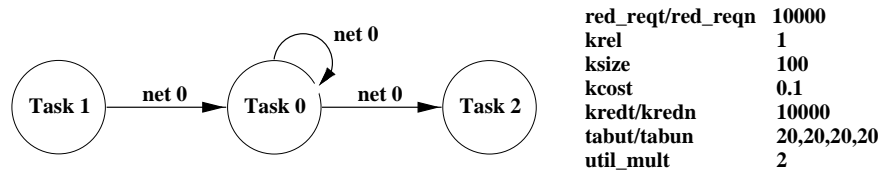


Figure 3.8: Three Task Model

The hardware library for this example is a subset of a larger library. The set of possible hardware is 0-2, 5-8, 10-12 and 15-17, see Table 3.6. A value of -1 indicates no value is set. The coverage factor indicates the probability of detecting and recovering from a

failure when no decision mechanism is used, an acceptance test is used and a majority voter is used.

id 0 type proc FR .000017 Coverage .610,.721,.963 speed 100000 cost 978,.16 capacity 16	id 1 type proc FR .000015 Coverage .651,.717,.984 speed 137000 cost 195,.45 capacity 32	id 2 type proc FR .000013 Coverage .670,.728,.973 speed 152000 cost 1434,.20 capacity 16
id 5 type bus FR .000001 Coverage .574,.659,.845 speed 1030000 cost 1500,.05 capacity 14	id 7 type bus FR .00000053 Coverage .558,.642,.895 speed 1100000 cost 1700,.03 capacity 8	id 8 type bus FR .00000063 Coverage .529,.602,.835 speed 1200000 cost 1800,.02 capacity 14
id 10 type sensor FR .000017 Coverage .5,.75,.95 speed -1 cost 1000,.05	id 11 type sensor FR .000051 Coverage .5,.75,.95 speed -1 cost 1100,.04	id 12 type sensor FR .000013 Coverage .5, .75,.95 speed -1 cost 1200,.03
id 15 type actuator FR .000017 Coverage .45,.7,.9 cost 500, .05	id 16 type actuator FR .000015 Coverage .45,.7,.9 cost 600,0.04	id 17 type actuator FR .000013 Coverage 0.45,0.7,0.9 cost 700,0.03

Table 3.6: Hardware Resources for Example 1

The software library consists of three alternative software resources, see Table 3.7. Two decision software resources representing acceptance and voting mechanisms respectively are also indicated.

id 0 type logical FR .0000001 variants 1,2 C 1833 period .075 cost 1377	id 1 type variant FR .0000001 variants -1 C 1900 period .075 cost 1924	id 2 type variant FR 0.000001 variant -1 C 1800 period .075 cost 924
id 3 type test FR .0000001 C 100 period .075 cost 1000	id 4 type voter FR .0000008 C 200 period .075 cost 2000	

Table 3.7: Software Resources for Example 1

Details of the three application tasks are given in Table 3.8. Task structures are also

employed to store details of the decision mechanisms employed.

id	0	id	1	id	2
tabu	0,0,0,0,0	tabu	0,0,0,0,0	tabu	0,0,0,0,0
type	processing	type	sensing	type	action
hardware	0	hardware	10	hardware	15
software	0	software	-1	software	-1
spares	1,2	spares	11,12	spares	16,17
variants	1,2	variants	-1	variants	-1
operational	12000	operational	12000	operational	12000
service	0	service	0	service	0
mes_total	0,300,300	mes_total	300,0,0	mes_total	0,0,300
no_bits	8	no_bits	8	no_bits	8
id	3	id	4		
tabu	0,0,0,0,0	tabu	0,0,0,0,0		
type	decision	type	decision		
hardware	0	hardware	0		
software	3	software	4		
spares	1,2	spares	1,2		
operational	12000	operational	12000		

Table 3.8: Task Data for Example 1

In Table 3.9 data for the service and network are presented. The network is able to employ hardware 15, 16 and /or 17. Note that the service has tasks ordered 1,0,2. The order tasks are presented in each service is important (sensing, processing, action); the numbering convention is not.

service	0	network	0
no_tasks	3	tabu	0,0,0,0
deadline	.075	hardware	5
period	.075	spares	7,8
operational	12000	operational	12000
tasks	1,0,2	no_bits	6

Table 3.9: Service and Network Data for Example 1

Three X-Topmeter experiments were undertaken for the default parameter settings. The experiment lasted for 17 Steps. Multiple experiments are required as the underlying GA is non-deterministic and therefore cannot be guaranteed to converge to a *good* solution on each experiment at an acceptable speed. Statistics for those steps where an improvement occurred are shown in Table 3.10.

The best evaluation function value found is 4863 from a bit string with the following integer values:

$$task1|task0|task2||net0 = 0, 0, 3, 3|0, 0, 2, 0|3, 0, 0, 2||2, 0, 0$$

Steps	Generations	Time	Evaluations	Fitness	Average Fit
1	1	0.00	10	6358199	49577388.10
2	2	11.00	19	6060391	20568495.80
3	4	32.00	33	6028802	6103524.40
4	8	53.00	46	5998385	6035237.00
5	17	102.00	91	5995435	5998090.00
6	21	146.00	115	5994937	5995385.20
7	22	154.00	118	5994902	5995331.90
8	39	293.00	189	4808260	5876237.80
9	45	314.00	207	8890	4328323.00
10	51	341.00	234	7950	8796.00
11	53	352.00	246	7550	8148.00
12	58	364.00	268	6670	7462.00
13	63	376.00	290	6199	6622.90
14	64	377.00	295	5810	6442.70
15	71	387.00	321	5790	5808.00
16	79	395.00	342	5230	5734.00
17	214	576.00	824	4863	5193.30

Table 3.10: Results for Best Experiment for One Service Example

X-Topmeter took 10 minutes and 214 generations to find the best solution. The evaluation function result for the best solution is shown below. This is the standard format for X-Topmeter output files. It indicates that all task, network and service MTTF requirements were met. A platform of at least 10 hardware units is predicted.

```

mttf fitness: task= 0.00
number bytes to be transported=24000.0
bandwidth = [.0699, .0655, .0600] network fitness= 0.00
mttf service 0=12242.0 service fitness= 0.00
util (num) of hware 2 = 0.316 (2)
util (num) of hware 5 = 0.000 (2)
util (num) of hware 12 = 0.000 (3)
util (num) of hware 15 = 0.000 (3)
cost fitness: hardware= 3761.40, software= 292.40
overall fitness = 4863.00

```

The best topology consists of a sensing task that uses three copies of sensor 12 and a sensor fusion voting mechanism¹; a processing task that uses two copies of processor 2; and an action task that employs three copies of actuator 15 in parallel. The utilisation figures imply a single type 2 processor. However, since two parallel copies of the processor are employed in task 0 at least two processor 2 units are required in the platform. X-Topmeter takes the larger number. The IMA communication network employs two copies of hardware 5 in parallel.

¹A sensor fusion mechanism processes sensor data in much the same way as a voter processes control data output.

Brute Force Search

The illustrative example introduced above was subjected to a brute force search to show the size of the search space and to give an impression of the time to solve a problem using enumerative techniques. Each bitstring is tried in turn. The first eight alternatives in integer form are:

(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0) (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2) (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0) (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1)
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2) (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3)

The search took three weeks to run on a sun sparc classic workstation. The best evaluation function value was 4859.9. Thus, the GA found a solution within 0.062% of the optimal solution in 0.03% of the cpu time. The best solution for this example in integer form is 0200 0033 3000 110.

In Figure 3.9 results of the X-Topmeter and bruteforce searches are shown in block diagram form. Blocks placed left-to-right represent resources placed in *series* and blocks stacked vertically represent resources in *parallel*. The block diagrams to the left of the vertical bars represent task topologies and those to the right the proposed topology for the IMA network. Data is passed between, and within, each task via the network. The form of topology employed by both the X-Topmeter and Bruteforce experiments is identical. However, the processing tasks use different processors, and the X-Topmeter network swaps one resource 5 bus for a type 7 bus.

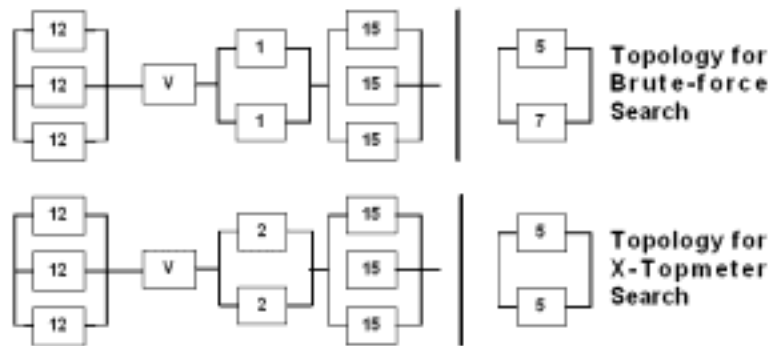


Figure 3.9: X-Topmeter Versus Bruteforce

3.4.2 Illustrative Example 2

Consider a hypothetical control system for which an initial functional decomposition has occurred and two services identified. The two services are modelled using a simple

three tasks representation, see Figure 3.10. Three networks are employed: sensors to processors, processors to processors, and processors to actuators. Design data including periods and MTTF requirements for each service are shown in Table 3.11.

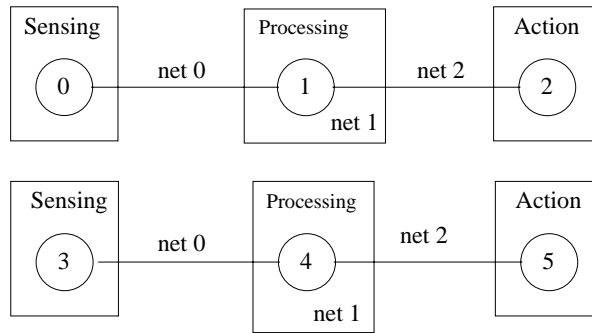


Figure 3.10: Services for Illustrative Example 1

Service Id	Period	Target MTTF
0	0.005 secs	12000 secs
1	0.006 secs	12000 secs

Task Id	Type	Message size	Target MTTF
0	0	42,0,0	14000 secs
1	1	0,10,42	12000 secs
2	2	0,0,0	10000 secs
3	0	60,0,0	10000 secs
4	1	0,100,85	12000 secs
5	2	0,0,0	10000 secs

Table 3.11: Library Data for Services and Tasks

A data library of admissible resources, see Table 3.12, has been produced. A dash indicates that this characteristic is not relevant for the identified resource.

To represent a set of architectural components for this problem 66 bits are required. The integer version for a typical string is:

$$task0|task1|...|task5||network0|net1|net2 = 1, 2, 0, 2|2, 0, 0, 0|3, 0, 0, 1|2, 0, 0, 2|0, 1, 1, 0|2, 0, 0, 0||1, 1, 1|1, 1, 1|1, 1, 1$$

In this example task 0 employs one copy of resource 1, two copies of resource 2 and no decision mechanism. The first element to produce a result is used downstream.

Results for a variety of bit strings for one task with different resources and copies of the resources under a perfect coverage assumption are shown in Table 3.13. In each quarter of the Table a sequence of models is shown. The model with the lowest ability

Id	Type	FR	Coverage	Speed	Cost
0	sensor	.000031	0.642, 0.707, 0.925	107000	527, .006
1	sensor	.000046	0.667, 0.767, 0.969	166000	785, .009
2	sensor	.00019	0.674, 0.759, 0.945	187000	838, .048
4	processor	.00017	0.610, 0.721, 0.963	137000	978, .045
5	processor	.00014	0.651, 0.717, 0.984	152000	195, .016
6	processor	.00028	0.670, 0.728, 0.973	128000	434, .015
8	bus	.000076	0.681, 0.744, 0.914	1030000	501, .029
9	bus	.00013	0.658, 0.742, 0.995	1023000	454, .016
10	bus	.0000045	0.629, 0.702, 0.935	1047000	318, .029
12	actuator	.0000082	0.622, 0.736, 0.945	116000	631, .0073
13	actuator	.000025	0.659, 0.753, 0.900	186000	279, .013
14	actuator	.000023	0.648, 0.783, 0.955	136000	570, .053
12	software	.000018	-	-	1616
13	software	.000010	-	-	1962
14	software	.000014	-	-	1072
15	variant	.000015	-	-	1720
16	variant	.000010	-	-	1865
17	variant	.000010	-	-	2121

Table 3.12: Library Data for Resources

to detect and tolerate a fault is shown at the top and the model with the most fault tolerance at the bottom. Intuitively, the first model should have the lowest rank and the last model the highest rank. If coverage is assumed to be perfect for all of these options this is not always the case. This is because fault tolerance has been purchased at the cost of extra units that may fail, and hence reduce reliability.

To gain a more accurate measure of the ability of a topology to detect and tolerate faults the MTTF measure is adjusted by using different coverage factors for the situation where no decision mechanism is used, an acceptance test is used and a majority voter is used. The Nuclear industry employs a similar approach, called *Beta* factors. Coverage factors are employed in all three examples presented in this chapter.

Model	String	MTTF	Rank	Model	String	MTTF	Rank
simple	1000	16290	25	parallel1	1110	20580	23
accept1	1001	21020	21	standbyu	1101	35970	10
parallel	2000	24440	20	accept1	1103	35310	11
majn	2003	40560	7	parallel	2202	28790	16
parallel	3000	29870	14	standbyu2	2201	37770	9
binomial	3000	13580	-	majn	2203	49520	3
standbyu2	3001	38770	6	parallel	3301	34020	13
majn	3003	48920	4	standbyu2	3302	42810	5
				majn	3303	58100	1

Model	String	MTTF	Rank	Model	String	MTTF	Rank
simple	0100	10770	28	parallel	1110	20700	22
accept1	0101	14000	27	standbyu3	1111	28500	17
parallel	0200	16160	26	accept3	1113	36060	9
majn	0203	28440	18	parallel2	2220	28830	15
parallel	0300	19750	24	standbyu3	2221	38630	7
standbyu2	0301	25800	19	maj3	2223	49610	2
majn	0303	34330	12				

Table 3.13: MTTF for Task 1

Results for a typical experiment are shown in Table 3.14. Parameters of the GA are the default settings, except for the mutation probability, which is set to 5%. The experiment lasted for 38 Steps, the first and last five of which are shown. A further four experiments with the same parameters gave the following best fitness values: 6586, 7007, 9960, 6721. The average fitness value for the five experiments was 7370.

Steps	Generations	Time	Evals	Fitness	Average Fit of Pool
1	1	0.00	10	15002246.0	59620922.7
2	5	183.00	34	8924752.7	14417004.1
3	6	221.00	42	7377400.0	13362878.8
4	7	275.00	49	4281630.2	9151211.6
5	10	372.00	63	3687291.9	5596778.3
34	177	4892.00	1108	7143.3	7501.5
35	232	5204.00	1417	7036.1	7132.6
36	266	5445.00	1642	6782.8	7131.4
37	412	6284.00	2484	6757.6	6945.9
38	415	6297.00	2498	6586.6	6760.7

Table 3.14: Results for 2 Service Example

The best fitness function value of 6586 results from a string with the following integer values:

$$task0|...|task5||net0|net1|net2 = 1, 2, 0, 1, 0|2, 0, 0, 0, 0|3, 0, 0, 0, 1|2, 0, 0, 1, 0|0, 2, 1, 0, 0|2, 0, 0, 0, 0||1, 1, 1|1, 1, 1|1, 1, 1$$

Both services have predicted MTTF values greater than their requirements; 12020 and 10100 seconds respectively. The cost element value is 1616 units for hardware and 560 units for software. The size element is 4410 indicating a platform of 21 items. Thus the search has found a **feasible** solution, see Figure 3.11.

The solution has just achieved its MTTF requirements for each service. In fact this is not surprising. The penalty function penalises tasks and services that fail to meet their reliability targets heavily. Once a feasible solution has been found, $mttf=0$, the tool selects for topologies with the smallest cost and size. Since reliability is partly a

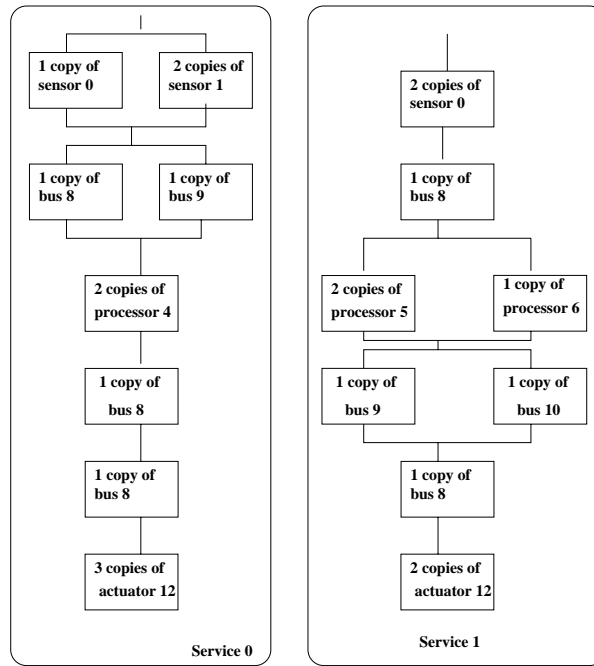


Figure 3.11: Best RBD Formulation

function of the number of copies employed there is an incentive to reduce the number used to a minimum. In other words the tool is formulated so that it will attempt to meet MTTF targets, but minimise the number of resources employed once this is achieved.

In order to assess the quality of the GA implementation a simple *sensitivity analysis* of the parameters of the GA has been undertaken. The parameters investigated are:

1. overall GA parameters: crossover probability, mutation probability, pool size
2. select function: tournament, exponential, proportional (steady-state)
3. create function: two-point crossover, three point crossover, uniform crossover
4. merge function: bestfit, tendency, replace

The baseline model is that used to produce the results shown in Table 3.14. A single parameter is changed and 5 runs for each new search undertaken. For instance, in line one of Table 3.15 a crossover rate of 40%, rather than 60%, is employed. In Table 3.15 the results of this analysis are presented. FE is the number of function evaluations to obtain the best result.

Parameter	value	res1(FE)	res2(FE)	res3(FE)	res4(FE)	res5(FE)
xover prob	40%	6587(2498)	6587(2403)	7007(2469)	18960(4200)	6721(2812)
xover prob	80%	6726(3066)	7201(2765)	6764 (11207)	6750(2820)	8113(741)
mutation prob	0.5%	7106(1894)	7828(1034)	31690(615)	7600(754)	8426(799)
mutation prob	10%	8625(975)	7737(2012)	7247(2497)	10809(545)	8955(1270)
pool size	20	6198(4093)	60037(472)	12355(822)	6555(1953)	8370(2975)
pool size	30	6576(1800)	7584(1334)	8555(2141)	6768(1326)	11529(1275)
selection method	proportional	7444(818)	7733(639)	6721(464)	7159(1028)	9103(1542)
selection method	exponential	8195(1341)	7738(683)	11310(680)	13489(933)	7136(795)
xover method	3	9827(680)	8457(717)	7720(2270)	8241(1109)	9393(1659)
xover method	uniform	11870(490)	11113(708)	7313(2126)	7137(1485)	9609(820)
merge method	tendency	11025(848)	11121(1055)	8544(6642)	6628(8970)	8042(1166)
merge method	replace	7775(1189)	7559(2398)	7034(2929)	8940(761)	7732(1174)

Table 3.15: Sensitivity Analysis

The average fitness values for each of these sets are 9172, 7111 12530, 8675, 18703, 8202, 7632, 9574, 8728, 9406, 9032 and 7808. This indicates a fairly robust search space, although the small size of the example should be noted. The best single result was 6198 produced using a pool of 20 possible solutions. In this case 4093 function evaluations (200 generations) were undertaken. Similarly, an evaluation function value of 6587 was produced for two runs of the tool employing a low crossover probability. In this case the tool converged in 2500 function evaluations (250 generations).

The number of function evaluations required to find the best results varied wildly. It should be noted that the search was truncated by time, so that searches which investigated areas with complex, and hence time consuming, reliability models tended to complete fewer iterations.

Overall the results from this simple sensitivity experiment indicate that parameter settings that maintain diversity and therefore slow convergence are likely to produce better results. This is consistent with results in the literature for combinatorial optimisation problems.

Consider a restriction imposed as a result of a safety requirement that *The computer hardware implementing the design must have redundancy*. This implies that the following bit strings for tasks are removed from the set of admissible components:

$$(01\ 00\ 00\ **)\ (00\ 01\ 00\ **)\ (00\ 00\ 01\ **)$$

One method of dealing with this restriction is to change the inadmissible alternatives into admissible alternatives. That is strings (01 00 00 **) (00 01 00 **) (00 00 01 **) are transformed to strings (10 00 00 **) (00 10 00 **) (00 00 10 **) respectively. This option has been implemented for this illustrative example.

A second alternative is to penalise inadmissible alternatives by adding an extra element to the evaluation function. A third approach is to employ a history function based on TS. The history list is used to determine how often a particular value for a variable can be investigated by the search. Results of five experiments for each alternative, for the standard parameter set, are given in Table 3.16

Parameter	res1 (FE)	res2 (FE)	res3 (FE)	res4 (FE)	res5 (FE)
Add extra copy	10062(1022)	6768(2818)	7613(625)	7914(515)	9611(883)
Penalise results	1933506(91)	12664(1454)	10402(620)	8155 (1810)	11846(1121)
Tabu	8899 (2168)	7504 (799)	8233 (806)	7072 (2968)	8297 (774)

Table 3.16: Removal Experiments

In this example the TS hybrid produces the best results indicating that this hybrid may be an appropriate addition to the X-Topmeter tool. Note that the penalise inadmissible alternatives version led the tool in one case to explore very time consuming options. Since the experiments were time constrained this experiment terminated after only a few generations.

3.4.3 Illustrative Example 3

This example shows the effect on the required Task, Network and hence Service topologies of changes to the reliability requirements for each Service in a system. All other characteristics of the problem remain constant. Consider a distributed control system that must provide five control Services, see Figure 3.12. Tasks 0 to 4 and 7 to 11 may employ acceptance tests or voters. The designer has also determined a library of resources. The resources each task may employ are presented in Table 3.17 and in more detail in Appendix A.

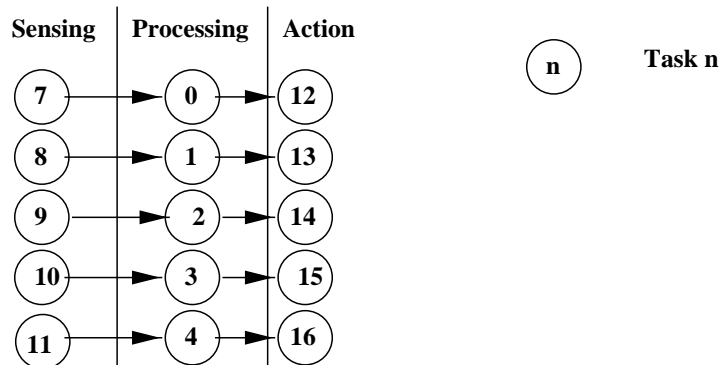


Figure 3.12: Logical Architecture for Example 3

Task	0	1	2	3	4	7	8	9	10	11	12	13	14	15	16	Net0
hware 1	0	1	2	2	1	10	10	10	10	10	15	15	15	15	15	5
hware 2	1	2	3	1	2	11	13	11	12	11	16	16	16	17	17	7
hware 3	2	3	4	3	3	12	14	13	13	14	17	18	19	18	19	8

Table 3.17: Admissible Resources for Example 3

In the early stages of the production of a system the designer may be unsure of the level of reliability that can be assured given the available resources and may therefore wish to investigate the effect of changing reliability requirements on the reliability, cost and size of an acceptable topology. The designer may use the results in discussions to determine acceptable trade-offs between reliability, cost and size of the computer-based system being developed. The designer may also consider other issues, such as maintainability, based on the predicted topology.

A pool of ten solutions, mutation rate of 1% and 2-point crossover rate of 60% are employed in the baseline model. The initial weighting factors are 0.1 for reliability, 100 for cost and 1 for size. The tool terminates if 200 generations have elapsed since a new best solution was found, or if 7200 seconds have elapsed, or if 10000 generations have elapsed.

Three experiments were undertaken for each of the following MTTF targets for the Services in the system: 3000, 5000, 7000, 9000 and 11000 time units, see Figures 3.13 to 3.17. In this illustrative example the tool was most likely to run out of time for high MTTF requirement experiments. This is mainly because the complexity and therefore resolution time of reliability models increases as more redundancy is employed.

Service MTTF target of 3000

The best experiment produced a minimum evaluation function value of 21840 units. All Services are predicted to meet the 3000 MTTF target. The predicted Service MTTFs were 3150, 3920, 3890, 3140, 4490 and the bandwidth on the bus was 0.20.

In Figure 3.13 topologies for the control services are presented. To show the demarcation between tasks a solid vertical line is drawn. The topology for Service 2 for instance, consists of a sensor (implemented on hardware resource 13), a processor (implemented on hardware resource 3) with an acceptance test on the resulting control data, and an actuator (implemented on hardware resource 15). Data is assumed to flow between the boxes along a bus (implemented on hardware resource 5).

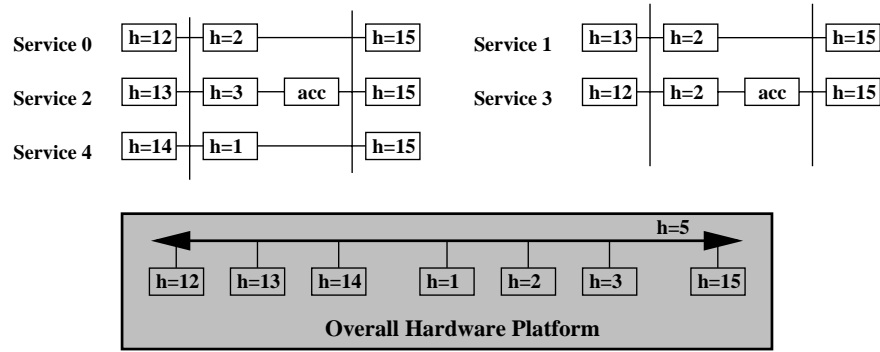


Figure 3.13: MTTF Target of 3000 Units

The overall platform, in terms of the predicted set of hardware resources, is also shown in Figure 3.13. The double arrowhead line indicates a bus resource. A 14 unit hardware platform is predicted:

- 2 copies of sensor resource 12, 2 copies of sensor 13, 1 copy of sensor 14,
- 5 copies of actuator resource 15,
- 1 copy of processor resource 1, 2 copies of processor 2, 1 copy of processor 3, and
- 1 copy of bus resource 5.

In this version of X-Topmeter each instance of a sensor / actuator is assumed to indicate a unique physical hardware unit. Processors and Buses can be partitioned in such a way that more than one Service can have access to the same hardware unit. Thus the number of buses and processors of each type employed is determined by an utilisation formula.

Figure 3.13 shows that Services 0, 1 and 4 employ simplex topologies for all tasks. Services 2 and 3 employ simplex sensing and action tasks, but implement an acceptance test on the processing task results. A simplex IMA network is employed. This simple arrangement is expected due to the small reliability requirement. The GA penalty function in this case has selected a solution primarily by its size and cost.

Service MTTF target of 5000

In the best X-Topmeter experiment the minimum $P(x)$ value found was 35700. All Network and Service MTTF targets were met. The predicted Service MTTFs were 5820, 5020, 5220, 5200, 5220 units respectively and the bandwidth on the bus was 0.26. The size of the predicted hardware platform was 18, see Figure 3.14.

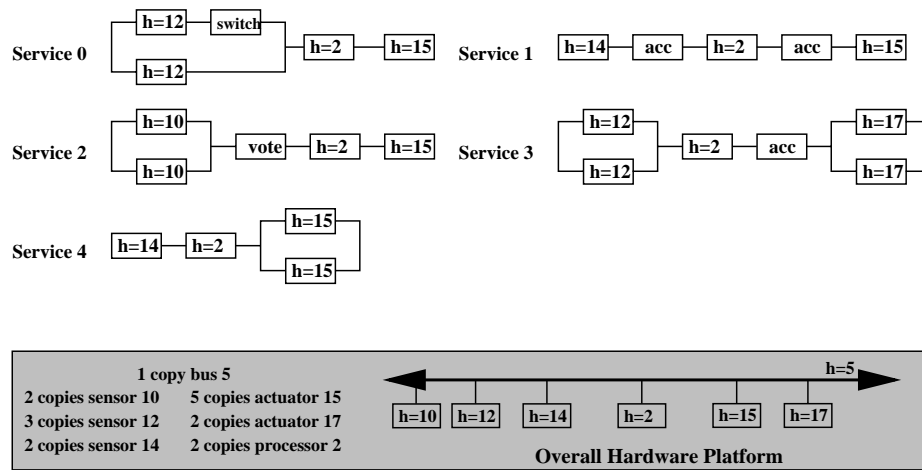


Figure 3.14: MTTF Target of 5000 Units

The tool has increased the complexity of the Service Topologies over those employed for a 3000 MTTF requirement. Service 0 implements a cold spare sensing task topology. Services 3 and 4 implement a parallel action task. In a parallel topology the first actuator to become active is taken. It is assumed that all actuator failures are omission failures. The overall topology is a single IMA bus topology with seven hardware resources. Only a single processor type is used.

Service MTTF target of 7000

In the best X-Topmeter experiment the $P(x)$ value was 52600. All Network and Service MTTF targets were met. The predicted Service MTTFs were 7030, 7030, 7180, 7220, 7190 respectively and the bandwidth on the bus was 0.28. A platform of 21 hardware units is predicted, see Figure 3.15.

The increase in reliability requirements has led to additional replication in the sensing and action tasks. Processing tasks remain relatively simple. This is due to the cost of producing software and the probability of hardware and software failures. A number of task topologies are now somewhat skewed in that clear lanes through a Service are no longer obvious. Restrictions could be placed by the designer to *correct* this feature.

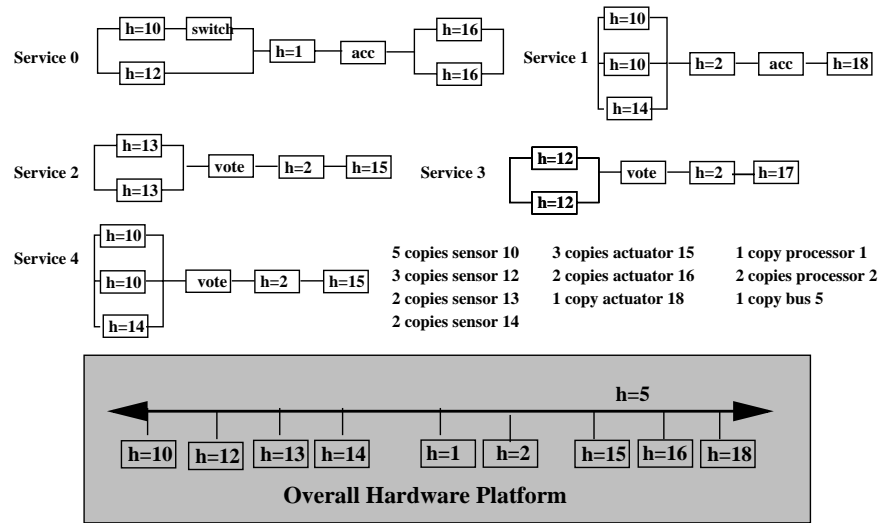


Figure 3.15: MTTF Target of 7000 Units

Service MTTF target of 9000

In the best X-Topmeter experiment the minimum $P(x)$ value was 83170. All Network and Service MTTF targets were met. The predicted Service MTTFs were 9070, 9500, 9230, 9100 respectively and the bandwidth on the bus was 0.47. A platform of 26 hardware units is predicted, see Figure 3.16.

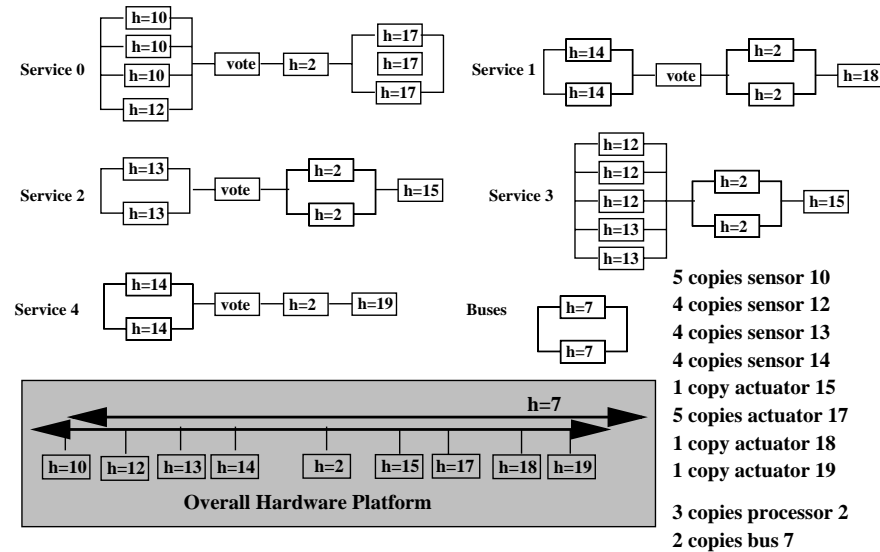


Figure 3.16: MTTF Target of 9000 Units

The increase in replication exhibited previously as reliability requirements rise has continued. Extra data buses can be employed for two reasons. First, to carry the volume of data being sent between tasks. Second, to replicate messages so that a lost

message does not cause an omission failure. In this example two buses are required to carry the volume of data that needs to be transferred between elements in the system. Processor resource 2 remains the favoured processing resource.

As replication increases the tool has swapped to using different resources to implement tasks in the system. This may be due to a cost for reliability trade-off. Once a decision is made to switch to a parallel topology from a simplex topology the reliability of each unit does not need to be the same as for a simplex unit. Since cost is often related to reliability lower cost units with less reliability may be employed in the parallel topology.

Service MTTF target of 11000

In the best X-Topmeter experiment the minimum $P(x)$ value was 269230. The predicted Service MTTFs were 11020, 10050, 11220, 10890 and 11160 respectively and the bandwidth on the buses was 0.63. Services 1 and 3 have failed to meet their reliability target. This implies that there was no acceptable result within the constraints on the search. A platform of 44 hardware units is predicted, see Figure 3.17.

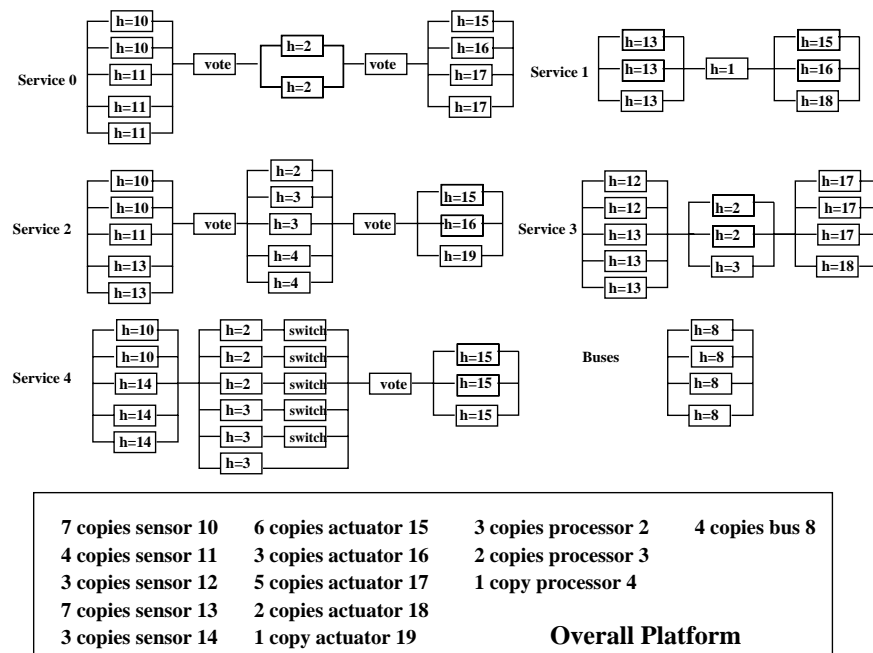


Figure 3.17: MTTF Target of 11000 Units

Data requirements in the system imply two buses should be employed. Both of these buses are replicated. Thus the IMA network employs a single hot spare network topology. Overall the system topology is very large. Most tasks employ redundancy and decision mechanisms to provide appropriate reliability.

Results show that the tool is unable to find a topology that ensures all Services meet a MTTF requirement of more than 11000 units, see Table 3.18. Services 1 and 3 have MTTF times of 10050 and 10910 respectively. This is either because no such topology exists or because the tool has converged to a local, infeasible, topology. The latter would indicate that for high MTTF requirements the tool is unable to traverse the search space effectively. The designer may wish to rerun the tool using parameter settings that emphasise diversity over intensification operators.

MTTF	Fitness	<i>Cost_{hardware}</i>	<i>Cost_{software}</i>	Size	Steps	Evals	Met?
≥3000	21840	13800	85620	14	63	2524	Y
≥5000	35700	18410	14560	18	50	2156	Y
≥7000	52600	22400	19560	21	47	2203	Y
≥9000	83170	31340	16560	26	42	4485	Y
≥11000	269230	52390	16730	45	53	1268	N

Table 3.18: Overall Topology Results for Default Weightings

Hardware library data shows that sensors are an order of magnitude less reliable than processors or actuators. The failure rate of the sensors in the resource library is approximately 1×10^{-5} per hour, whereas the failure rate for processors and actuators is 1×10^{-6} per hour. Therefore, sensing tasks are the first tasks to be subjected to redundancy. Processing is relatively expensive due to the cost of producing software. Hence processing tasks are often subjected to little redundancy. In practice, the designer may wish to balance the search differently. For example, spread software costs over N instances of the system.

Processing tasks are subject to hardware and software failures. This may however be ameliorated by the ability of software to detect and mask failures. The designer can use a Coverage factor to reflect this ability, see Section 2.4.1, and in this case employing software redundancy may become a preferred choice.

The results presented in Figures 3.13 to 3.17 show that topological complexity increases with reliability requirements. Furthermore, it costs 20400 price units more in hardware and 3800 more price units in software to ensure a service reliability of 9000 time units than it costs to ensure a service reliability of 3000 units. The size of the Topology increases from 14 units to 26 units.

Many more copies of resources are used when the reliability requirement is 11000 rather than 3000 units, but the tool has still failed to find an appropriate topology. This result shows the sensitivity of Architectural Topology problems to changes in requirements when using a set of alternate resources with similar reliability characteristics. The designer may wish, in this case, to extend the range of admissible resources to ensure

the required MTTF is attained.

To show the effect of changing Service reliability requirements on the topology model consider Table 3.18 showing results for the overall platform and Figure 3.18 showing the evolution of a Topology for Service 0. For clarity only tasks are shown in Figure 3.18. The thick black vertical lines indicate task boundaries.

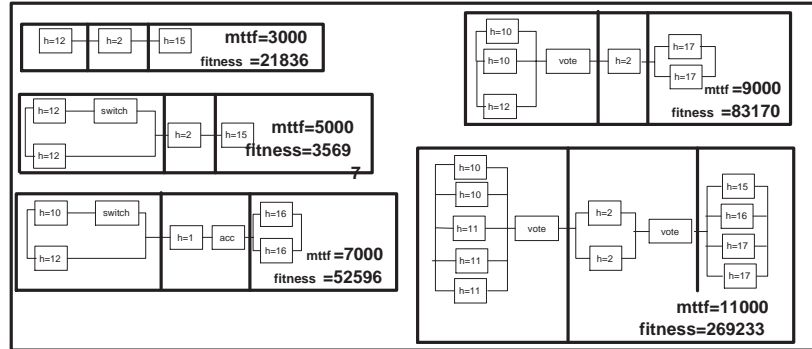


Figure 3.18: Task Topologies for Service 0

The designer can now make decisions based on the Topologies presented by X-Topmeter. A simplex topology is employed to produce a Service topology with a target MTTF of 3000 time units. For the 5000 unit target a cold spare is employed for the Sensing task. The reliability gains from replicating sensors are greater than the gains from replicating processors or actuators. If the reliability requirement is 7000 time units an acceptance test on the processing output and hot spare actuators are also employed. At 9000 time units a TMR sensing task and hot spare action task topology is used. Finally, by 11000 units a TMR sensing task, hot spare with voter processing task and quad redundant actuators are used in an attempt to ensure an acceptable level of reliability.

3.4.4 Design Freedom for the Topology Problem

The designer now has a baseline architectural topology to use during the design process. As design progresses the designer revisits the topology problem to ensure that the baseline topology remains feasible. However, the approach presented in this thesis also allows the designer to set out different design scenarios and predict their impact on the topology. In this Section the following four scenarios show the power and flexibility of the approach:

- Alternative set of resources to the baseline model.
- Effect of emphasising different elements of the evaluation function. For instance, in areas with very severe size restrictions, such as a satellite, the k_s factor can

be increased.

- Restrictions imposed by the designer, including mandating replication.
- Effect of changing the probability of detecting and recovering from a failure.

Alternative Resource Set

The designer has decided, as a result of the analysis undertaken in Section 3.4.3 and subsequent discussions on the maintainability of the control system, to restrict the set of admissible resources to three per task (network) type for the entire system. The set of admissible hardware resources for this topology is shown in Table 3.19.

Task	0-4	7-11	12-16	net 0
hardware 1	0	10	15	5
hardware 2	1	11	16	6
hardware 3	2	12	17	7

Table 3.19: Restricted Resource Set

Results for the five Service MTTF requirements introduced in Section 3.4.3 but changing the resource set, are shown for Service 0 in Figure 3.19. Data relating to the performance of the tool and the predicted system platform is given in Table 3.20.

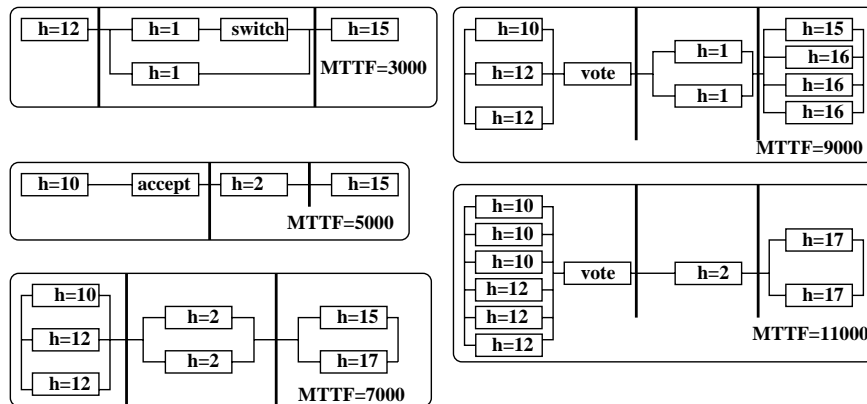


Figure 3.19: Results for Alternate Resource Set

Service 0 meets the required reliability for all five targets. In comparison with the baseline model (Figure 3.18) more replication has been employed. For instance, for the 7000 MTTF target the sensing task employed is triple hot sparing in the restricted example and a single cold spare in the baseline model. The topology implied for the 11000 MTTF is less complex than for the baseline model.

MTTF	Fitness	$Cost_{hardware}$	$Cost_{software}$	Size	Steps	Evalns	Met?
≥ 3000	27700	15400	5560	16	47	1336	Y
≥ 5000	66010	23500	11560	25	40	1554	Y
≥ 7000	96890	30400	16320	30	57	2108	N
≥ 9000	113780	31330	17380	33	57	3258	Y
≥ 11000	202870	42930	18450	44	63	1909	N

Table 3.20: Performance Data for Alternate Resource Set

In this example the restriction has not been entirely successful for the overall system. The tool has a tendency to become ‘stuck’ in local minima. For instance, the tool was unable to find an acceptable topology for all Services if the MTTF requirement is 7000 units. Service 4 was predicted a MTTF of 6950 units. In an attempt to meet the 7000 unit target the size and cost of the predicted hardware platform has increased by 9% and 45% respectively over the baseline model. Note, that since a topology that meets the 9000 target exists, one must exist that can meet the 7000 target. The tool has been deceived. In practice the designer would need to alter other parameters, such as the weighting factors or pool size, in order to find an appropriate solution.

The tool has however done better than the baseline model at finding a topology for Service MTTF requirements of 11000 units. Only Service three has failed to meet this requirement (predicted MTTF of 10940). This is probably partially due to the reduction in the complexity of the search space engendered by reducing the set of admissible resources. The size of the resulting topology is 44 units.

As a result of the analysis undertaken on this reduced resource set scenario the designer may wish to introduce alternative resources with greater reliability. Alternatively a cheaper actuator may allow actuator redundancy to be cost effective. For the 7000 MTTF requirement Service 4 employs three sensors in parallel and two processors with voting, but only a single actuator. The tool has deemed the cost of adding an extra actuator to be prohibitively expensive relative to the gain in reliability for the Service as a whole.

Alternate Weighting Values

In the reduced resource set example a better, lower fitness value, result was found for the 11000 unit MTTF target than for the baseline model. The tool in the baseline search converged to a relatively poor local minimum. The rate at which the tool converges to a solution is partially determined by the set of weighting factors: k_{size} , k_{cost} and k_{rel} . The resulting topology may, in some cases, be highly sensitive to changes in these factors.

One of the areas in which reuse between projects of a similar type may be appropriate is in the setting of weighting factors. Parameters, such as weighting factors, encapsulate the corporate knowledge of designers of particular types of systems on the relative importance of different characteristics and the area of the solution space the tool should be investigating. It is therefore envisaged that the quality of the results will increase as designer knowledge is brought to bear.

To illustrate the effect of weighting factors on the topology produced, the baseline model is subjected to changes in the value of each k factor. The baseline X-Topmeter tool employs weighting factors of 1, 100 and 0.1 for the MTTF, size and cost elements of the penalty function respectively. Three alternative sets of weighting factors (20, 100, 0.1), (1, 200, 0.1) and (1, 100, 1) are employed to emphasise each factor in turn. Other combinations of weights can be envisaged.

The Service 0 topologies resulting from increasing the k_{rel} weighting factor to 20 are shown in Figure 3.20. Data relating to the overall platform and performance of the tool is presented in Table 3.21. This shows that the tool is unable to find Topologies for Services 2 and 3 if the MTTF requirement is 11000 units. As expected, size and cost increase as MTTF requirements become more stringent. The tool undertakes the largest number of evaluations to find solutions for topologies with MTTF requirements of around 7000 time units. For small MTTF requirements a solution is found relatively quickly. For large MTTFs the tool converges to an unacceptable topology quickly. This gives some indication of the problems that X-Topmeter finds hard.

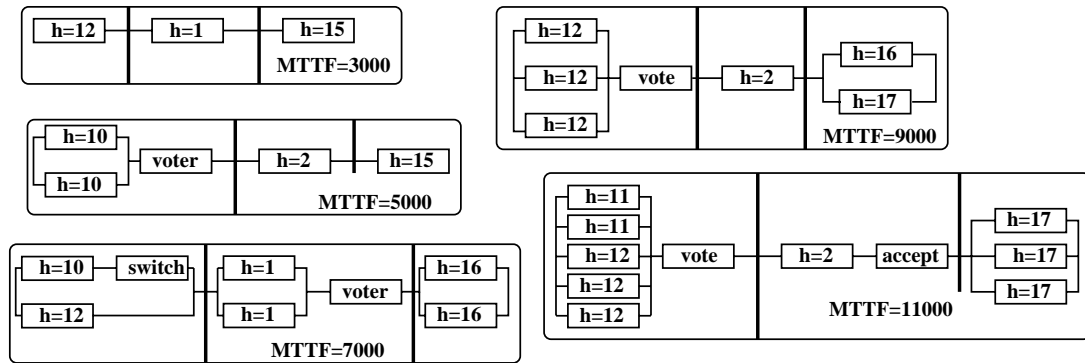


Figure 3.20: Results for $k_{rel} = 20$

MTTF	Fitness	$Cost_{hardware}$	$Cost_{software}$	Size	Steps	Evals	Met?
≥ 3000	24850	15480	9560	15	58	1401	Y
≥ 5000	39220	19800	11390	19	46	1979	Y
≥ 7000	82260	29640	9000	28	61	3838	Y
≥ 9000	134520	35790	13390	36	43	1224	Y
≥ 11000	334490	51550	18750	51	72	3582	N

Table 3.21: Performance Data for $k_{rel}=20$

Compare the results shown in Figure 3.20 with those in Figure 3.18. The effect of increasing the emphasis on reliability is mixed at low MTTF requirements, with the processing task of the 3000 target service having more redundancy and the sensing task on the 5000 target service less. The 9000 target topology exhibits more redundancy, whilst the 11000 target exhibits similar levels of redundancy. A topology to meet the 11000 unit reliability requirement remains elusive, service 2 fails to meet the target (predicted MTTF of 10940).

Weightings are now changed to reflect an emphasis on the size of a topology. Hence weightings of (1, 200, 0.1) are employed. Data relating to the overall platform and performance of the tool is given in Table 3.22. Figure 3.21 shows results for Service 0 for the usual set of five reliability requirements.

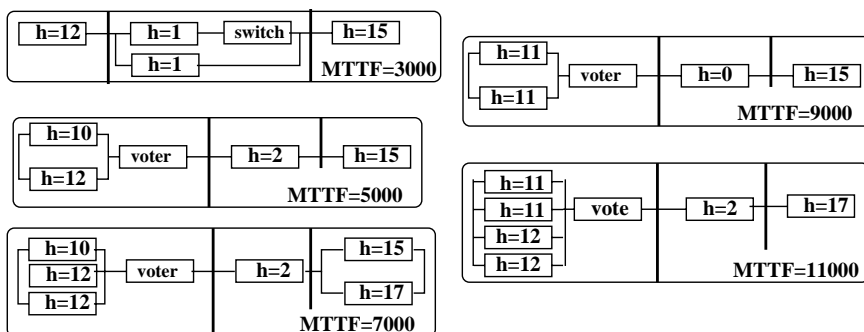


Figure 3.21: Results for $k_{size}=200$

All Services are able to meet their MTTF requirements for all five targets. In particular the tool was able to produce an acceptable topology for the 11000 unit requirement. This indicates that results for the illustrative example are sensitive to the value of the combined cost and size weights used. The previous weighting led the tool towards a non-optimal part of the search space. A smaller number of units are used to fulfil the 11000 MTTF requirement than in the baseline model for the processing and action tasks.

MTTF	Fitness	$Cost_{hardware}$	$Cost_{software}$	Size	Steps	Evals	Met?
≥ 3000	47540	13860	11560	15	58	2248	Y
≥ 5000	61030	17710	14560	17	48	1555	Y
≥ 7000	92350	22980	18560	21	46	3080	Y
≥ 9000	139780	29210	16560	26	46	1493	Y
≥ 11000	185330	347333	18560	32	67	2914	Y

Table 3.22: Performance Data for $k_{size} = 200$

Overall the fitness value is smaller than the baseline topology, indicating that the fitness function is consistent in that smaller values are generated for feasible than for infeasible solutions. These results also indicate that increasing mutation to maintain diversity in the population, and hence increase the number of generations in an experiment, may be effective.

Finally, consider a situation where cost is the most vital characteristic of the topology. The weightings for this option could be (1, 100, 1). Results for the five sets of experiments using these weightings are given in Table 3.23 and Figure 3.22.

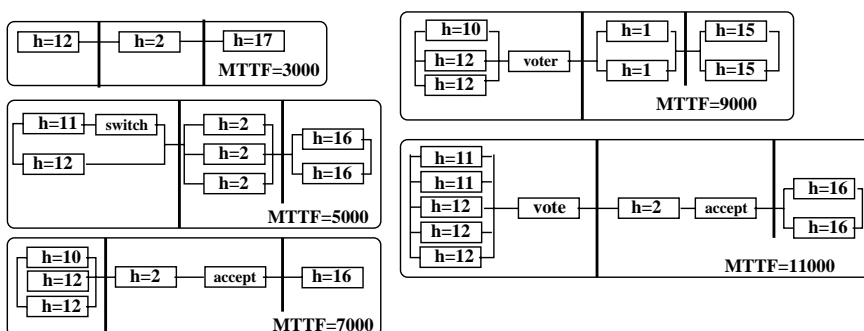


Figure 3.22: Results for $k_{cost}=1$

MTTF	Fitness	$Cost_{hardware}$	$Cost_{software}$	Size	Steps	Evals	Met?
≥ 3000	43060	13900	9560	14	56	2235	Y
≥ 5000	62170	19200	10560	19	50	2785	Y
≥ 7000	92530	25070	14560	23	49	1163	Y
≥ 9000	141120	32260	15810	30	50	2671	N
≥ 11000	190100	39220	16380	36	60	1655	N

Table 3.23: Performance Data for $k_{cost}=1$

The tool is unable to find a feasible solution for high MTTF requirements when the

penalty for the cost of the topology is high. The tool trades reliability against cost. The size of the overall platform has been reduced by 0, 1, 2, 4 and 8 units respectively for the five MTTF targets. This has come at a cost of a reliability 0.6% below the 9000 requirement for Service 2 and a reliability of 0.6% below the 11000 requirement for Service 0.

The designer is faced with a choice. Is the reduced size of the platform sufficient to warrant arguing for a 0.6% fall in reliability? Are aspects of the design open to refinement to increase reliability, while maintaining the reduction in size? Use of this approach cannot answer these questions. However, it does inform the designer both of the gains to be made and allows analysis of the designer's decisions at an early stage in the design process.

Restricted Search Space

A feature of the sequence of baseline topologies shown in Figures 3.13 to 3.17 is that different topologies may be employed to implement each task within the Service. It may be that some combinations of task topologies are not acceptable to the designer [49]. For instance, different sensing and processing task topologies may not be acceptable if implemented in the same Service (Figure 3.23). Second, the designer may wish to stop topologies that have been rejected on previous visits to the topology problem being employed. Third, the designer may insist on redundancy, see Figure 3.23. Finally, a re-use strategy from previous systems of the same type may be in force.

Consider implementing a *redundancy must be employed* restriction on the baseline model. In this scenario the standard weightings of (1, 100, 0.1) are used and non-redundant moves are made inadmissible. Results for the overall topology are presented in Table 3.24 and the topologies for Service 0 are shown in Figure 3.23.

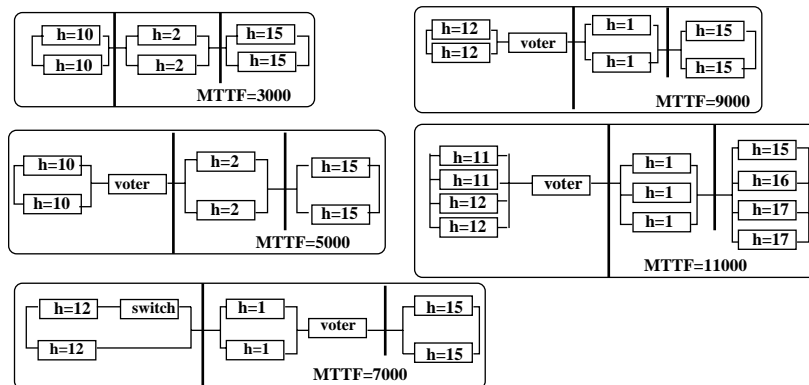


Figure 3.23: Results for Redundancy Required Alternative

MTTF	Fitness	<i>Cost_{hardware}</i>	<i>Cost_{software}</i>	Size	Steps	Evalns	Met?
≥3000	75980	22270	8560	27	46	1386	Y
≥5000	71040	23830	10560	26	52	2308	Y
≥7000	76770	26130	12560	27	50	1325	Y
≥9000	83070	28110	18560	28	68	2403	Y
≥11000	190100	39220	16380	36	60	1655	Y

Table 3.24: Performance Data, Redundancy Required

Results show that a topology that meets the MTTF requirements has been found for all five topologies if redundancy is mandated. The size and cost of topologies for small MTTF requirements is somewhat larger than the baseline model. In fact 13 extra units are employed for the 3000 target problem. This is to be expected and may indicate that in some circumstances mandating redundancy produces topologies that are larger and more costly than necessary. This is particularly likely as hardware reliability improves. Projects that previously required redundancy may eventually be acceptable with a single unit.

For larger MTTF requirements, mandating redundancy has led to feasible solutions that were not found in the baseline model. Removing the possibility of low cost, and low reliability, options has led the tool to focus on new areas of the design space. The topologies produced for Service 0 appear somewhat more plausible than the baseline model. In particular the topology for the 11000 target now seems much more plausible as an implementation.

A second demonstration of scenarios involving restrictions is to link the topology of the sensing task to that of the action task. The configuration of resources available to the sensing task is forced to be the same as the configuration of resources available to the action task. However, the sensing task is allowed to employ a decision mechanism, the action task is not. Data on the resulting platform is given in Table 3.25 and the topology for Service 0 in Figure 3.24.

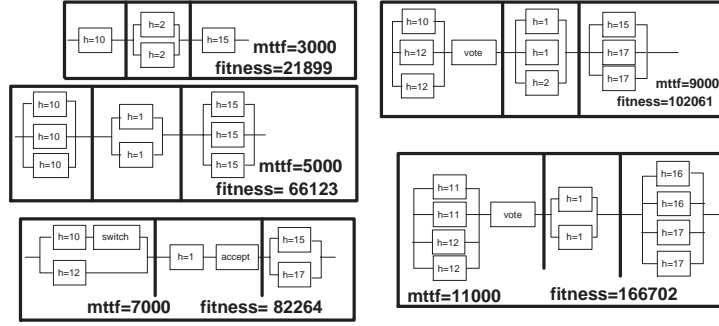


Figure 3.24: Results for Sensing Equals Action

MTTF	Fitness	$Cost_{hardware}$	$Cost_{software}$	Size	Steps	Evals	Met?
≥ 3000	21900	16620	6380	14	35	2627	Y
≥ 5000	66120	25670	10560	25	39	1187	Y
≥ 7000	66370	27100	11560	25	35	976	Y
≥ 9000	102060	32310	17570	31	36	2917	N
≥ 11000	166700	45350	14380	40	37	2117	N

Table 3.25: Performance Data, Sensor Topology Equals Actuator Topology

This restriction leads to *balanced* topologies at the expense of increased cost for small MTTF requirements and the possibility of failing to meet MTTF requirements for larger topologies. Services 0 and 3 failed to meet their MTTF requirement for the 9000 and 11000 MTTF target experiments. It should be noted however that the approach used to implement this restriction is very simplistic. Since two thirds of the tasks are now linked the likelihood of premature convergence of the GA employed is greatly increased. Methods to overcome this problem are available, including increasing the mutation rate and TS techniques.

Effect of changes in Predicted Coverage Rates

The effects of using a decision mechanism in a topology are three fold; it increases the probability of detecting and hence recovering from a hardware failure, it increases the cost of implementation and it may reduce the reliability of the system. There is a trade-off between an increase in reliability due to detection and a reduction due to the extra resources employed to implement the test.

To reflect the effect of a decision mechanism on overall reliability a *coverage* factor is used in the reliability models employed to predict task, network, and service reliability

in X-Topmeter. A coverage factor takes on a value between 0 and 1 and indicates the proportion of failures that can be identified and masked by the topology using the decision mechanism. Since two different decision mechanisms can be employed three coverage factors are required:

1. No decision mechanism employed
2. Acceptance test employed
3. Voter employed

Topologies that employ no decision mechanism, such as a parallel topology, are given a low coverage value. Topologies that employ an acceptance test can detect some value failures and are therefore given a higher coverage factor. Topologies that employ voting mechanisms can identify and mask a variety of failures (e.g omission, early and value failures) and are therefore given the highest coverage factor. The exact value of the coverage factors are selected by the designer from previous experience or analysis of the failure properties of the system being designed.

In Table 3.26 the coverage factors employed for each resource in the baseline search are presented. Thus, for instance, a tuple of (.610, .721, .963) indicates that 60% of failures can be detected and masked by the resource if no decision mechanism is employed. It also indicates that 72% can be masked if an acceptance test is employed and 96% if a voting mechanism is employed.

Model	Coverage Values for Hardware Resources
Baseline	(.610,.721,.963)(.651,.717,.984)(.670,.728,.973)(.642,.707,.925)(.667,.767,.969) (.574,.659,.845)(.581,.744,.814)(.558,.642,.895)(.529,.602,.835)(.522,.636,.845) (.5,.75,.95) (.5,.75,.95) (.5,.75,.95)(.5,.75,.95) (.5,.75,.95)(.45,.7,.9) (.45,.7,.9) (.45,.7,.9) (.45,.7,.9)(.45,.7,.9)
New	(.51,.721,.963)(.551,.717,.984)(.57,.728,.073)(.542,.707,.925)(.567,.767,.969) (.574,.659,.845)(.481,.744,.814)(.458,.642,.895)(.429,.602,.835)(.422,.636,.845) (.5,.75,.95) (.5,.75,.95) (.5,.75,.95) (.5,.75,.95) (.5,.75,.95) (.45,.7,.9) (.45,.7,.9) (.45,.7,.9) (.45,.7,.9) (.45,.7,.9)

Table 3.26: Coverage Factor Tuples

Producing accurate coverage data for each resource that may be employed in the system is likely to be difficult. For instance, the resource may not have been used in a comparable control environment before. Therefore, it is necessary to determine how sensitive the results of a search for an architectural topology are on the coverage values chosen. Hence, in Table 3.26 a second set of coverage factor tuples is proposed.

Topologies resulting from changing the coverage factors are shown for service 0 in Figure 3.25 and results for the platform are shown in Table 3.27. The results are very similar to that of the baseline model, showing that the topology is relatively insensitive to changes in coverage rates. It may well be that the relative size of each factor is more important than the absolute size.

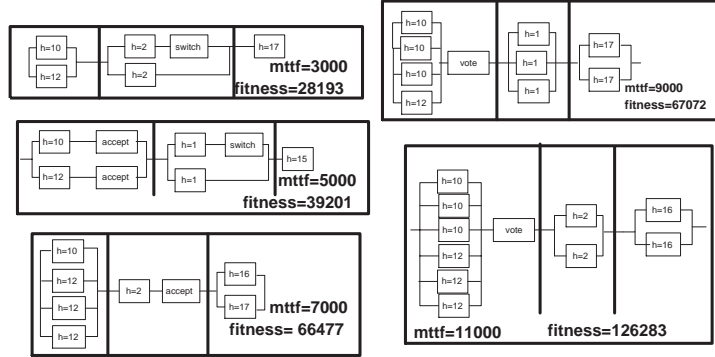


Figure 3.25: Results for Coverage Factors

MTTF	Fitness	$Cost_{hardware}$	$Cost_{software}$	Size	Steps	Evals	Met?
≥ 3000	28190	16380	9560	16	42	1255	Y
≥ 5000	39200	20480	10560	19	52	3211	Y
≥ 7000	66480	28220	11560	25	52	1582	Y
≥ 9000	67070	27330	18390	25	67	3788	Y
≥ 11000	126280	38280	16560	34	75	4007	N

Table 3.27: Performance Data, Coverage Factors

The 11000 target MTTF experiments failed to produce a topology that met the target for all five services. Service 4 was predicted to have an MTTF of 10930.

3.5 Discussion

The architectural topology problem considered in this chapter has hitherto been addressed in an *ad hoc* manner leading to a system topology being set very early in the design process. A fixed topology makes it more difficult for the designer to produce a system design that meets the reliability requirements of the system. Automated tool support would allow the designer to play out scenarios with a set of possible topologies so that the architectural topology of a system emerges during the design process. The topology becomes a degree of design freedom.

In this chapter a first attempt at an approach that supports the production of an architectural topology has been introduced. The approach and supporting tool are flexible enough to allow the designer to investigate different design scenarios. The tool produces a set of task and network topologies for each Service in the control system under consideration. It employs reliability models, based on block diagram and Markov models of proposed topologies. Cost and size characteristics of a proposed topology are also taken into account during the selection process.

There are many scenarios that could be investigated using the approach. It has been shown that the designer is able to vary the relative importance given to the three main characteristics of a topology. The weights used represent the corporate knowledge of the designers of previous applications as to the most appropriate trade-offs between cost, size and reliability. It has also been shown that restrictions on the set of admissible architectural topologies may be introduced.

A quantitative approach to resolving the architectural topology problem appears on the evidence presented in this chapter to be viable. However a number of issues remain. First, the architectural forms considered have been confined to those covered by the GUARDS architecture. The exposition in Chapters 2 and 3 appears to be prescriptive in parts. However, there is no reason in principle why the approach, and tool, cannot be adapted to consider other architectural forms, such as shared memory.

The changes required to investigate a shared memory system could require considerable effort and in the worst case the work presented in Section 3.2 would need to be reconsidered in its entirety. This is because the reliability models and search space investigated would be different for a shared memory system. For instance, designers may have to consider copy tasks for non fully connected shared memory systems; this would have a considerable impact on the models and search space employed. On the plus side, the generic architecture employed is a system level decision and is therefore likely to remain constant throughout the design process. In this case the work implied by Section 3.2 should only need to be undertaken, at worst, once per project. The work required to implement X-Topmeter for a different bus based system is likely to be less profound, see Section 5.5.6.

Second, there is the independence issue. How independent are the services provided by a system? More importantly can a reliability measure be produced for these Services in isolation. Independence is unrealistic. However, at early stages in the design process the interactions between services may not be clear. The ability to cope with inter-service dependencies is therefore based on the quality of the system and reliability models employed. In Chapter 5 a case study is presented in which a single sensing task provides data for two services to show one way in which this issue could be addressed. The approach is designed to support the selection process, it is therefore realistic to make assumptions about independence in the early stages of the process

and upgrade the models employed as appropriate.

A third, and related, issue is that of the set of reliability models employed to predict the MTTF of a service. At present these represent a limited set. More complex, and system specific variants can be envisaged. This may involve a large amount of modelling work at various points in the design process. However, in many cases analysis of this type needs to be undertaken anyway. For the purposes of this tool the designer should employ the minimum level of modelling complexity possible. The aim is to compare the relative merits of a set of alternative topologies, not provide precise MTTF predictions.

A fourth issue is that of computational load. As a design becomes more complex the reliability models employed impose an increasing computational requirement. It should become apparent when the modelling and computational loads make the continuation of this approach unviable. However, by this point the architectural topology of a system should be stable. Techniques exist to minimise the number of evaluation function calls required to produce an acceptable solution. Restrictions on the search space should also minimise this problem.

A final issue relates to the flexibility of this approach with respect to the desired dependability characteristics of a system. Libraries allow different characteristics to be evaluated. There is no reason in principle why these libraries cannot, for instance, be adapted to support the production of availability measures of a topology. In this way systems with different non functional requirements can be investigated. Furthermore, the strings used to represent a solution in X-Topmeter can be changed to accommodate any desired number of different configurations of resources, including different decision mechanisms. The author contends that the approach put forward in this chapter is flexible and powerful enough to allow such extensions. A tool like X-Topmeter should prove to be a valuable aid to the designer of an architectural topology. It will be even more valuable if research into reliability model generators is successful.

In Chapter 4 the allocation problem takes the resolution of the architectural topology problem discussed in this chapter and focuses on the processing tasks. The aim is to allocate the software units to hardware units. Results from the allocation problem may require the architectural topology problem to be revisited, especially if the cost and platform size are deemed to be too high. Interactions between these two problems are discussed in Chapter 5.

Chapter 4

The Topology Problem at the Configuration Level

Resolving the allocation problem facilitates the construction of complex software systems from real-time software units. These software units can be passive data objects or active processes, that have periods and deadlines. Software units are assigned to, and eventually executed on, the processing resources (processors) of a distributed parallel execution platform. The emphasis of the allocation problem is therefore:

- prediction of the Worst-Case Response Times (WCRT) of Services *given* a complete mapping of software units to processing hardware units.
- assignment of software units to a set of processing hardware units derived from the proposed system architectural topology.

The problem of optimal assignment and precise timing analysis of units of real-time systems is NP-hard [93]. As well as timing goals for an assignment, cost goals and resource usage constraints can be envisaged. Sensors and actuators are not typically considered as part of the allocation problem, although Thompson [160] is looking at the “optimisation of smart and dumb sensors and actuators”. Thompson’s approach employs a GA and is complementary to the work presented in this thesis.

A chosen allocation represents one possible partial solution to the topology problem for a given system. Thus, in the context of the GTP the allocation problem can be informally defined as

“map the set of software based active processes (sub-tasks) proposed for a given architectural topology to a known set of processing hardware units. Furthermore, assign the messages arising from this allocation to a given set of network hardware units.”

The resulting allocation should be *feasible* in that all resource usage constraints and all Service WCRT deadlines are met. If more than one feasible allocation exists the size and cost of the allocation should be minimised. The allocation problem may be formally defined as an ATP or SAT problem.

The allocation problem presented here differs from the standard problem formulation addressed in the literature in two respects. First, the set of active objects (sub-tasks), passive objects (messages) and configured resources (processing and network) to be employed are determined by the resolution of the architectural topology problem. A processing hardware platform is produced with the sub-tasks and messages assigned to it. The predicted hardware platform can be thought of as a physical implementation of the selected architectural topology.

A second feature introduced by considering the allocation problem as part of the GTP is an increase in variability of the size of the hardware platform. Hitherto, attention has focused on *cramming* software units onto a fixed hardware platform. In this case the emphasis of the allocation problem is schedulability. A feasible allocation exists, or it doesn't. By relaxing the fixed size constraint the designer is given extra design freedom. The allocation problem is transformed into producing the minimum size of platform required to implement the topology, such that timing requirements are met.

Multiple routes through a control Service may be introduced to provide fault tolerance and reliability (Chapter 2). As a result the allocation problem needs to focus on *transactions*. A transaction takes the form of a sequence of sub-tasks that describe a path through a Service. Some sub-tasks may be constrained to run on particular processors (for example input and output sub-tasks), the rest are able to execute on any copy of the processing resource they have been assigned to in the selected architectural topology for the system. Other restrictions on allocation are possible. For instance, if transaction replication is used for fault recognition, or increased availability, the replicas must be physically separated.

Architectural topologies may be selected from an early stage of the architectural level of the design process and revised as design progresses. Thus, the allocation problem may be revisited on a number of occasions. Resolution of the allocation problem provides feedback to the architectural topology problem. For instance, the appropriate percentage of messages that are inter-processor can be set. If a feasible, and acceptable, hardware platform cannot be found this obviously has an impact on the validity of a proposed architectural topology.

In Section 4.1 previous research into the allocation problem is reviewed and in Section 4.1.1 the elements required to resolve the allocation problem are presented. Recent results have shown that the use of Simulated Annealing is appropriate [117]. Thus, in Section 4.3 a tool that employs SA is presented. Finally, in Section 4.4 illustrative

examples show the efficacy of the tool.

4.1 The Allocation Problem

The literature on the allocation problem can be split into three categories: papers that address static allocation [117, 163, 3], papers that address static allocation and scheduling problems simultaneously [32, 39], and papers that address dynamic allocation of sub-tasks. The last of these categories relates mainly to multistage interconnection networks [113] and is not appropriate to the models used in this thesis. Scheduling is assumed, in this thesis, to be via priority based scheduling, as presented in Section 2.6. Hence, schedulability analysis can be employed as part of an allocation process.

The majority of allocation papers focus on load balancing in order to minimise communication overheads [38]. The hard real-time allocation papers however focus on the NP-hard nature of the allocation problem for these systems. The ‘strange shape’ of the search space for this problem has been noted [117]. That is, good allocations are often found close to poor, or inadmissible, allocations making it difficult for search algorithms to converge to an optimal solution. Tindell et al [163] address the standard allocation problem for a TDMA bus based multiprocessor system. Nicholson et al [117] extend this work to consider non fully connected point-to-point shared memory networks.

A platform is one physical implementation of an architecture and consists of configured hardware units. The platform required for a particular allocation problem is a function of a number of factors:

- maximum number of units of each hardware resource that may be employed
- the number of sub-tasks to be allocated,
- the number of sub-tasks (messages) that cannot be placed on the same processor (bus) because of the form of the selected architectural topology,
- the WCET of sub-tasks and messages, and
- the WCRT requirements for each processing task

An architectural topology sets the type of platform to be employed and is a given in the allocation process. For instance, a shared communication (bus-based) architecture is investigated in this thesis. A number of generic architectural topologies have been produced for platforms of this architecture, including the GUARDS architectural topologies introduced in Section 3.1.

Point-to-point architectures have a number of advantages over other forms of distribution: there is no physically shared communication media to schedule and they re-scale (off-line) with the minimum of disturbance. One example of a point-to-point network is DIA (Data Interaction Architecture) [156]. This architecture has been designed specifically to support real-time applications. In DIA, nodes are linked via dual port memories; the network is not fully connected.

In a point-to-point architecture some transactions need to be extended so that they can be mapped onto the available hardware platform. For example, if sub-task 0 and sub-task 1 need to communicate but are bound to nodes that do not have a direct link, routing of messages is required. This routing takes the form of extra sub-tasks on intermediary nodes.

In the rest of this Section the elements of the variable platform size allocation problem for bus based systems are introduced.

4.1.1 Representing an Allocation

A Directed Acyclic Graph (DAG) of precedence constrained sub-tasks is produced by the designer of a system based on the functional design and the Architectural Topology selected for each processing task in a computer based control system, see Figure 4.1. The DAG indicates a set of possible paths through a Service and therefore determines the set of transactions in a system. If diverse copies of tasks are employed then each copy may employ a different number of sub-tasks. Thus, the number of sub-tasks to be allocated is partly a function of the level of replication, and diversity, employed in the architectural topology selected to provide fault tolerance and reliability.

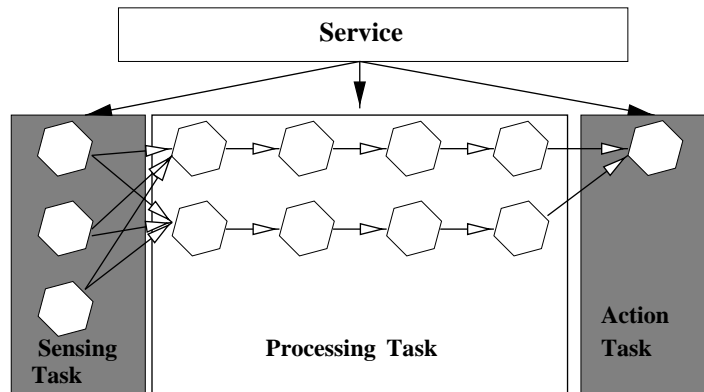


Figure 4.1: An Allocation Problem

Assume that each sensor and actuator indicated by the architectural topology tool (X-Topmeter) represents a physical hardware unit. Thus, allocation of sensors and actuators is not required. Furthermore, assume that the deadlines for each transaction

can be adjusted to take into account the WCRT of sensors and actuators.

Emphasis is placed on transactions as not all processing elements form part of a processing task. For instance, software based sensor fusion sub-tasks, that are logically part of a sensing task, need to be allocated to processing resources. A sensor fusion sub-task receives data from a number of sensors, manipulates this data, and acts as a *congruent* data source for the transactions in the service. A congruent data source is one in which a single message is broadcast to multiple locations. It is assumed that if the WCRT of the transactions in the system can be met the WCRT of the corresponding Services can be guaranteed.

In schedulability analysis the path (transaction) with the longest WCRT determines the WCRT of the Service ¹. In Figure 4.2 a processing task consisting of two transactions is presented. Sub-tasks 0 to 4, and 9 represent transaction zero and sub tasks 0, and 5 to 9 represent transaction one. A third transaction would be formed if the two transactions were linked by a communication between pst_2 and pst_7 , for example. In this case the third transaction would consist of sub-tasks 0,1,2,7,8,9.

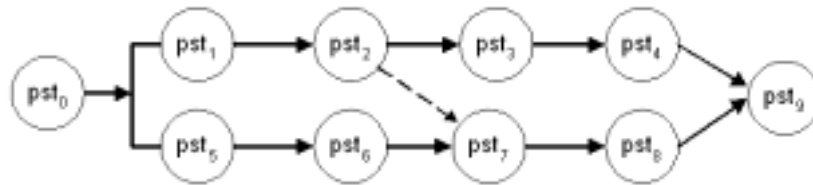


Figure 4.2: Processing Sub-tasks

Sub-task (pst) 0 is a sensor fusion sub-task that acts as a congruent data source. Sub-task 9 is an output voter sub-task that receives data from a number of transactions and produces a congruent data source. This data source is used by actuators to control the application. The WCRT of a task is the maximum WCRT of the transactions that form the task. In this case $\max(trans_0, trans_1)$, or $\max(trans_0, trans_1, trans_2)$ if the extra message is employed.

Two attributes of each sub-task are used to determine a set of possible allocations, namely: *Priority* and *Processor Residency*. The *Bus Residency* of the messages emanating from each sub-task also needs to be determined. Messages are passive objects. They inherit the priority of the sending/receiving sub-tasks. Each allocation can thus be represented as a set of three ‘genes’, see Figure 4.3.

¹For the sake of brevity in the remainder of this chapter the phrase *processing task* will refer to not only a processing task but also any sensor fusion sub-tasks. The deadline of a processing task thus represents the deadline for the processing elements of a Service.

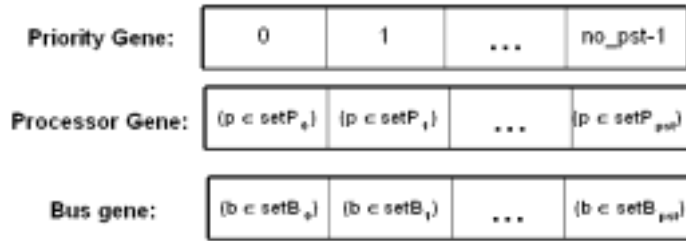


Figure 4.3: Allocation Problem Representation

The setP term in Figure 4.3 indicates the set of processing units that a sub-task may be placed on. This set is determined by the appropriate resource assignment in the selected architectural topology. Similarly, setB indicates the set of buses that a message may be placed on. The number of sub-tasks is *no_pst*, number of processors is *no_proc*, number of messages is *no_mes* and the number of buses is *no_buses*.

Possible numerical alphabets for the allocation problem include integer, permutation and binary. The priority gene requires a permutation alphabet. A permutation problem has a set of elements each of which takes on a value from a set of admissible values. No two elements may have the same value. For example, one permutation of the sequence 0123 is 0213. The processor and bus residency genes employ an integer alphabet in which more than one element may take the same integer value. For example, one value for a four element integer string is 0121. The user defines the range of values for each element in the string. As a result more than one sub-task may be resident on the same processor.

The values to be set to resolve an allocation problem formulated as an ATP, are:

- Priority for each sub-task: a PERMUTATION with range 0 to (no_pst-1),
- Processor Residency for each sub-task: an INTEGER with range 0 to (no_proc-1)
- Bus Residency for each message: an INTEGER with range 0 to (no_buses-1).

4.1.2 Evaluating an Allocation

Once an allocation has been produced, by setting the values of the three gene structures for each processing sub-task and message in the system, an evaluation of the quality of the allocation can be made. The evaluation function contains a number of capacity and timing constraints that must be met for a proposed allocation to be *feasible*. The allocation evaluation (penalty) function is:

$$P(x) = \alpha F(x) + \beta G(x)$$

The function F(x) determines the value of an allocation with respect to the following *goals*:

1. minimise cost of hardware and software employed in the platform (cost)
2. minimise number of physical hardware units employed (numhware)
3. minimise total WCRT of tasks in the system (WCRT)

Parentheses indicate the name of the goal. The weighting factor α is a vector of three elements. For instance, each unit of cost is multiplied by a factor of α_1 . Thus, $F(x)$ is a linear function.

The function $G(x)$ determines the evaluation function value resulting from any constraint violations. It takes the form of the sum of the square of total constraint violations. That is the number of units a constraint is missed by is counted and this value squared. The resulting number is subjected to a weighting factor. The vector β currently consists of weighting factors for the following 9 constraints ²:

1. Given sub-tasks (messages) must not be allocated to the same hardware unit (parallel)
2. Memory capacity per processor must not be exceeded (memp)
3. Maximum band-width of buses must not be exceeded (memb)
4. Maximum number of messages received per processor must not be exceeded (numesp)
5. Maximum number of messages per bus must not be exceeded (numes)
6. Maximum number of sub-tasks per processor must not be exceeded (numpst)
7. Number of sub-tasks that fail to meet WCRT requirements must be zero (wcrtpst)
8. Number of tasks that fail to meet WCRT requirements must be zero (wcrtt)
9. Number of bytes of inter-processor communication between sub-tasks within a task must not exceed the budgeted task total (transmes)

In order to calculate $P(x)$ for a proposed allocation schedulability analysis of the sub-tasks, messages, transactions and ultimately Services in the system is undertaken. The schedulability techniques introduced in Section 2.6 are pessimistic, and therefore, safe.

²Parentheses indicate the name of the constraint.

4.1.3 Producing New Allocations

Consider, a set of possible changes to the three primary attributes of an allocation; priority, processor residency and bus residency. First, the priority gene. To maintain exactly one copy of each priority value in an allocation only swap moves are allowed. A swap move transposes the values of two elements of a gene structure. Restrictions may be placed on the set of elements that may be swapped.

Second, both change and swap moves can be performed on a processor residency gene. A change move is one where the value of a single element in the gene structure is changed. For instance, assign a randomly chosen sub-task to a processor randomly chosen from a set of admissible processors (setP) for that sub-task.

Finally, for the bus residency gene, change and swap moves can be employed.

One possible set of admissible moves for the allocation problem is therefore:

1. Swap priorities of two sub-tasks (50%)
2. Move a single sub-task to a different processor in setP (15%)
3. Swap the processor residency value of two sub-tasks. Ensure that the two sub-tasks use the same hardware resource. (10%)
4. Move a message to a different bus (15%)
5. Swap the bus residency of two messages. Ensure the two messages use the same hardware resource. (10%)

Assignments 2 to 5 determine the size of the hardware platform. So, for example, if two communicating sub-tasks are allocated to the same processor a single processor is required. However, if the sub-tasks are allocated to different processors, two processors and a bus are required. Assignment 1 partly determines the timing characteristics of a set of sub-tasks allocated to the same processor. It therefore indirectly determines the size of the platform. For example, extra processors may be employed to ensure the WCRTs of particular sub-tasks.

In Figure 4.4 the effect of the five admissible moves on the gene-string is shown graphically. The double headed arrows indicate swap moves and the single headed arrows change moves. The percentages indicate the default probability that each move is used in an iteration of a search algorithm. In this example an average of 50% of moves involve sub-tasks swapping priority value.

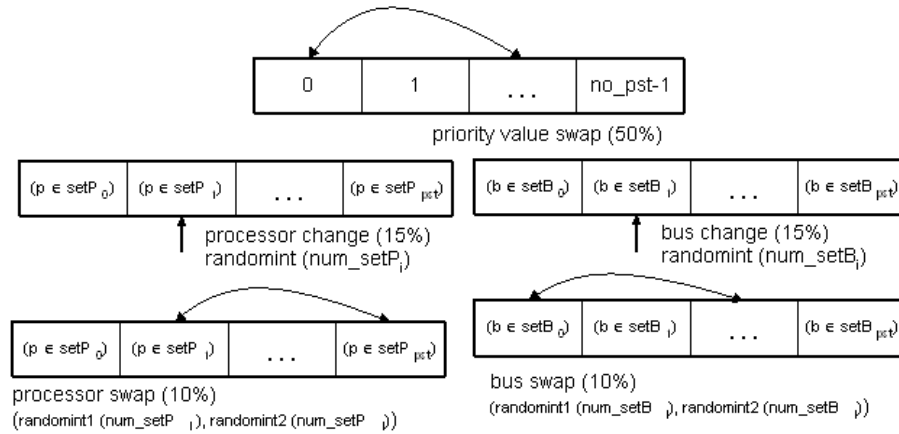


Figure 4.4: Moves for the Allocation Problem

4.1.4 Evolution of an Allocation

Allocation has typically been undertaken late in the design process once a set of sub-tasks and a hardware platform have been proposed, and in many cases implemented. The sheer number of alternative allocations makes the resolution of this problem difficult. Furthermore, a majority of possible allocations may be infeasible in that the transactions cannot be guaranteed to meet their required WCRTs. A failure to produce an acceptable allocation can be very costly.

Resolution of the architectural topology problem, for a particular level of design decomposition, allows the allocation problem to be addressed earlier in the design process. It also allows the designer to predict any changes to the optimal allocation implied by a design decision. Since, in the formulation of the allocation problem presented in this chapter the size of the platform is not fixed, a resolution to the allocation problem can almost always be found. However, the size and cost of the platform required to produce a schedulable system may be deemed excessive by the designer of a system.

If an allocation is deemed acceptable, given the currently selected architectural topology, then design can progress further. If the allocation resulting from an allocation search is deemed unacceptable then one of the factors that determined the allocation must be changed.

Suppose that for a given topology 10 processing units were required to meet all timing and resource usage constraints. The designer was aiming to produce a platform with eight processing units. Four design choices are available:

1. relaxation of the reliability requirements leading to a topology with less sub-tasks to be allocated

2. relaxation of the size constraints
3. relaxation of the timing requirements
4. change in the functional design.

The first choice will lead to a reworking of the architectural topology problem. Choices 2 and 3 will probably require only a reworking of the allocation problem. Choice 4 will require reworking of both elements of the topology selection process. The ability to allow an allocation to emerge as part of a resolution to the GTP is one of the desirable features that prompted the particular approach put forward in this thesis.

4.1.5 Problem Formulation - ATP

The allocation problem is a known NP-hard [93] combinatorial optimisation problem. An allocation x requires three assignments to be made. First, each sub-task is assigned a unique priority number. Thus, $x_{\psi z} \rightarrow (0, 1)$, where $x_{\psi z} = 1$ implies priority ψ is assigned to sub-task z . Second, each sub-task is assigned to a processor. Thus, $x_{zp} \rightarrow (0, 1)$, where $x_{zp} = 1$ implies sub-task z is assigned to processor p . Finally, each message is assigned to a bus. Thus, $x_{mb} \rightarrow (0, 1)$, where $x_{mb} = 1$ implies message m is assigned to bus b .

The primary side-constraint is to ensure that the WCRT of each task is met. Further, side-constraints are used to ensure that the 9 constraints presented in Section 4.1.2 are not exceeded by the chosen allocation.

In order to solve, or at least postulate a solution for, the allocation problem a great deal of information is required about the design of the system and its hardware platform. However, given a proposed architectural topology a prediction of an appropriate resolution to the allocation problem can be produced during the design process. The default set of assumptions for the baseline X-Alloc guided search tool are:

1. Only processing tasks, and inter-processor communication networks, need to be allocated to a hardware platform.
2. Processing tasks consist of a set of precedence constrained sub-tasks (Figure 4.2) and can be modelled as periodic with offsets (see Section 2.6).
3. Each sub-task (message) in parallel must be allocated to a different hardware unit. More than two sub-tasks (messages) in parallel can be required.
4. Priority of sub-tasks is set on a global basis, since the number of sub-tasks is fixed by the resolution of the architectural topology problem.

5. The Dependability predicted by the *X-Topmeter* tool is NOT changed by the allocation process. An unrealistic assumption, but one that allows the tool to concentrate on the timing characteristics of an allocation.
6. Number of hardware units that must remain operational for a task to remain operational is assumed to be one of each type employed by a task.
7. Assumptions presented in Section 3.2.1 are still in force.

4.2 Resolving the Allocation Problem

Like the architectural topology problem, a variety of heuristic and search techniques have been employed to aid resolution of the static allocation problem, see Figure 4.5. The relative effectiveness of each technique varies depending on the exact formulation of the problem. A survey of these techniques with respect to the allocation problem and a justification for the use of Simulated Annealing (SA) is presented.

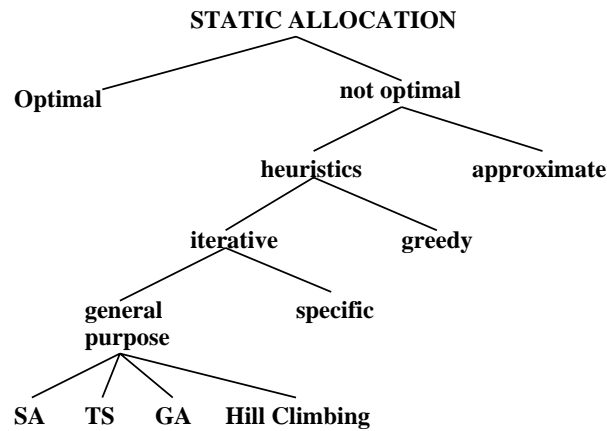


Figure 4.5: Search Techniques for The Allocation Problem

Since the standard allocation problem is NP-hard optimal search techniques, such as branch-and-bound or network flow techniques, cannot be used. Non optimal techniques have therefore been applied. All the techniques discussed below have been subjected to hybridisation in an attempt to overcome perceived weaknesses. Hybridisation with Tabu Search (TS) may prove effective and is within the scope of the approach employed in the prototype allocation tool, X-Alloc. Hybridisation of constructive heuristics and search techniques may also be effective.

Steepest Descent

Steepest descent (Hill Climbing) is the simplest search approach applied to the allocation problem. A single allocation is produced and an exhaustive set of single

change moves applied. The best of these allocations is taken as the start of the next iteration. The search terminates when no better allocation is found. Two major disadvantages of this approach are evident: the excessive run-time required and the inability to overcome walls between local minima. Surprisingly good results sometimes emerge however. Most steepest descent algorithms are employed as part of a hybrid approach [3].

Tabu Search (TS)

TS has been employed by Porto & Ribeiro [128] to allocate non real-time sub-tasks to an heterogeneous multiprocessor network. A tabu list, see Section 2.9, is maintained for each (sub-task, processor) pair ³. A pair is placed on the tabu list for a finite number of iterations whenever moving the given sub-task to the stated processor results in a worse allocation. In this case worse implies a larger evaluation function value has been returned.

This approach can be extended to hard real-time allocation problems. In addition to a tabu list for processor residency, lists for bus residency (message, bus) and priority (sub-task, priority) can be employed. Furthermore, tabu list structures can be extended to make moves that violate resource constraints tabu. For instance, a sub-task move to a processor that has reached its sub-task residency capacity limit.

The run-time of TS is reduced by forcing elements to stay longer on the tabu list. TS sometimes runs into a ‘dead-end’ without having found a feasible solution, when started for an initial non-feasible solution. A TS employed by a maintainer for a new variant of a system is likely to start with a reasonable solution and therefore is less likely to be subject to this problem.

Elements of TS may be readily hybridised with other search and heuristic techniques to improve the overall search strategy. For instance, a TS/SA hybrid is compatible with the allocation strategy employed in the X-Alloc tool. This extension would be along the lines of the hybridisation presented in Chapter 3.

Genetic Algorithms

Variants of GAs have been employed to resolve the allocation problem. For instance, Thompson [160] employs a GA as part of a sensor / actuator allocation process. Chromosomes may be formed from the list of sub-tasks on each processor, for a particular allocation [7]. Alternatively each allocation may be represented by one chromosome onto which all relevant information is encoded [154]. Greenwood et al [65] employ a variant of GAs, known as Evolutionary Strategies (ES). ES works with intensification

³A pair represents a proposed allocation of a particular sub-task to a particular processor.

operators (e.g. crossover) and a survival of the fittest strategy.

GAs mainly suffer from sensitive parameters and the inability to find feasible allocations for large examples. In addition, the basic assumption that strong parents are likely to generate strong children is not true for the allocation of real-time sub-tasks [3]. GAs deliver better results when a very high mutation rate is employed. In large applications there is also the problem of storing large numbers of chromosomes. Furthermore, it has already been established that GAs do not perform well on fine-tuned searches in highly combinatorial spaces [63, 68].

Constructive Heuristics

The purpose of constructive heuristics is to find any feasible allocation by some simple strategy avoiding spending too much time in optimisation. One such technique, List processing [136] is based on one or more global priority lists. Ramamritham's [137] approach to resolving the allocation problem consists of two parts:

1. The list processing element which decides whether clusters of communicating sub-tasks should be assigned to the same processor.
2. A search heuristic employed to assign clusters of sub-tasks to particular processors. The search also determines a feasible schedule.

The main advantage of constructive heuristics is high computation speeds. They may, like search techniques, be highly sensitive to the implementation of elements of the heuristic. In list processing the difficult part is to implement an appropriate priority function.

4.2.1 Simulated Annealing

Simulated Annealing (SA) has been employed previously to address the allocation problem [163, 117, 32, 39]. Typically, deadline misses are included as a penalty in the evaluation function of each potential allocation. Most authors attempt to handle the allocation and scheduling problems simultaneously.

SA is a variant of the generic search algorithm, see Section 2.7.1, which employs:

1. Selection. This is simply the allocation surviving at the end of each iteration of the algorithm. This solution may not be the best solution found so far.
2. Creation. The current allocation is manipulated to generate a new allocation. Two types of move are employed; *change* and *swap*.

3. Merging. In SA only two allocations need to be considered each iteration (existing and proposed). The proposed allocation replaces the existing allocation if its' evaluation function value is less than that of the existing allocation. If the proposed move has a value greater than or equal to the existing allocation it may still replace the existing allocation with a probability based on the SA temperature parameter. The probability of a *worse* allocation being accepted falls as the search progresses.

The standard SA algorithm was presented in Section 2.10. A cooling schedule defines the merging approach used by an annealing algorithm and hence determines the algorithms ability to converge to a solution. There are two parts to a schedule; the cooling rate and the cooling mechanism. Options for the cooling rate include:

1. Cool at a geometric rate. Typically, $\alpha = 0.95$
2. Cool in arithmetic steps. Typically 0.1

In both cases the value of the temperature parameter is reduced from an initial value. The initial value can either be set or can be determined by the annealing algorithm as the temperature at which a given proportion of moves are accepted. Once the schedule and associated parameter have been chosen, the user is presented with a further choice on whether to allow increases in temperature, or not. One heating approach is tempering. A tempering mechanism, see Section 2.10, employs a parameter that specifies the threshold percentage, which when reached “reheats” the annealing algorithm.

An allocation SA search is terminated once any one of three stopping criteria is met:

1. Maximum number of permissible evaluations reached
2. Maximum permissible time has elapsed
3. Number of evaluations since the last new *best* solution exceeds a maximum value

One perceived disadvantage of SA is that it is sensitive to its parameters and it is difficult to find parameters that will fit for all application examples. Experimental evidence has shown that allowing the temperature parameter to rise (tempering) as well as fall, can produce better results than a strict cooling process.

A second perceived disadvantage of SA for the allocation problem is the inability to converge quickly to a feasible solution. This is mainly due to the nature of the search space and is known as the deception problem. A search is deceived into moving to a part of the search space that does not contain good solutions. The cooling process then

stops the algorithm escaping from this area. The search space for an allocation typically becomes fractured as a result of resource usage constraints. Annealing techniques find it difficult to traverse such search spaces [117]. A good solution may always be ‘hiding’ behind an inadmissible point in the search space.

Previous experience has led the author to believe in the effectiveness of SA. Furthermore, the experience gained on previous work can be transferred. It is generally true that the standard *vanilla* flavour of a search technique can be improved on by domain knowledge and familiarity with the technique. This has proved to be one of the problems with comparative evaluation of different techniques and has become known as the “beauty contest” problem, see Section 2.7.2.

The sensitivity to parameter value problem may be perceived as an advantage, as well as, a disadvantage. It is envisaged that the allocation tool will be applied a number of times during the topology selection process. Before each run the parameters employed can be adjusted. So, for a particular application the evaluation function values that produce the best results will emerge during the selection process. The weights used will partly reflect the relative importance given to each factor by the designers and partly reflect the nature of the search space. This approach allows the domain knowledge of designers to be factored into the process.

SA parameters are honed to work most effectively for a particular application by this process. The weighting and parameter settings can be stored and used either by the maintainers of the system when an upgrade is being contemplated, or the designers of similar systems in the future. New systems, or variants of the existing system, may use these settings as a starting point.

The extension to allow extra variability in the size of the platform employed implies that a feasible allocation can nearly always be found. The size and cost of this feasible platform may however be unacceptable to the designers. This extension is also expected to help reduce the impact of the deception problem, as is hybridisation with TS techniques.

SA was also chosen for practical reasons. The SAMson [103] environment on which the allocation tool, X-Alloc, is based is compatible with the GAMeter environment, on which X-Topmeter is based. Results from X-Topmeter can be employed in X-Alloc, after appropriate interpretation and manipulation. The input data required for X-Alloc is more extensive than that for X-Topmeter due to the level of decomposition of the design at which the allocation problem may be resolved.

Like GAMeter SAMson provides a set of ingredients from which a recipe to solve a particular allocation problem can be chosen. It also provides the ability to ‘cook’ the chosen recipe to select the best allocation, in the form of an SA. Producing a particular recipe from the given allocation problem ingredients is non trivial. The ingredients and

‘garnish’ that need to be coded for each problem include:

- a set of processing sub-tasks for which an allocation is to be produced
- connections between sub-tasks in the allocation, in the form of messages
- set of admissible allocations
- how to produce a new allocation from an existing allocation
- schedulability and resource constraint measures
- evaluation function and weighting factors

The recipes employed for different problems are likely to be similar in form. To extend the analogy: many variants on pasta in a red sauce can be envisaged. The basics however remain the same. All recipes are likely to require considerable coding effort. This effort falls as the design process continues. Re-evaluation of the appropriateness of the current ‘recipe’ is required each iteration.

4.3 X-Alloc

X-Alloc is designed to aid resolution of an ATP formulation of the Allocation problem. The assumptions presented in Section 4.1.5, the elements of an allocation presented above and the SAMSON environment are merged to produce a baseline tool.

An X-Topmeter search produces an architectural topology, which includes a logical platform consisting of a set of configured sensor, actuator, processor and bus resources. These resources form the basis for the implementation of each control Service. The baseline X-Alloc tool investigates the allocation of processing sub-tasks and inter-processor messages. The end-to-end deadline of each transaction is calculated to ensure that overall Service deadlines are met.

Consider a single service system that employs a TMR processing task topology, see Figure 4.6. The topological configuration of this task has been produced by X-Topmeter and consists of three transactions each of which terminates at sub-task 7, a voter mechanism.

To allocate this set of transactions five new data structures are required:

1. Vector of allocation parameters (11)
2. Sub-tasks - each task is constructed from a set of sub-tasks (12)
3. Processors - each sub-task is allocated to a single processor unit (8)

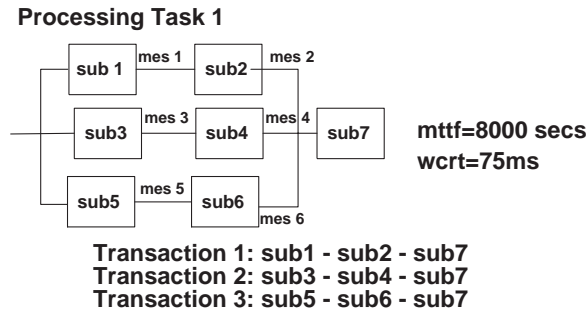


Figure 4.6: Transactions for a TMR Task

4. Messages - each sub-task may send up to three messages to other sub-tasks (16)
5. Buses - each message is allocated to a single bus unit (7).

The number of elements in each structure is shown in parentheses. Details are given in Appendix A. Structures 3 to 5 are particular instances (units) of the resources employed in Chapter 3.

A feature of the transition from an architectural topology to an allocation problem is the modelling of hot and cold spares. Hot spares can be modelled directly, cold spares cannot. However, the WCET of sub-tasks subject to cold sparing can be extended to incorporate the switching time to the spare. Thus, cold spares are accommodated in the following manner. First, each cold spare is treated as a hot spare with extended WCET. Second, a number of processors and buses are designated as cold spare resources. The cold-spare sub-tasks and messages are allocated to these resources, via the setP and setB structures.

The baseline implementation of X-Alloc has the following features:

1. An allocation is represented as a set of tuples containing three integers denoting the priority and processor residency of each sub-task and the bus residency of each message in the system.
2. The evaluation function penalises cost, WCRT and resource constraint violations. It does not explicitly employ a size element. Experimentation has shown that a cost function, which implicitly includes a penalty for the number of units used, is sufficient to find an acceptable allocation. The vectors of weighting factors α and β are determined by the user on a per experiment basis. Default settings are presented in Table 4.1.
3. An initial temperature based on an 80% acceptance rate of proposed moves by the annealing algorithm.

4. Move functions that are implementations of the moves presented in Figure 4.4. The user may alter the proportion of moves of each type employed by an experiment.
5. A geometric cooling schedule that employs a monotonic cooling scheme is used to reduce the probability of accepting a move to a ‘worse solution’, as an experiment progresses. Temperature equilibrium is attained after 1000 proposed moves.
6. Three termination criteria are employed: 7200 seconds, 100000 proposed moves or 500 proposed moves since the last new best solution was found

Parameters	Value
Temperature Ratio	80%
Cooling Schedule	Geometric
Cooling Mechanism	Monotonic ($\alpha = 0.95$)
Temperature Equilibria Mechanism	proposed moves
Equilibria Proposed Moves	1000
Current Neighbourhood Operator	user defined
Maximum time, proposed move, no change limits	7200,100000,500
Collect statistics every n iterations	50
k_{cost}, k_{wcr}	0.1, 0.01
kparallel,kmemp,kmemb,knumes,knumesp,	200, 0.01, 0.01, 10, 10
knumpstp,kwcrtp,ktransmes,kwcrtt	10, 500, 10, 50

Table 4.1: X-Alloc Parameter Set

In the following illustrative examples the default parameter set is employed unless otherwise stated. The illustrative examples are those introduced in Section 3.4. The allocations produced for these examples represent a “snap shot” of the resolution of the GTP at one point in the design process.

The schedulability algorithm employed in X-Alloc is that put-forward by Bate and Burns [10], see Section 2.6. Therefore, each sub-task is modelled as a periodic active object with offsets. Appropriate offsets, along with the WCRT of each task, are calculated by X-Alloc.

Inter-processor messages are included in calculations of the WCRT of a transaction and hence the processing tasks in a system. An inter-processor message is given the priority of the sender sub-task. If two messages of the same priority (i.e from the same source) are resident on the same bus the tie is broken by the priority of the receiving sub-task. The sub-task with the highest receiving sub-task priority is given the highest priority value.

The illustrative examples presented in Chapter 3 have been extended to incorporate a set of sub-tasks for each transaction. Together these sub-tasks should reflect the characteristics of the tasks they are a decomposition of. Thus, the cumulative WCETs of sub-tasks in a transaction should be no greater than the WCET of the parent task. Similarly, the cumulative number of bytes of data sent by the sub-tasks through the bus networks should be no greater than that budgeted for in the resolution to the architectural topology problem for the parent task. X-Alloc ensures this by penalising allocations that exceed the appropriate budgets.

4.4 Illustrative Examples

Four elements of the X-Alloc tool production process required considerable work:

1. Extension of the existing data structures and introduction of new structures
2. Effect of search space restrictions on the ability to find a solution and the effect of relaxing these restrictions
3. Implementation of the schedulability analysis so that a quick and safe evaluation of the schedulability of any proposed allocation could be produced
4. Setting of the SA algorithm parameters

The data structures employed in X-Topmeter were extended so that decisions made in the architectural phase, and subsequent decomposition of processing tasks into precedence constrained sub-tasks, could be investigated. Furthermore, the characteristics of a platform are somewhat different from the architectural topology produced by X-Topmeter. Individual units (primitive resources) rather than generic resource types are the focus of attention. Thus, sub-task, message, bus and processor structures were introduced. These structures reflect the characteristics of their parent task and resource structure.

The example generation program was used to test that the new structures could represent a set of example allocations. Initially, no separate structure for messages was employed. This can be seen from the extensive message information stored in the sub-task structure element *message*, see Appendix A. This however, made the program too complex and message structures were introduced.

X-Alloc was initially produced with a small variability in the size of platform. Moves that broke a number of constraints were made inadmissible. In this circumstance the implemented X-Alloc found it difficult to produce a solution. A wide variety of parameters and cooling schedules were employed. In the end it was decided to greatly

expand the size of the *virtual* platform and then to penalise allocations that employed costly allocations. This approach has proved to be very effective with the tool rapidly converging to a solution for the examples attempted so far. The reason appears to be the effect of such an approach on the structure of the search-space.

The author was able to employ a prototype tool produced by Iain Bate, to predict the schedulability of each proposed allocation. Work was required to interface X-Alloc with this tool and to produce the results in a form that could be effectively evaluated by X-Alloc. Some of the interactions were very subtle and a number of experiments were required to gain assurance in the results being produced.

Throughout the X-Alloc development and prototyping process a variety of parameter values have been employed. Eventually, a set of values that seem to give reasonable results emerged. These values usually need tweaking to produce the best results for any given example, but provide a suitable starting point.

4.4.1 Illustrative Example One

Illustrative example one consists of a single control service implemented as three tasks, 1,0,2, see Figure 4.7. The architectural topology selected by X-Topmeter consists of a sensing task implemented as a TMR topology, a processing task implemented as a hot-spare replicated topology, and an action task implemented as a triple redundant topology. Both lanes in task 0 employ copies of the same software unit and the same processing hardware resource (processor type 2). Both lanes also employ data provided by a sensor fusion sub-task, sub-task 0. This sub-task transmits a single message across a single bus to ensure a congruent data source.

A single network is used to transfer data between, and within, tasks. Messages 0 and 5 are effectively the same message received by two sub-tasks. Thus, X-Alloc is able to deal with eight separate allocations of sensor fusion sub-tasks and the sub-tasks that receive sensing data. For example, sub-tasks 0, 1 and 6 may be on the same processor. In this case messages 0 and 5 are intra-processor. Alternatively, sub-tasks 0 and 1 are on the same processor, while sub-tasks 0 and 6 are on different processors. In this case message 0 will be an intra-processor message and message 5 an inter-processor message.

Six more scenarios of this kind can be produced. A congruent data source for three receiving sub-tasks is even more complex. This has proved to be one of the limitations on the effectiveness of the current X-Alloc tool.

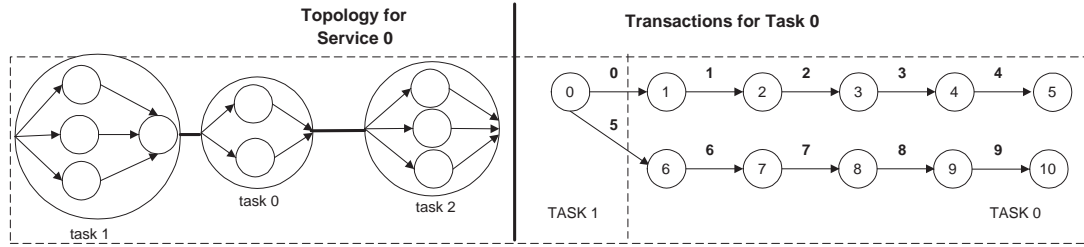


Figure 4.7: Architectural Topology for Illustrative Example 1

In total there are 11 sub-tasks to allocate to processors and 10 messages to allocate to buses. Messages from sensors to sub-task 0 and from sub-tasks 5 and 10 are not included in the analysis. Extensions to include such messages can be envisaged as a step towards incorporating sensors and actuators into the allocation process.

The selected architectural topology for example one and the set of sub-tasks for the service are shown in Figure 4.7. The inter-processor communication network consists of two copies of each message sent in parallel. This has been interpreted as one message through transaction 1 and one through transaction 2. Both sets of messages are implemented on buses of resource type 5.

A second possible interpretation is that each individual message should be sent twice, on separate units of resource type 5. The baseline X-Alloc is unable to support this interpretation. Extending the allocation algorithm may require changes to the message structures and to the distributed offset analysis employed. This extension is considered in Chapter 5.

The deadline, and period, of task 0 is 75ms. Thus, X-Alloc must find an allocation of sub-tasks and messages for which $\max(\text{trans1}, \text{trans2})$ is less than 75. All time measurements have been transformed to milliseconds. In Chapter 3 the MTTF of tasks is expressed in hours. This difference in time scale must be borne in mind when evaluating a selected topology.

Allocation Search

Three X-Alloc experiments were undertaken using the default settings, except k_{cost} was set to 0.01, rather than 0.1. Statistics for a typical experiment are shown in Table 4.2. There were 19 steps recorded for this experiment. The SA algorithm converged rapidly to a solution, requiring only a single application of the cooling schedule. In fact the algorithm converged to the best solution before this cooling occurred. This indicates that this example has a very well behaved, small, search space in which it is ‘easy’ to determine local acceptable minima. The algorithm effectively operated as a hill

climbing search.

Steps	Value	User Time	CPU Time	Number of proposed Moves	Number of accepted Moves	Temp
1	1600005448	0	0.0	0	0	1.0
5	6357	1	0.6	42	26	1.0
10	2335	4	1.6	110	54	1.0
15	645	26	6.2	384	137	1.0
19	435	59	15.0	928	339	1.0
end	435	94	23.0	1428	512	0.95

Table 4.2: Results for Example 1

The best evaluation function, $P(x)$, value was 435, broken down into 155 for the cost of the platform and 280 for the WCRT of task 0. The Service met its WCRT requirements. The best solution employed the following sub-task gene structure values:

Priority 10 1 9 2 7 3 5 6 0 8 4
 Processor 1 4 4 3 3 3 2 2 2 1 1

Thus sub-task 0 is allocated to processor 1 and has priority 10. Similarly, sub-tasks 9 and 10 are allocated to processor 1 and have priorities 8 and 4 respectively. So overall, the number of processing units employed in the platform is 4, allocated as follows ⁴:

Processor 1 [10][9][0] numsub=3 capacity=16
 Processor 2 [8][6][7] numsub=3 capacity=16
 Processor 3 [3][5][4] numsub=3 capacity=16
 Processor 4 [1][2] numsub=2 capacity=16

The best solution employed the following bus residency gene values: 4 * 3 * * 4 * * 4 * *. Starred results indicate communication is intra-processor and therefore the stated bus is irrelevant. This illustrates why the implemented SA is configured to reject proposed message moves that give the same evaluation function value as the existing allocation. If a message is intra-processor any move to a different bus will have no effect on the evaluation function value. In this example two communication buses are employed:

Bus 3 [2] priority 9
 Bus 4 [8][0][5] priority 0,10(1),10(3)

Two messages have the same priority, inherited from the source sub-task. The tie is broken by looking at the priority of the receiving sub-task. Message [0] is given a higher priority than message [5] because it's receiving sub-task has a priority of 1,

⁴[x] = sub-task x allocated to named processor or message x allocated to named bus.

whilst the receiving sub-task for message [5] has a priority of 3.

A total of 256 bytes of inter-processor communication are required per invocation of task 0. No capacity constraints are violated by the chosen allocation. Two buses are employed to keep messages 2 and 8 apart for redundancy purposes. The total number of units required to fulfil both the real time and dependability requirements for processing task 0 is therefore six. Task 0 met its deadline with a WCRT of 28ms, see Figure 4.8.

Message 5 has a WCET of 0. This is because it is physically the same message as message 0. It is included to show how X-Alloc deals with congruent data sources. The number above each circle in Figure 4.8 indicates the processor a sub-task is allocated to. The numbers above arrowed lines indicate the bus a message is allocated to. No number indicates an intra-processor message.

Id	D	Offset	WCET	WCRT	Met?
pst 0	75	0	2	2	1
pst 1	75	3	1	9	1
pst 2	75	9	6	15	1
pst 3	75	16	4	20	1
pst 4	75	20	2	24	1
pst 5	75	24	1	28	1
pst 6	75	3	1	9	1
pst 7	75	9	6	15	1
pst 8	75	15	4	21	1
pst 9	75	22	2	24	1
pst10	75	24	1	26	1
mes 0	75	2	1	3	1
mes 5	75	2	0	3	1
mes 8	75	21	1	22	1
mes 2	75	15	1	16	1

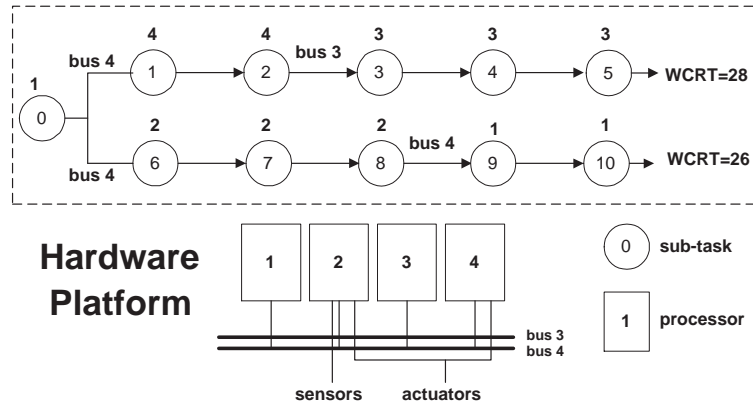


Figure 4.8: Solution to Illustrative Example 1

X-Topmeter predicted a processing platform of at least two processors. The processing platform produced by X-Alloc indicates that in fact four processors are required. This increase in size is partly a result of the need to guarantee a WCRT for the processing task. The allocation tool also attempts to minimise cost, rather than absolute size, of the final processing platform. A cost for size trade-off may have occurred. X-Topmeter employs a fairly naive method, utilisation, to determine the number of units required. The decomposition into sub-tasks and explicit allocation to particular units inevitably produces a more accurate prediction of the required processing platform. X-Topmeter and X-Alloc agree that two buses are required in the hardware platform.

In order to ensure convergence between the results predicted by the architectural topology and the allocation searches the accuracy of the platform size prediction made during the architectural topology search must be as high as possible. Furthermore there is an onus on the designer to investigate any discrepancies, like those highlighted in this example, in the analysis phase of each Analysis-Synthesis-Evaluation cycle. A failure to identify such discrepancies may lead to surprises late in the design process leading to considerable reworking of the design.

In conclusion, once an allocation search can be employed for a proposed architectural topology the designer is able to consider the combined effect of dependability and timing requirements on the topology for the proposed control system. The designer should investigate any discrepancies between the results from the two search engines.

4.4.2 Illustrative Example Two

This example takes the results of the two service architectural topology produced in Chapter 3 and a decomposition of the processing tasks, and attempts to allocate the resulting sub-tasks to an appropriate hardware platform. The architectural topology (left-hand column), sub-tasks and messages (right-hand column) employed are shown

in Figure 4.9. Tasks 1 and 4 are processing tasks and are implemented as dual and triple redundant topologies respectively. Sensing and action tasks 0,2,3 and 5 are not considered as part of the resolution of the allocation problem in this example as none of them employ a decision mechanism.

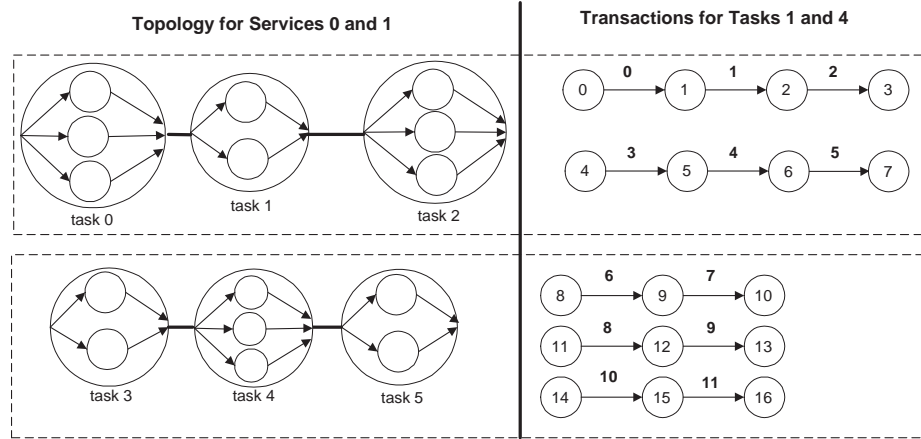


Figure 4.9: Architectural Topology for Illustrative Example 2

This example employs three networks. Only the allocation of messages to net1 (processor to processor) is investigated by X-Alloc. Thus extra buses will be required in the overall platform to accommodate communication in networks 0 and 2. This provides one obvious extension that could be incorporated into a new version of X-Alloc.

Three experiments were performed using X-Alloc. The annealing algorithm for a typical experiment on this example ran for 25 seconds (user time) and 3 steps were recorded, see Table 4.3.

Steps	Value	User Time	Number proposed Moves	Number accepted Moves	Temp
1	30130	0	0	0	4.0
5	9662	2	52	24	3.80
10	1463	15	277	111	3.61
END	1463	46	777	327	3.43

Table 4.3: Results for Example 2

The algorithm converged very quickly to a solution. This is partly a result of the use of X-Alloc at an early stage in the design process, partly a result of the increase in variability of platform size, and partly a result of the architectural topology chosen for this example. In total 777 moves were proposed and 327 accepted. The best allocation

found employed the following values for the three gene structures:

```

Priority    2 8 15 7 14 5 4 3 11 0 9 12 10 13 6 16 1
Processor  4 4 4 4 1 1 1 1 6 5 5 8 9 9 10 10 10
Message    * * * * * 8 * * 5 * *

```

Processors 0-4 are of processing type 4, 5-9 are of processing resource type 5 and 10-14 resource type 6, see Table 3.11 for details. Buses 4-7 are of resource type 9 and 8-11 are of resource type 10. The *s in the message gene indicate that only 2 inter-processor messages were required in the allocation. The processor gene clearly shows the clustering of sub-tasks by transaction. This clustering occurs because minimum WCRTs can be produced for intra-processor messages as they are not subject to blocking or interference from other messages.

X-Topmeter predicted a processor platform consisting of at least 5 processors and 3 buses. A further 5 sensors and 5 actuators were predicted. X-Alloc predicted a processing platform of 7 processors and 2 buses. Three different processing resources and two different network resources are employed⁵. The number of buses has been reduced by the simple expediency of ensuring that all messages in task 1 and transaction 3 are intra-processor. This reduces the cost of the platform, as ownership costs of different resources are a significant proportion of the overall cost of a platform.

The priority ordering of sub-tasks on processors with more than one resident was:

```

proc 1    [7][6][5][4]
proc 4    [0][1][3][2]
proc 5    [9][10]
proc 9    [12][13]
proc 10   [16][14][15]

```

The evaluation function had a value of 1463, see Table 4.3. Only two elements of the evaluation function had non-zero values: Cost with a value of 1383 and the WCRT penalty element with a value of 80. The weightings employed for these elements were 0.1 and 10 respectively. All transactions, and hence both tasks, are predicted to meet their deadlines. The resulting platform is shown in Figure 4.10.

The size of the platform in this example results from choices made in the resolution of the architectural topology for this system and the need to keep transactions in the same task separated. The designer may wish to investigate a number of facets of this result in order to improve the proposed resolution of the topology problem.

For instance, the architectural topology result indicated that at least five processors would be required. This solution uses seven processors, two of which contain a single sub-task. Under what circumstances would it be possible to move sub-tasks 8 and 11 to processors already employed in the platform and thereby reduce the number of

⁵Units (processors or buses) are particular instances of resources (e.g 486 processors).

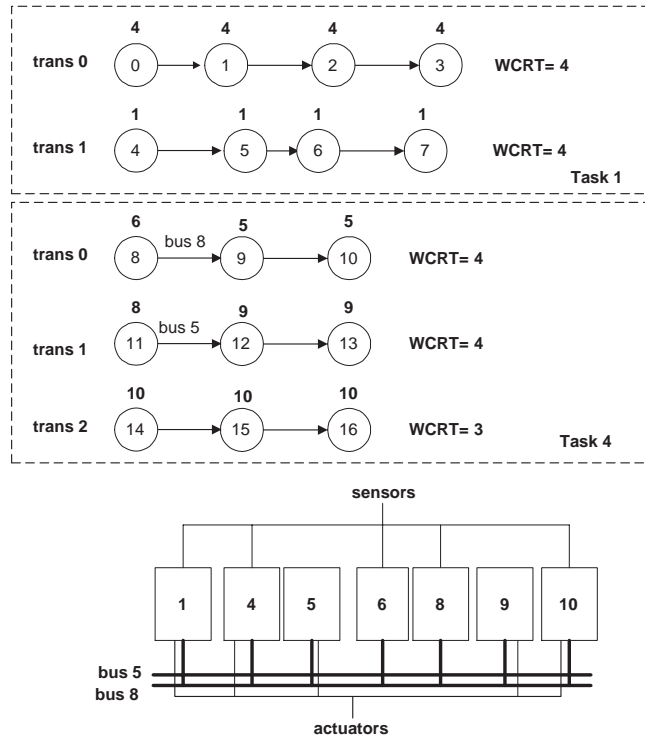


Figure 4.10: Solution to Illustrative Example 2

processors employed to five? It is the ability to investigate the implications of design decisions on the required topology and hence hardware platform early in the process that is a primary impetus for the work presented in this thesis.

4.4.3 Illustrative Example Three

In this example, one possible architectural topology for the five Service control system presented in Chapter 3 is explored. The architectural topology employed is an exemplar one and does not represent any of the many architectural topologies produced in Chapter 3. It has been devised to show the range of architectural topologies that can be addressed by the allocation tool and to show some of the limitations of the current X-Alloc prototype. One set of sub-tasks arising from the tasks in the architectural topology are shown in the right-hand column of Figure 4.11. Processing tasks to be allocated to a hardware platform are shaded.

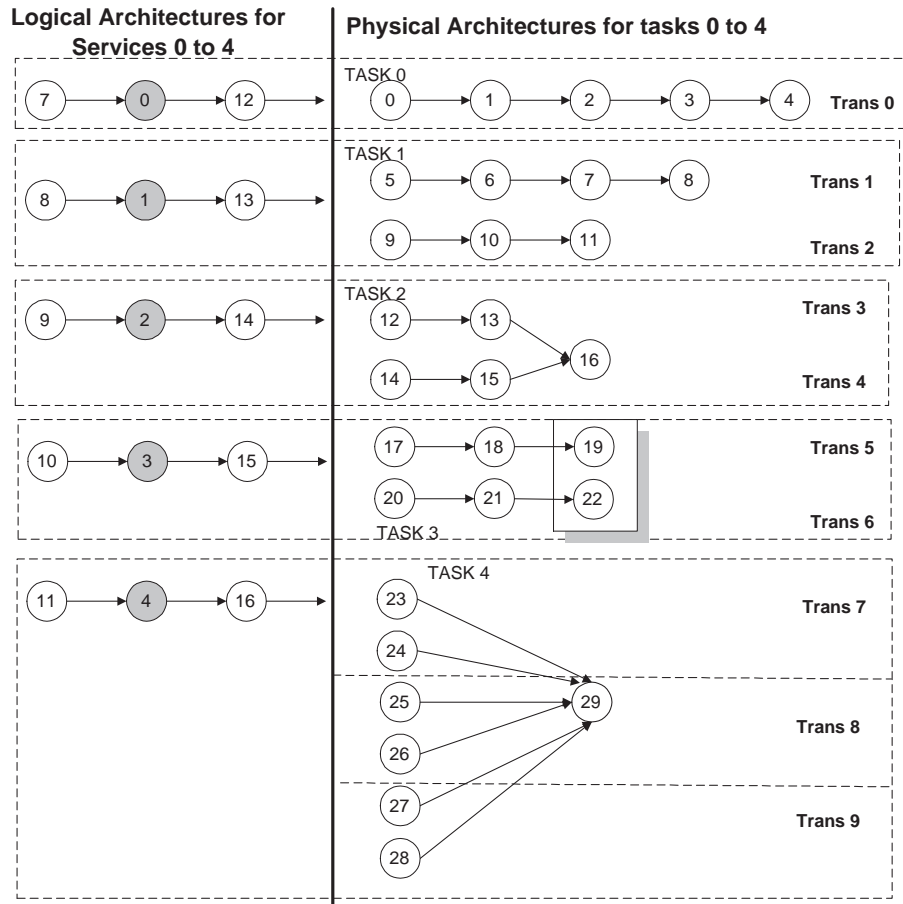


Figure 4.11: Architectural Topology for Example 3

The processing task topologies are simplex, dual-parallel, dual-parallel with voter, dual-parallel with acceptance tests and TMR with hot spares respectively. The deadlines for tasks 0 to 4 are 75, 50, 75, 25 and 75 milliseconds respectively. This results in 30 sub-tasks, 23 messages and 10 transactions for which a WCRT needs to be calculated. Transactions 0, 1, and 5-9 are to be placed on hardware resource two. Transactions 2 and 4 are to be placed on hardware resource three and transaction 3 is to be placed on hardware resource four. Hardware of resource type one is reserved for the decision mechanism sub-tasks (16 and 29). Details of the processing resources employed are given in Appendix A. The set of admissible Processors ($setP$) for this example are shown in Table 4.4.

Sub-tasks	setP	Parallel
0 - 4	10 11 12 13 14 15 16 17 18 19	none
5 - 8	10 11 12 13 14 15 16 17 18 19	9 10 11
9 - 11	20 21 22 23 24 25 26 27 28 29	5 6 7 8
12 - 13	30 31 32 33 34 35 36 37 38 39	14 15
14 - 15	20 21 22 23 24 25 26 27 28 29	12 13
16	0 1 2 3 4 5 6 7 8 9	none
17 - 18	10 11 12 13 14 15 16 17 18 19	20 21 22
19	10 11 12 13 14 15 16 17 18 19	22
20 - 21	10 11 12 13 14 15 16 17 18 19	17 18 19
22	10 11 12 13 14 15 16 17 18 19	19
23	10 11 12 13 14 15 16 17 18 19	24 25 26 27 28
24	10 11 12 13 14 15 16 17 18 19	23 25 26 27 28
25	10 11 12 13 14 15 16 17 18 19	23 24 26 27 28
26	10 11 12 13 14 15 16 17 18 19	23 24 25 27 28
27	10 11 12 13 14 15 16 17 18 19	23 24 25 26 28
28	10 11 12 13 14 15 16 17 18 19	23 24 25 26 27
29	0 1 2 3 4 5 6 7 8 9	none

Table 4.4: SetP for Example 3

The column titled “Parallel” indicates that the row sub-task must not be allocated to the same unit as the named set of sub-tasks. Thus, sub-task 5 should not be allocated to the same processor as sub-tasks 9, 10 and 11. This derived requirement is employed to comply with the structure of the selected architectural topology.

Inter-processor communication between sub-tasks is via buses of hardware resource type five, six or seven, see Appendix A. One network is used in this example for sensor-processor, processor-processor and processor-actuator communication. Up to five units of each network resource may be employed in the platform. The set of admissible buses (setB) for this example is therefore 0 to 14. Messages 0-6, 9, 10, 17, 21 reside on buses of resource type 5, messages 7, 8, 11-14, 18 and 22 reside on type 6 buses and messages 15, 16, 19 and 20 reside on type 7 buses.

Three allocation experiments were undertaken on this example. The experiment resulting in the smallest evaluation function value had a starting temperature of 7 and took 35 fitness value reduction steps to converge to a solution, see Table 4.5. The dispersion of sub-tasks due to the chosen architectural topology has ensured that a large platform is required. In turn this makes it easier for the X-Alloc to converge to an acceptable local minima. Thus the search terminated after 702 seconds. The

designer must now decide whether the final topology is acceptable.

Steps	Value	User Time	Number proposed Moves	Number accepted Moves	Temp
1	44512	0	0	0	7
10	17621	12	81	27	7
20	3606	56	320	106	7
30	3149	192	1067	358	6.32
35	2650	538	3099	1095	6.00
END	2650	702	4099	1402	4.41

Table 4.5: Results for Example 3

The standard parameter settings were employed, except for k_{cost} which took on a value of 0.1. Just over 1400 moves were accepted by the tool. The resulting best allocation of sub-tasks to processors is given in Figure 4.12.

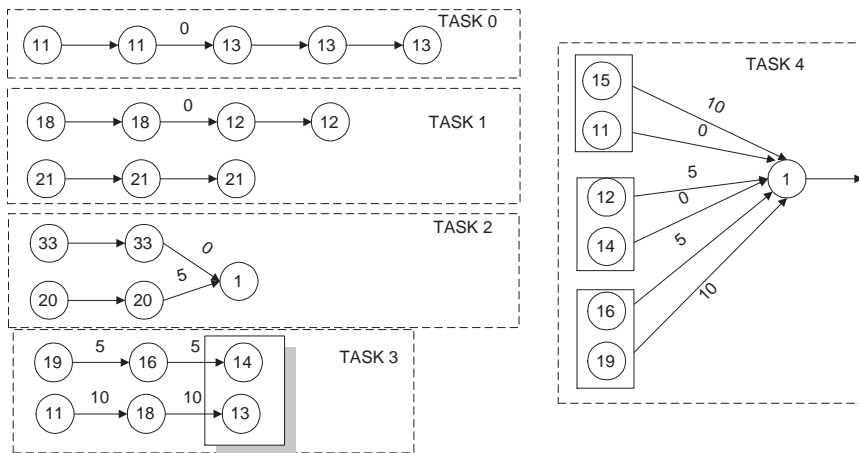


Figure 4.12: Solution to Illustrative Example 3

This solution employed the following gene structure values:

```

Priority    4 0 2 20 10 14 27 13 16 6 23 28 18 29 19 1 24 9 11
           17 8 21 7 5 22 25 26 15 12 3
Processor  11 11 13 13 13 18 18 12 12 21 21 21 33 33 20 20 1
           19 16 14 11 18 13 15 11 12 14 16 19 1
Message    * 0 * * * 0 * * * * 0 * 5 5 5 10 10 10 0 5 0 5 10

```

A processing platform of 11 processors and 3 buses is predicted by X-Alloc as one plausible resolution to this particular allocation, and hence processing topology problem.

The priority ordering on processors with more than one resident is:

Resource 1:	proc 1	[29][16]	Resource 3:	proc 20	[15][14]
Resource 2:	proc 11	[0][20][24]		proc 21	[9][10][11]
	proc 12	[7][8][25]	Resource 4:	proc 33	[12][13]
	proc 13	[2][22][4][3]			
	proc 14	[19][26]			
	proc 16	[18][27]			
	proc 18	[5][21][6]			
	proc 19	[17][28]			

The accompanying communication network employs three buses. Bus 0 carries messages 1,5,10,18 and 20. Bus 5 carries messages 12,13,14,19 and 21. Finally, bus 10 carries messages 15,16,17 and 22. So one bus of each network resource type is employed in the final processing hardware platform.

The evaluation function employed the standard periodic with offset schedulability algorithm and had a value of 2650. Only two elements of the evaluation function had non zero values: Cost with a value of 1470 and the WCRT penalty element with a value of 1180. So all sub-tasks that needed to remain separate were allocated to appropriate processors. All transactions, and hence tasks, met their WCRT deadlines. The WCRTs of tasks 0 to 4 were 8, 22, 37, 19 and 32 respectively. The resulting platform is shown in Figure 4.13.

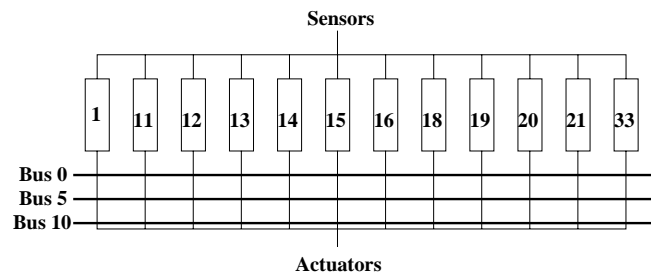


Figure 4.13: Platform For Example 3

The predicted platform appears to be rather large. A designer may wish to consider ways in which this has arisen. Would a relaxation of task deadlines or a tightening of sub-task WCET budgets be sufficient to reduce the size of platform? This question can be addressed by setting up appropriate scenarios and running X-Alloc experiments. If however the size of platform is a result of the underlying architectural topology then the designer needs to re-evaluate decisions made at that level. For example, three processing resources are mandated. Resources 1 and 4 are somewhat under-utilised. Maybe reliability constraints can be relaxed, or the set of available resources changed, or other restrictions placed on the set of admissible architectural topologies. These

questions can be addressed by setting up new scenarios for X-Topmeter to address. In this way the topology of a system can emerge as design progresses.

4.5 Discussion

In this Chapter the results of X-Topmeter, along with a decomposition into sub-tasks and timing requirements, were used to define a variable platform size allocation problem. A tool, X-Alloc, was developed that employs Simulated Annealing, rather than a GA, to predict a hardware platform and an allocation of transactions to this platform. Questions about the approach to the allocation problem developed in this chapter remain. For instance:

1. Why was a single search technique not employed to resolve the GTP?
2. Why has X-Alloc converged to a solution so quickly in the illustrative examples?
3. Why has the link between X-Topmeter and X-Alloc not been automated?

The illustrative examples have shown the qualitatively different nature of the architectural topology and allocation problems. The former is characterised by a computationally expensive evaluation function and the latter by a large set of alternative allocations. To allow a wide variety of architectural topology and allocation problems to be addressed different search techniques are applied to the two problems.

The complexity of the allocation search space may be heavily influenced by the choice of architectural topology. For instance, diversity introduced for fault-tolerance may make it easy to find an allocation that meets all the WCRT targets. This is because relatively few sub-tasks are allocated to each processor. This is the case in the illustrative examples shown in this chapter. Alternatively, if many sub-tasks are allocated to the same resource the X-Alloc tool will find traversing the search space much harder.

The architectural topology of a system emerges as design progresses. Restrictions on the set of admissible architectural topologies can be made during this process for a variety of reasons. For instance, the impact on the platform required to meet timing requirements, of particular architectural topologies, may lead to restrictions on the set of admissible architectural topologies, see Section 5.4.

Some of the links between X-Topmeter and X-Alloc could be automated once these tools have settled down into more stable versions. This is especially true late in the design process once the architectural topology has for all intents and purposes been fixed. However, architectural topology searches can be employed earlier in the design process than allocation searches. It is unhelpful to consider producing a platform and

allocating sub-tasks to it until a plausible set of resources and sub-tasks to be used have been postulated.

There is a more philosophical issue relating to what the tools are attempting to achieve. The aim is to produce a design aid based on quantitative evaluations, rather than an automated solution technique. Thus, forcing the designer to explicitly link the architectural topology to the allocation process in a way that requires some interpretation of results is valuable. If nothing else a clearer insight into the effect of design decisions on the topology required to ensure dependability / timing requirements will result.

A number of additions need to be made to reduce the obligations and restrictions imposed by the assumptions employed in X-Alloc. These extensions should aid integration between the tools. In particular, extensions to deal with multiple networks and sensor / actuator allocations are required. In some cases how to incorporate these additions is straight forward, in others it is not. In particular those additions that call for changes to the schedulability analysis may prove to be the most difficult to make. It must be said that X-Alloc does address issues not previously considered by an allocation tool. In particular setting the size of the processing platform.

In some cases limitations on the set of architectural topologies that may be implemented can result from an inability to model and evaluate the schedulability of the resulting allocation options in X-Alloc. One limitation is the complexity of the DAG of precedence ordered sub-tasks that can be transformed into transactions and therefore modelled and evaluated. For instance, the prototype X-Alloc is unable to model multiple messages being sent between the same two sub-tasks. This is a common FT approach. In Chapter 5 this deficiency will be addressed.

A second limitation is the use of cold spares. Schedulability analysis of cold-spares remains a research issue in the real-time community [134]. X-Alloc currently adds extra time units to the WCET of a task to allow for the switch over. This implicitly assumes an omission failure for a single cycle. It is not clear at present how more effective models of cold-spares can be incorporated.

The platform and allocation predicted by X-Alloc, along with the sensor and actuator platform predicted by X-Topmeter, form one possible resolution to the GTP. The resulting topology is based on information from the designer on the set of possible topologies for dependability, and the quality of the modelling and evaluation of these architectural topologies. Extra design information is then used to produce a platform and allocation based on this architectural topology. There are therefore many factors that have determined the final predicted topology. An appropriate topology will only emerge after many iterations and refinements of the data provided to the tools.

Chapter 5

Case Study and Evaluation of Approach

The aims of this Chapter are three-fold. First, to consolidate the ideas and techniques introduced in Chapters 3 and 4 by presenting a case study. This study takes the form of two versions of a Computer Assisted Braking (CAB) system, see Sections 5.1 to Section 5.3. CAB is taken from a real-world example analysed by the author as part of an industrial collaboration. A third version of CAB is discussed in Section 5.5.6 to show the flexibility and extendibility of the topology selection approach presented in this thesis.

Second, to investigate interactions between the elements of the topology selection process, see Section 5.4. A discussion on interactions between the topology problem and functional design, and the topology problem and other design issues, is presented.

Finally, in Section 5.5 the new and novel topology selection approach presented in this thesis is evaluated to determine whether it has fulfilled three primary aims. The lessons learned from the experiments and case study are also presented.

5.1 Case Study: Computer Assisted Braking System

In this case study an ‘imaginary’ brake-by-wire control system based on research work from the motor industry, is analysed¹. Two functional solutions to this control problem will be postulated and the topology selection process introduced in Chapters 3 and 4, applied to them.

Various x-by-wire systems that eliminate mechanical systems in favour of electronics,

¹A version of a drive-by-wire system has been introduced in the Daimler-Benz F200 design study vehicle.

are already in service around the world. Examples including cruise controls, electronic throttles and aircraft control systems. Boeing [108] have indicated the following reasons for moving to an x-by-wire system:

- Easier to build due to reduction in number of cables, linkages, actuators, etc
- Weight is reduced
- Reliability and maintainability are improved
- Handling characteristics are improved by automatic compensation functions
- Additional protection is provided against inadvertent manoeuvres

The aim of a brake-by-wire system is to remove the mechanical linkage between the brake pedal and the hydraulic brake system, and replace it with sensors on the pedal, and a number of computer-controlled actuators. This allows the manufacturer to improve the braking performance of the car by providing facilities such as anti-lock braking (ABS), emergency stop detection and enhancement, and braking proportional to pedal travel regardless of vehicle load (load compensation).

Anti-lock Braking Systems [151] are now standard on many vehicles. ABS is designed to prevent skidding and help drivers maintain steering control during an emergency stop demand. In cars equipped with ABS the driver keeps a firm foot on the brake, allowing the system to rapidly and automatically pump the brakes. Four-wheel ABS, which allows the braking of each wheel to be controlled, is assumed.

The aim of emergency stop detection and enhancement is to overcome a perceived flaw in ABS technology, the driver! An article in *The Times* on 1st August 1996 noted that researchers have discovered that many motorists instinctively back off after slamming on the brakes during an emergency stop, when they feel the reaction from the ABS. As a result 70% of drivers do not use the full potential of their anti-lock brakes. The emergency stop detection and enhancement facility detects that a genuine emergency stop attempt is in progress and applies the maximum braking capability of the vehicle.

The aim of load compensated braking is for the vehicle to have the same braking characteristics regardless of the size and distribution of the load on the vehicle. The vehicle should brake in the same way if there is a single occupant or if there are five people and their luggage. The detectors for load compensation take the form of pressure sensors on the axles.

Clearly, a “brake-by-wire” system is safety critical (failure to brake on demand is an obvious hazard) and hard real-time (a long response time can lead to unacceptable braking characteristics).

Functional Requirements

The mechanical and hydraulic design of the braking system has already been determined. The braking system context diagram is shown in Figure 5.1. The following specifications are reproduced from Pumfrey & Nicholson [122]. They are not intended to represent current technical knowledge on braking systems, although they are intended to be plausible. Relevant specifications include:

- Hydraulic actuators capable of supplying pressures of up to 200 bar will be employed. Maximum rate of pressure increase and decrease must be better than ± 2000 bar/sec
- The hydraulic line to each wheel will be fitted with feedback pressure sensors to allow closed-loop control and switching to the BASIC actuators
- Each axle of the vehicle will be fitted with pressure transducers to measure the load on the axle
- Each wheel will be fitted with rotation sensor(s)
- The output from sensors, transducers and actuators are assumed to vary continuously with input, with no significant latency. This allows response latencies for sensors, transducers and actuators to be ignored when calculating the response time of any proposed control system.

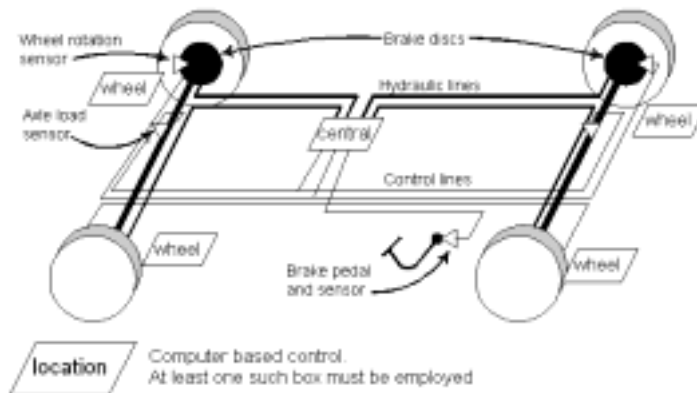


Figure 5.1: CAB System Context Diagram

Performance requirements for the system include a requirement for graceful degradation in case of failures. To accomplish this a “fallback” algorithm which is capable of running independently on redundant hardware units, and which is capable of providing minimum braking is mandated. Furthermore, it is mandated that this fallback algorithm should be able to employ physically different actuators to the primary braking

system. This implies that the system must be able to detect a failure in the actuators via feedback sensors. It also implies that the allocation algorithm must place the fallback software onto different processors than the ABS software.

Topological Requirements

The highly safety-critical and real-time nature of the application has lead the manufacturer to impose further design constraints:

1. The MTTF of each Service must be no less than 26280 hours, representing three years of continuous usage. Assuming that usage is not continuous and failure rates are lower during inactivity a MTTF of three years is deemed to be adequate for the lifetime of the vehicle.
2. The maximum permissible latency from pedal movement to brake effect is 40 ms.
3. The computer hardware implementing the design must have redundancy.
4. There must be redundancy in any communication system used between the hardware units.
5. Topologies are restricted to be a maximum of TMR with two spares per lane.

Topology requirement one sets the reliability side constraint targets for the architectural topology tool. Topology requirement two sets the period of the computing system and therefore the WCRT targets for each control Service. Requirements three to five imply restrictions on the search space available to a designer.

Initial requirements and functional analysis have led the designers to postulate a set of logical architectures [23] for the *brake-by-wire* control system. In Version 1 it is postulated that an integrated computing platform employing three communication networks is appropriate. In Version 2 an integrated computer system that employs just one data network is investigated.

Elements of a CAB System

Versions of the CAB system are presented in Sections 5.2 and 5.3. X-Topmeter addresses the Architectural Topology problem. X-Alloc uses the proposed architectural topologies to predict a processing platform and accompanying allocations of software based sub-tasks and messages. Extensions to the existing prototype tools are presented.

The topology design space traversed by X-Topmeter is bounded by the set of admissible architectural topology components. The limitations placed on the set of admissible architectural topology components for a CAB system are:

- 1 Minimum number of resources employed for each task is 1
- 2 Maximum number of resources employed for each task is 3
- 3 Maximum number of copies per resource for each task is 3
- 4 Tasks can employ hot or cold spares
- 5 Tasks can employ acceptance tests or majority voting
- 6 Minimum number of resources employed for each network is 1
- 7 Maximum number of resources employed for each network is 3
- 8 Maximum number of copies per resource for each network is 3
- 9 Networks can employ hot spares

The dependability of a proposed CAB system is calculated on the basis that for the system to fail to provide any braking both the ABS and fallback BASIC functionality must fail. Each CAB control service is given a dependability target in the form of a MTTF requirement. A number of reliability models are available to calculate this measure for different proposed architectural topologies, see Appendix A.

Dependencies between tasks and services are introduced in the extensions to X-Topmeter used to investigate this case study. System designers need to assure themselves that the current set of reliability models is acceptable in these circumstances. The hierarchy introduced in Section 2.4.2 indicates the expressive power of the reliability models that may be employed to predict the dependability of a proposed topology.

Versions 1 and 2 of CAB are assumed to employ priority based scheduling. Processing sub-tasks and messages are modelled as periodic with offsets, which allows the schedulability analysis introduced in Section 2.6 to be employed. X-Alloc picks the offsets for each sub-task and message in a proposed allocation. The value this offset may take for any sub-task is constrained by the WCRT of the previous message in the system DAG. If a sub-task has no predecessors it is given an offset of 0.

In Table 5.1 the set of resources that may be selected for a CAB System is presented. Sensors provide data on four different characteristics of the environment. Three types of actuator are required. Note that some sensing resources are not only from different vendors but employ logical diversity, that is qualitatively different means of sensing the required characteristic. Three network resources are included in the library. In total X-Topmeter may select from 28 resources.

Data for this table was gleaned from a number of sources including the US military standard Hdbk-mil-std-217E [166], Bowles [16], ARINC [74], members of Rolls-Royce Aeroengines division [35], members of Airbus and web-sites of sensor, actuator and processor vendors. The actual numbers used however are inventions based on this

data. Commercial organisations record failure rate data but keep it out of the public domain. Data on hardware resources includes coverage factors for the hardware based on no decision mechanism, acceptance and voter decision mechanisms.

Resource	Type	Description
0-2	Pedal sensor	Indicates how hard the brake pedal has been pressed
3	Rotation sensor	Toothed wheel, and an edge detector. These sensors are accurate for rotational speeds of between 0.35 and 56 rps (equivalent to 1.8 to 288 km/h).
4	Rotation sensor	Rotating disc with alternate conductive and insulating sections. The conducting sections ground a voltage from a carbon brush, producing a square wave.
5	Rotation sensor	A toothed wheel, but this time it spins between a light source and an opto sensor
6-8	Pressure transducer	Measures load on the axle
9-11	Pressure sensor	Hydraulic pressure sensors to provide feedback on actual braking pressure applied.
12-14	Annunciation	Indicates state of brake-by-wire system to driver
15-17	Feedback actuator	Provides feedback pressure onto brake pedal
18-21	Pressure actuator	Hydraulic actuator: supplies pressures \geq +/- 2000 bar/sec.
22	Data Bus 1	Bus from Vendor 1
23	Data Bus 2	Bus from Vendor 2
24	Data Bus 3	Bus from Vendor 3
25	Proc 1	Processor from vendor 1
26	Proc 2	Processor from vendor 2
27	Proc 3	Processor from vendor 3

Table 5.1: Library of Admissible Resources

5.2 CABV1: Integrated Architecture with 3 Networks

In an integrated CAB system environmental data is sent to a central processing platform which calculates the appropriate control response and drives all four-brake actuators at the required pressure, see Figure 5.2.

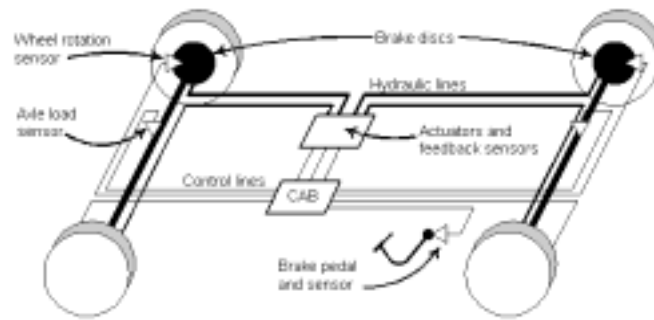


Figure 5.2: Integrated CAB System

Six Services are provided by the system:

- | | | |
|-----------|-----------|--|
| Service 0 | BASIC: | Fallback functionality. Provides braking relative to pedal sensor input |
| Service 1 | ABS: | Performs anti-lock braking for each wheel |
| Service 2 | LOAD(1): | Performs load balancing calculations for front two wheels |
| Service 3 | LOAD(2): | Performs load balancing calculations for rear two wheels |
| Service 4 | STOP: | Performs emergency stop detection calculations and actions |
| Service 5 | FEEDBACK: | Checks feedback sensors for actuator failures. Performs driver feedback in the form of back-pressure on the brake pedal. |

Two services are used for the enhanced braking (LOAD) facility as the effect of load characteristics on the axle with the driving wheels will be different to that on the axle with the passive wheels. The actuators employed in Service 0 are physically different to those employed in Services 1 and 4. This is implemented to fulfil the logical redundancy requirement for the braking activity. Further redundancy can be added by the X-Topmeter tool.

In Figure 5.3 the CAB Version 1 system description is shown. The engine bay of a motor vehicle is a very hostile environment for communication devices due to the possibility of electromagnetic interference and noise (EMC) [111]. As a result it has been decided that separate communication networks will be employed between the sensors and the computing platform, within the computing platform, and between the computing platform and the actuators. Values in parentheses in Figure 5.3 indicate the minimum number of units of that type required to implement the system.

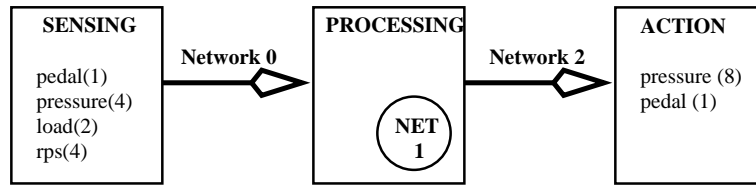


Figure 5.3: CABV1: System Description

The processing tasks employed in Services 0 to 5 can be further broken down into a set of sub-tasks. The logical architecture for these tasks is shown in Table 5.2 and Figure 5.4.

Service	Sub-tasks
0	pedal sensor - In - basic - monitor - out - BASIC pressure actuators
1	pedal & wheel sensors - In - ABS - monitor - out - ABS pressure actuators
2	front axle sensors - In - load - monitor - out
3	rear axle sensors - In - load - monitor - out
4	pedal sensor - In - emergency - monitor - out - ABS pressure actuators
5	pressure sensors - In - Monitor - Out - pedal actuator

Table 5.2: CABV1: Sub-tasks

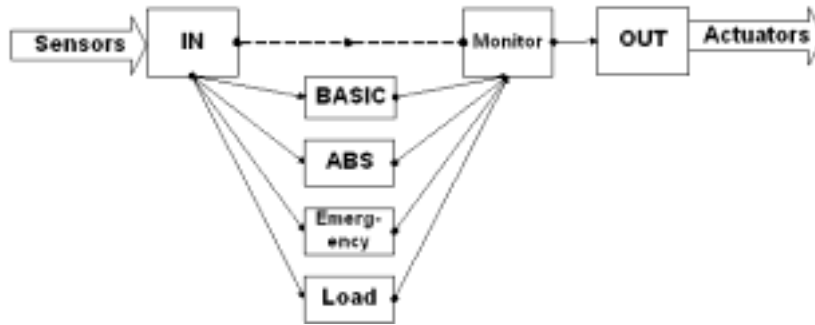


Figure 5.4: CABV1: Logical Architecture of Processing Tasks

One critical shared component of these Services is the OUT sub-task. This sub-task takes results from the processing tasks in Services 0 to 5 and calculates the final brake pressures to be applied.

5.2.1 Implementation of X-Topmeter and X-Alloc

X-Topmeter has previously been implemented using binary search strings, see Section 3.3.1. However, the latest version of the *X-GAmet* environment [104] allows

integer representations of string alleles. X-Topmeter was therefore updated to X-TopmeterV2. Each task now employs a gene consisting of 4 integers. Similarly, each network employs a gene consisting of 3 integers. New functionality has been added to allow each version of the CAB system to be analysed. The default X-Topmeter parameter set remains unchanged.

Five features of CAB Version 1 required changes to the prototype X-Topmeter tool:

1. Service 1 employs data from sensor task 0 (brake pedal sensing), as well as data from wheel rotation sensors, to calculate the ABS breaking pressure to each wheel. The resulting pressures are further modified by the load values. Thus the topologies for tasks 0 and 3 have been explicitly linked.
2. Services 2 and 3 employ dummy actuators. There is no direct throughput to the actuators via these Services.
3. The actuators employed in Services 1 and 4 are the same physical actuators. Similarly, the sensors in services 0 and 4 are the same sensor units.
4. The processors employed in Service 0 and 1 are separate hardware units.
5. The Sensing task for Service 1 must provide data on the rotation of all four wheels. This task is therefore a *k-out-of-n* task with k and n taking a value of 4. The sensing task for Service 5 and actuators for services 0 and 1 are also modelled in this manner.

To allow a task to use data from another task a new element has been added to the structure of each service (*servicemap[i].joint[r]*) indicating whether a result from another sensing [r=0], processing [r=1] or action [r=2] task is used by the ith service. The topology for the joint task must have already been processed this iteration of X-Topmeter. This restriction is not onerous as tasks and services can be presented in any order to X-Topmeter. The structure of the Service reliability model has been altered to take into account joint tasks. It is assumed that all the joint tasks must provide an output for the overall task to operate correctly. As a result the joint task is modelled as a series RBD.

A dummy task is given a model number of 200. This indicates to X-Topmeter that a Service reliability model with only two tasks should be employed. It also indicates that 0 cost, perfect task reliability and 0 size should be ascribed to this task. The task remains part of the system and the designer can consider the effects of a task simply by removing the 200 value and stating appropriate resources.

To model the use of the same actuator units in two or more services the gene values for the appropriate action tasks are linked. For instance, the gene value for action tasks

2 and 14 are forced to be identical. The evaluation function is altered to give a zero cost and size to the action task 14.

Altering the evaluation function to accommodate the requirement for task 1 to use a physically different processor to task 4 is straightforward. The adjustment is only required if both services use a processor of the same type.

In a k-out-of-n (kofn) structure a (sub)-system can function if k or more components are working. Hitherto a 1-out-of-n system has been assumed in this thesis. A kofn model is required for the internal structure of the sensing tasks 3 and 15. Sensing task 3 produces four rotation sensor data streams. It is therefore assumed that any failure will cause a failure of the ABS and a 4-out-of-4 model is required. Alternatives, such as 2-out-of-4 could be implemented if only a diagonal pair of sensors need to work for safe braking.

X-TopmeterV2.1 uses the existing three network illustrative example as a baseline search implementation. The services, tasks and admissible resources for CABV1 are given in Table 5.3. The standard evaluation function has been amended to take note of the use of dummy tasks and shared units.

Service task	0			1			2			3			4			5		
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
hware1	0	24	18	3	24	18	6	24	*	6	24	*	0	24	18	9	24	12
hware2	1	25	19	4	25	19	7	25	*	7	25	*	1	25	19	10	25	13
hware3	2	26	20	5	26	20	8	26	*	8	26	*	2	26	20	11	26	14
Net	0	1	2															
hware1	21	21	21															
hware2	22	22	22															
hware3	23	23	23															

Table 5.3: Resource set for CABV1

A pre-emptive priority-based scheduler is proposed for the CABV1 design. To meet the requirement for a 40 ms latency from pedal movement to the start of braking activity, a 20 ms period is employed for the IN and OUT modules. This implies an end-to-end deadline for the Services, and hence processing tasks in the system, of 20 ms.

The X-Alloc tool has been extended to accommodate two deficiencies of the prototype X-AllocV1.5, namely

1. ability to deal with one sub-task receiving data from multiple sources.
2. ability to employ multiple messages between the same two sub-tasks

In X-AllocV2.1 the set of data items each sub-task employs is logged. These data items imply messages that must arrive before the sub-task can be processed. Thus, the offset of a sub-task is adjusted to be the maximum WCRT of the messages it requires. For instance, if there are two messages incident on a sub-task the offset for the sub-task is given by $O_{pst} = \max(WCRT_{m1}, WCRT_{m2})$.

To be conservative this maximum WCRT rule is also applied to a topology where multiple messages are sent between the same two sub-tasks. A less pessimistic rule for this case would be to take the WCRT of the first message to arrive. The tool may be easily changed to employ this rule as a WCRT minimisation function is provided in X-AllocV2.1.

The priority of multiple messages with the same source and destination remains an issue. In X-AllocV2.1 if messages of the same priority are employed on the same bus then the id number is used to break the tie. Other more complex rules may be envisaged. However, since in this case all such messages *should* be placed on different buses, for fault tolerance purposes, this rule has proved adequate.

5.2.2 Results from X-Topmeter and X-Alloc

Two sets of experiments have been undertaken using X-TopmeterV2.1 to set a *free-form* and a *full redundancy* architectural topology for the CAB system. In the free form version redundancy is implemented in the form of physically separate implementations of the BASIC and ABS services. In the full redundancy version all tasks are subject to redundancy.

The service models have been refined over four iterations around the problem using X-Topmeter. Initially, for instance, the way to deal with the use of the same physical hardware units in multiple services was not clear. Similarly, the reliability model that should be employed for each action task became clearer once an initial architectural topology had been selected.

The data generation process for the initial iteration was expensive in time and designer effort. In particular, producing the data libraries and system structures in a form X-Topmeter could interpret was complex and time consuming. Future prototypes will need a better GUI for this aspect of the tool. The *turn-around* time for subsequent iterations was relatively quick. The most time consuming part proved to be coding of new reliability models for specific tasks. It was found that the need to interpret the results for a particular experiment made the structure of the proposed system, and its interaction with a hardware platform, much clearer.

The architectural topology for services resulting from a typical free-form experiment are shown in Figure 5.5. Results from task, t0, are used in service 1. Each resource

5 sensor in task 3 and resource 9 sensor in task 14 implies 4 wheel sensors. Similarly each type 18 resource actuator employed in services 0 and 1 implies 4 hardware units. The dotted boxes labelled with an 18 are the same physical units. Thus, failures of service 1 and 4 are linked.

The processors implied by the use of resource 24 in services 0 and 1 (see Figure 5.5) are physically different processors. Thus, the minimum size hardware platform for this topology consists of 20 sensors, 17 actuators, 9 buses, and 4 processors. Sensors have been subjected to the most replication because they are relatively inexpensive, fail at a rate an order of magnitude above other hardware resources and must be relied on for the services to work. These results are consistent with real-world control systems. This topology is predicted to meet the given reliability requirements.

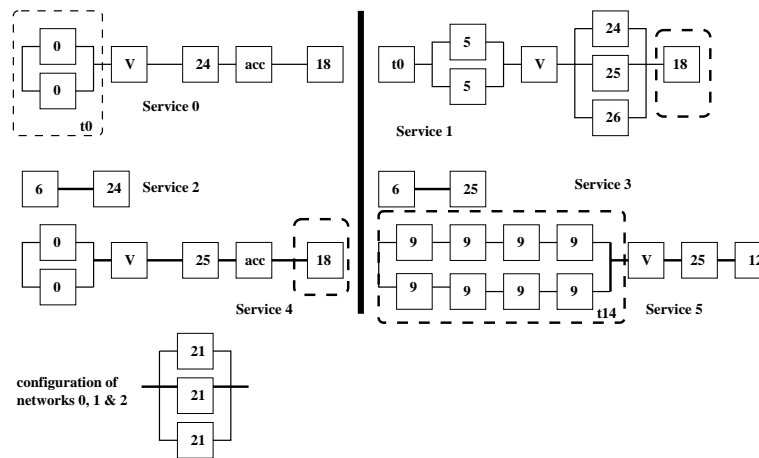


Figure 5.5: CABV1: Free-form Architectural Topology

One area the designer may wish to investigate in the next version of the CAB design is the large number of buses employed in the system. Each communication network is triple redundant, that is three copies of each message are sent. Analysis of the data volume for each bus should be undertaken. This analysis is one of the reasons why a single IMA network is employed in CABV2.

In Table 5.4 statistics on the steps taken by the algorithm are presented. The solution (soln) value represents the evaluation function value of the best architectural topology in the pool of solutions. The evaluation function penalises alternatives that miss reliability targets. It also penalises alternatives that indicate more costly (or larger) topologies more than those that indicate cheaper (or smaller) topologies.

The average fitness (Av Fit) value is the mean evaluation function value for the pool of 20 solutions. Note that the average fitness may increase despite a reduction in the value of the best solution in the pool. This experiment took 6 hours 10 mins to complete on a Sparc workstation. A value within 10% of the best value was found

after 12338 seconds (3 hours 26 mins).

Steps	Soln	Time	Gens	Evalns	Av Fit
1	7569470	0	0	20	537642035
10	462653	3654	32	207	626611
20	276748	5651	105	626	317569
30	183947	7232	241	1383	27901989
40	148919	12338	667	3876	162997
45	139450	15139	898	5274	39275
END	138366	24749	1678	10001	139436

Table 5.4: Results for Typical CABV1 Free-form Experiment

Results for the ‘forced redundancy’ architectural topology are shown in Figure 5.6. The minimum hardware platform implied by this topology consists of 23 sensors, 9 actuators, 9 buses, and 4 processors.

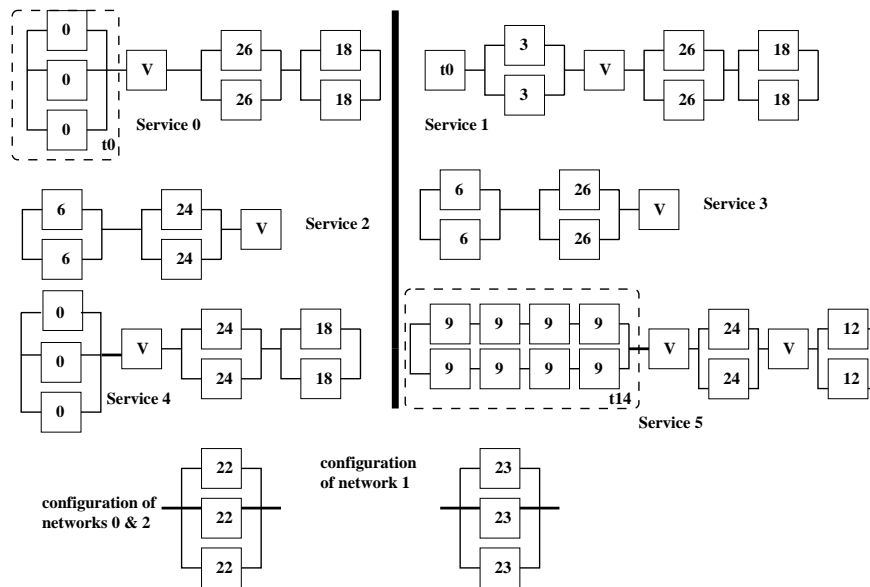


Figure 5.6: CABV1: Redundant Architectural Topology

In this example 3 more sensors and 9 more actuators are employed than in the free-form version. The size of the minimum platform is mainly increased because of the extra actuators employed. All services are predicted to meet their MTTF targets. In Table 5.5 statistics on the steps taken by the algorithm are presented. The minimum value topology took on a $P(x)$ value of 204366. This experiment took 5 hours 40 mins to complete on a Sparc workstation. A value within 10% of the best value was found after 8110 seconds (2 hours 15 mins).

Steps	Soln	Time	Gens	Evalns	Av Fit
1	34938753	0	0	20	554213800
10	423593	5369	52	332	432601
20	353033	5942	78	517	385055
30	261143	7259	169	1018	277612
40	218745	8526	272	1598	224832
50	217872	11103	465	2781	217881
60	204366	13735	643	3853	204367
End	204366	20295	1143	6802	204366

Table 5.5: Results for Typical CABV1 Redundant Experiment

The X-Alloc tool is employed to find a mapping for the processing tasks in the proposed architectural topology. The DAG of processing sub-tasks and messages to be mapped is shown in Figure 5.7. This DAG incorporates elements of Figures 5.4 and 5.5.

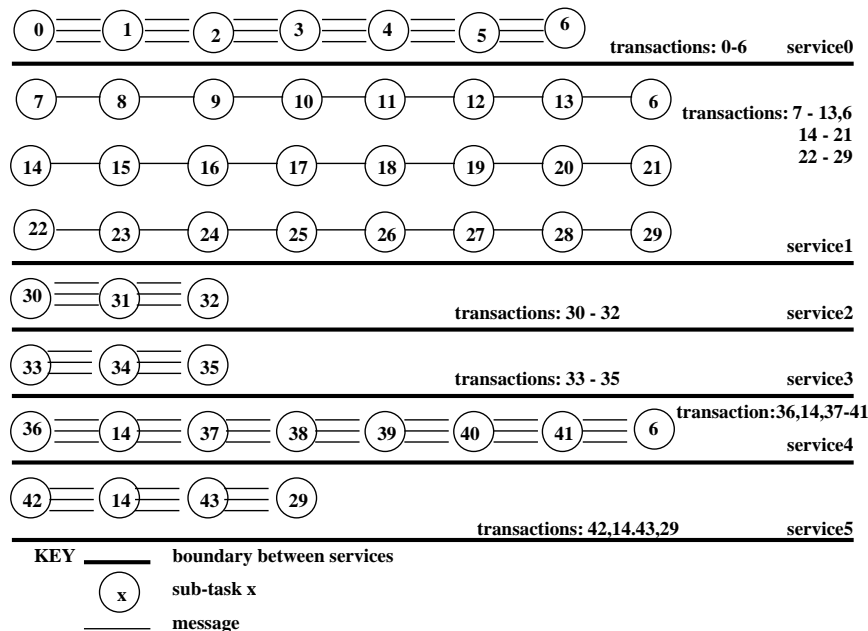


Figure 5.7: System-DAG of CABV1 for Free-form Architectural Topology

A set of statistics for a typical X-Alloc experiment for this problem are shown in Table 5.6. The solution value (soln) is a measure of the cost, resource usage and timing characteristics of the proposed allocation and accompanying processing platform. The temperature (temp) value indicates the probability of a worse solution being accepted per iteration. The experiment ran for 41 minutes on a Sparc workstation and a smallest P(x) value of 15406 was found. The temperature parameter cooled from 12 to 0.92

during the experiment. A value within 10% of the best value was found in 30 minutes. During the run approximately 50% of proposed moves were accepted.

Steps	Soln	Time	Proposed	Accepted	Temp
1	181943	0	0	0	12.0
10	92493	292	918	517	12.0
20	70527	846	1949	1120	10.3
30	47681	1472	3457	1935	4.53
40	21655	1794	4508	2384	2.85
45	15406	2179	5841	3008	1.47
END	15406	2430	6841	3475	0.92

Table 5.6: Results for Typical CABV1 Free-form Allocation Experiment

X-Alloc produced a platform and allocation that met all the timing requirements on the system, see Figure 5.8. Service WCRTs are predicted to be 20,20,10,3,20,19 respectively. In total 18 hardware units were employed, consisting of 5 processors of type 24, 3 processors of type 25, 3 processors of type 26 and 7 buses of type 21.

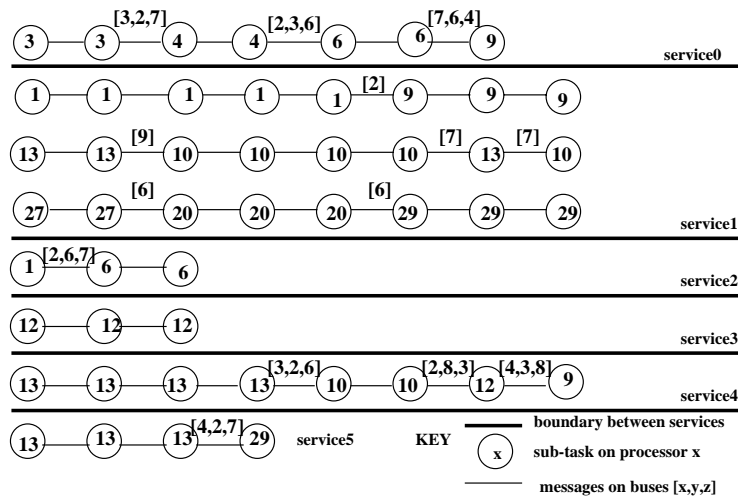


Figure 5.8: Results of CABV1 Free-form SA

The topology predicted for this version of CAB implies a large platform. The amount of data being transferred between elements in the system is somewhat greater than predicted during the architectural topology experiment. In particular inter-processor communication in services 1 and 5 exceeded the expected level. This data can be put back into the X-Topmeter tool for a second iteration of the selection process.

These results are similar to the findings of a qualitative analysis of this version of CAB by the author [106] and students attending the MSc in safety critical systems

at the University of York. This is an encouraging result for this approach. Note that a negative result is still useful in terms of the design process. Safety analysis also highlighted the sharing of processing resources exhibited in Version 1, especially the critical nature of OUT, as a problem. Thus the designer decides to produce a second variant with this information in mind, CABV2.

5.3 CABV2: Integrated Architecture with IMA Network

In the light of the X-Topmeter / X-Alloc results, and other design information, for CABV1 the designers have produced an alternative set of sub-tasks to implement the processing task for the CAB system, see Figure 5.9². The DAG of precedence constraints between sub-tasks and the interactions between these sub-tasks and the sensing / action tasks have also changed. The generic architecture for the system remains unchanged from CABV1.

In each period, a processing channel calculates a basic braking value from the pedal sensor value. The values of all the other sensors in the system are then used to determine whether the three modifiers (i.e. Anti-lock, Emergency stop enhancement and Load compensation) are required in the current cycle, and calculate the necessary changes in braking for each wheel to implement these modifiers. This implies that a decision mechanism is always employed in the architectural topology for each processing task. A channel may employ multiple processors and send messages over buses.

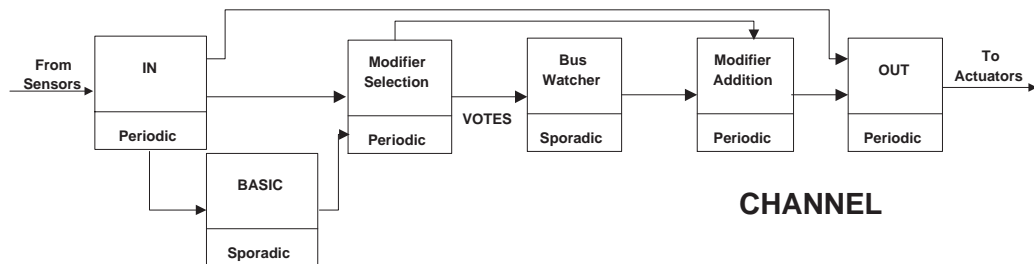


Figure 5.9: CABV2: Logical Architecture

The number of channels in the proposed topology determines the type of decision mechanism employed. If a single channel is employed an acceptance test is used to determine which modifiers to use. If two channels are used then both must agree. If three channels are used a modifier is added if at least two of the channels have determined that it is required. The actual amount by which braking at each wheel is to be increased or decreased to implement the modifiers is not communicated, as it is so dependent on the precise value of the sensors read by each channel. This means

²Figure 5.9 is reproduced from Pumfrey [133].

that, if one channel has not calculated a value for a modifier that has been voted necessary (i.e. other channels require it), then this channel must revert to the basic value initially calculated.

Analysis of the previous design has led to a decision to implement communication between tasks as a single IMA network. Calculation of the control action remains centralised, with hydraulic pressures being transmitted to the wheels. In Figure 5.10 the revised system is shown. In fact this version retains the same set of Services as Version 1.

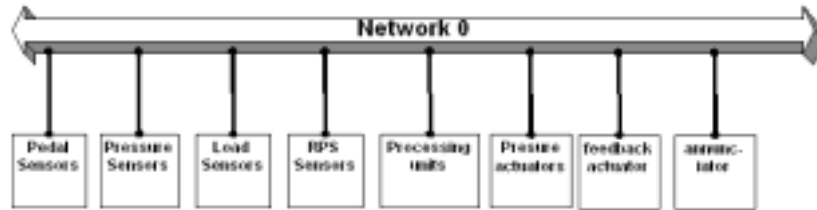


Figure 5.10: CABV2: System Description

Implementation of X-Topmeter and X-Alloc

The logical architecture for V2 implies a simple X-Topmeter search space for the processing tasks. The aim is to implement each channel, consisting of all 6 processing tasks, on a single processing resource. In fact, wherever possible a single processor should be employed. This implies that all processing tasks employ the same architectural topology component. Furthermore, voting must take place and therefore the processing tasks must employ a voter. The free-form and redundant experiments undertaken for CABV1 are amalgamated. At least two sets of processing channels are to be employed. Redundancy is also mandated in the sensing and action tasks. So, to implement X-TopmeterV2.2 three changes to X-TopmeterV2.1 are required:

1. Change the input data to reflect the changed WCET of processing sub-tasks and WCCT of messages.
2. Change the search space to reflect the fact that the logical architecture mandates a redundant architecture with decision mechanism. Sensing tasks may also employ decision mechanisms as part of IN.
3. Change the network structure to reflect the use of a single IMA network

Once the selected architectural topology and logical architectures have been combined a simple timing model of the proposed CABV2 processing tasks can be produced. For WCRT analysis the system can be decomposed into a single Service in which

one, two or three transactions are employed. Each transaction contains six sub-tasks: IN, BASIC, Modifier Selection, Bus Watcher, Modifier Addition and Out. As design decomposition occurs these sub-systems may be decomposed into a greater number of sub-tasks. The X-Alloc tool does not need to be changed from that employed in CABV1. The data input to X-Alloc needs to reflect the structure of the new logical architecture and proposed architectural topology.

Results from X-Topmeter and X-Alloc

The search space for the processing tasks in X-TopmeterV2.2 is severely constrained. However, the architectural topologies for the sensing and action tasks must also be determined. The reliability model for each service therefore has identical processing task elements, but may employ different sensing and action task topologies.

In Figure 5.11 the results for a typical X-TopmeterV2.2 experiment are shown. The architectural topology for each processing task employs 2 channels and 3 copies of each inter-processor message are sent along type 22 hardware resources. All six services are predicted to meet their MTTF requirements. The implied minimum size of platform is 47, consisting of 22 sensors, 18 actuators, 4 processors and 3 buses. The minimum $P(x)$ value was 172895.

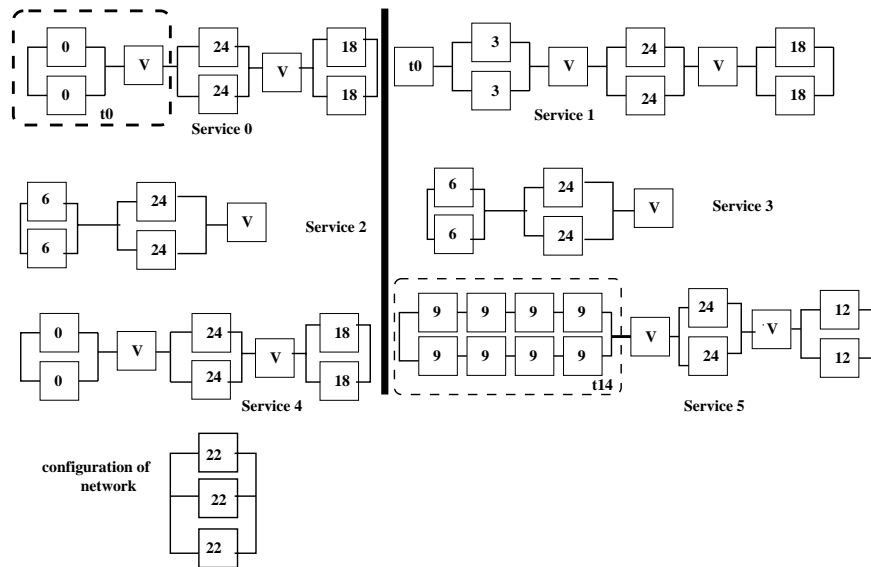


Figure 5.11: CABV2: Architectural Topology

In Table 5.7 statistics on the steps taken by the algorithm are presented. This experiment took 2 hours to complete on a Sparc workstation. A value within 10% of the best value was found in less than 30 mins. This reflects the highly constrained architectural topology search space for this design.

Steps	Soln	Time	Gens	Evalns	Av Fit
1	467750	0	0	20	950158197
10	298289	859	92	204	349231
20	187303	2423	375	770	187570
30	172906	4675	947	1914	173155
END	172895	6992	1460	2940	174181

Table 5.7: Results for Typical CABV2 Experiment

X-Alloc is employed to find a mapping for the processing elements of the architectural topology represented by Figure 5.11. The DAG of sub-tasks and messages to be mapped is shown in Figure 5.12. The main requirement on the tool is to produce appropriate priority and bus assignments to meet the WCRT of 20ms for each channel. The aim is to place the sub-tasks on two processors; one per channel.

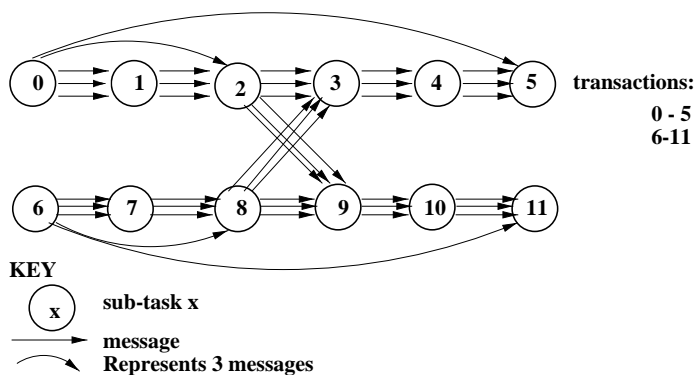


Figure 5.12: DAG of CABV2 Processing Topology

Statistics for a typical run of X-AllocV2.2 are shown in Table 5.8. The tool converged rapidly to a solution, due to the nature of the search space. The total run time was less than 7 minutes. This was not surprising as the tool only had to allocate 12 sub-tasks and 38 messages. As expected the temperature parameter fell during the allocation process. A reasonable result was found very quickly and most of the search effort went into producing slightly better results.

Steps	Soln	Time	Proposed	Accepted	Temp
1	934941	0	0	0	2.00
15	38185	37	254	143	2.00
20	940	117	820	465	1.90
25	703	270	2025	1155	0.98
END	703	393	3025	1697	0.58

Table 5.8: Results for Typical CABV2 Allocation Experiment

The allocation resulting from this search is shown in Figure 5.13. Unfortunately, the tool was unable to meet the timing deadlines and keep all sub-tasks in a transaction (channel) on the same processor. In total four processors and 5 buses are required. However, this platform ensures that the WCRT requirements are met.

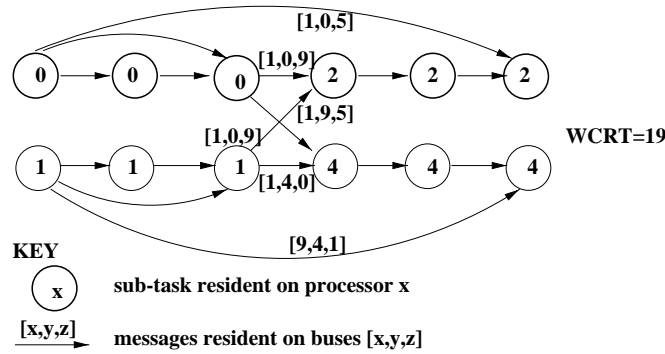


Figure 5.13: Results of CABV2 SA

The results of this system design are much more promising than those for CABV1. The data load on the system is manageable. However, since a single processor channel is not possible, timing requirements remain a problem. The designer may wish to investigate faster processors, which will require re-running X-Topmeter. Alternatively more stringent WCET budgeting may be applied. This would only require a re-working and re-running of the allocation process. The SA employed here is still at a fairly high level of abstraction. More detailed allocation searches can be employed as the sub-systems are developed.

Concluding Remarks

In Sections 5.2 and 5.3 two integrated platforms and accompanying allocations for the six service CAB system have been postulated. The deficiencies in the topology for CABV1, namely large quantities of data transfer and an excessively large platform,

were used to postulate a new set of processing sub-tasks for CABV2. The computing platform for CABV2 was much simpler than that for CABV1. The architectural topology problem for this version was essentially to determine a set of sensors and actuators and the number of computing channels so that Service MTTF requirements could be met. For timing purposes a very simple system-DAG was employed and an appropriate platform and allocation found. Subsequent iterations of the design and topology selection processes will need to concentrate on methods of reducing the computational load, or speeding up processing, so that each channel can be placed on a single processor. CABV2 appears at this stage to be a design that could provide acceptable reliability and timing characteristics.

5.4 Interaction Between Topology Selection and System Design

Interactions exist both between elements of a topology and between a topology and the system design. At a particular level of design decomposition a proposed control system has a number of attributes such as functionality, cost, set of admissible resources to implement it, resources currently employed to implement it, data communication requirements, timing attributes and dependability attributes. These attributes initially take the form of requirements, but predictions and estimates of the attributes of the final system are added as design progresses. The topology selection process first produces an architectural topology. This topology is then annotated with timing attributes and an allocation of software units, and messages, to a hardware platform.

Consider a high level functional design for a control system. To address the architectural topology problem, transition 1 in Figure 5.14, this functional design is transformed into a set of services, each of which contains 3 tasks and at least one communication network. For instance the CAB example can be modelled as 6 services. The set of attributes is similarly transformed to determine the dependability requirements for each task and service in the system.

This representation of the system is subjected to a search for an architectural topology. As part of this search the reliability of each task, communication network and service is modelled, and MTTF values predicted. The search also produces information on the resources employed. The resulting architectural topology is combined with the functional design to produce a new representation of the system, which incorporates functional units employed for reliability and fault tolerance purposes.

Interactions between different runs of the architectural topology tool and an architectural topology and system design (feedback link 2) can now be seen. For instance, the designer may restrict the set of admissible architectural topologies in a search based

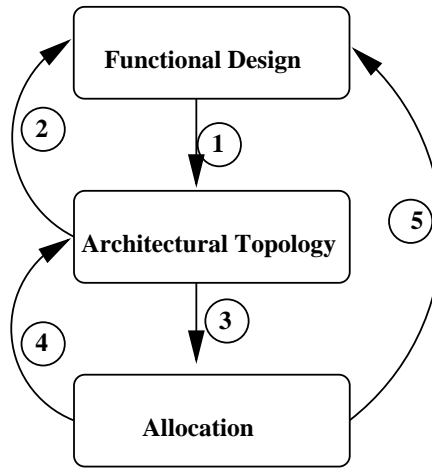


Figure 5.14: Interactions between Topology Selection & Design

on analyses of previous iterations of the X-Topmeter tool. Any control task or service that does not change when the functional design is changed may be re-used in the next architectural topology search. In the most restrictive case the designer may mandate a particular architectural topology for part of the system. In this way the architectural topology for the system emerges alongside system design.

Interactions between the architectural topology and other design issues relate mainly to the effect of introducing a particular architectural topology on the attributes of a system. The maintainability of a functional element may only be predicted once an assignment to resources has been made. Thus, for instance, maintainability may be improved by the decision to employ a redundant, rather than diverse, architectural topology. The maintainability of a system can emerge alongside the functional design and topology selection process.

Changes in other aspects of a system can have an impact on the topology selection process. For instance, a decision to change the computational model employed, or to change the design because of safety features of the proposed system, will have an effect on the quality of different topologies.

The effect of design decomposition can also be seen. As design progresses more effort is required to encapsulate the design as a set of tasks and Services. Similarly, more complex task reliability models will be required to produce a prediction of the reliability of a proposed architectural topology.

The architectural topology problem addresses dependability and resource aspects of a design. The set of sensors and actuators to be employed in the hardware platform can be predicted directly from the architectural topology selected for the proposed system design. However, the timing attributes of processing tasks remain as WCET budgets and WCRT requirements. At some point in the design process these processing tasks

are decomposed into a set of software processes that can be allocated to individual processors.

Effort is concentrated on the allocation of processing sub-tasks (messages) to particular processing (bus) units, link 3 in Figure 5.14, for a given design and architectural topology. The architectural topology determines both the set of sub-tasks and the resources they are to be allocated to. The processing platform, and timing attributes, of the proposed architectural topology and system design, can be predicted via an allocation search.

Results from an allocation search can have an impact on the acceptability of the proposed architectural topology, feedback link 4. For instance, suppose a particular processor is employed to provide fault-tolerance for one Service. The allocation search indicates the processing unit required for this dependability assignment is under-utilised. The designer may wish to place restrictions on the architectural topology to limit the effect of such resource allocations on the size of the required hardware platform. Similarly, suppose the selected architectural topology employs a large number of sub-tasks. The allocation search may indicate that the required Service timing properties can only be met if an unacceptably large number of processors are employed.

The architectural topology and allocation results are used to aid production of a new functional design, feedback links 2 and 5. They may also be used to determine a revised set of resources and /or architectural topologies³. This new design is then subjected to an architectural topology search. The resulting architectural topology is subjected to an allocation search. This process continues until a stable topology, allocation, hardware platform and system design emerge.

One aspect of the topology selection process needs to be emphasised. The emergence of a functional design and a topology during the design process is not purely mechanistic. The approach adopted in this thesis employs automated synthesis and evaluation steps. However, an analysis step is also mandated in the ASE process. It is during the analysis step that the designer has a considerable influence on the topology selection process.

The analysis stage involves the designer looking at the topologies produced by the tools. For instance, the designer should investigate the results of the two tools to ensure that no discrepancies, in the predicted size of the platform for instance, are evident and that the tools are converging towards an acceptable solution. The designer may decide to change the search spaces or the evaluation functions to investigate different design scenarios. Changes to the search space may include changes to the library of resources and the set of admissible topologies. Changes to the evaluation functions may include changes to the weighting factors, coverage factors or way in which the dependability of a topology is evaluated. The designer may also decide that system requirements,

³The new functional design may also require changes to existing reliability models.

such as WCRT targets, cannot be met using a topology of acceptable cost and that therefore some renegotiation of requirements is necessary.

In conclusion the convergence of the topology selection process to an acceptable topology is a function of the design process, the effectiveness of the architectural topology and allocation search engines, and the skills of the designer in guiding the emergence process. The primary agent for producing a good topology remains the designer.

5.5 Evaluation of the Approach

One aim of this thesis was to produce a set of quantitative measures that allow a topology to be selected with respect to its cost, reliability and timing characteristics. A second aim was to produce tools to automatically search a set of possible architectures to suggest a topology to the designer. The third, and most important aim, was to provide flexible support to the system designers so that the topology of a system could emerge during the design life-cycle. The X-Topmeter / X-Alloc approach has been produced to fulfil these aims.

Results of the illustrative examples and the case study provide much of the supporting evidence for the validity of the X-Topmeter / X-Alloc approach. Reference will be made to these studies in this exposition. The following attributes of the X-Topmeter / X-Alloc approach are evaluated to show that these three aims have been met:

- Off-the-shelf tools and coding of examples
- Evaluation functions
- Interactions
- Speed and scalability of the search tools
- Flexibility and extendibility

Usability issues are considered as part of the comparative analysis undertaken in Chapter 6.

5.5.1 Off-the-shelf Tools

The X-Topmeter / X-Alloc approach employs four off-the shelf tools; GAMeter, SAMson, SHARPE and a schedulability analysis tool. No formal evaluation of these tools has been undertaken. An informal evaluation takes the form of an analysis of their use on illustrative examples and the case study.

The GAMeter and SAMson tools have performed as expected. For instance, the CABV1 architectural topology search was a complex search involving computationally expensive reliability calculations. The tool was able to navigate this space effectively by discarding the more complex, and hence time consuming to evaluate, architectural topologies in favour of less complex topologies. In fact, the first 200 evaluations performed on CABV1 took an average of 17 mins 40 seconds per evaluation, whilst the last 1000 evaluations averaged 30 seconds per evaluation. The steps taken by the algorithm show the expected characteristics of an initial rapid fall in solution value, then a more steady improvement and finally very slow progress, once a feasible local optima has been found.

The X-Topmeter tool for CABV1 required 5500 lines of problem specific code to implement. Similarly, the X-Alloc tool for CABV1 required 3000 lines of problem specific code. GAMeter is coded in 16000 lines and SAMson in 15400 lines. Thus, production of X-Topmeter and X-Alloc for topology problems is not trivial. It should be remembered that a proportion of this code can be reused each time the tool is employed during the design process and between projects. This is particularly true of the reliability models and schedulability test employed.

The GAMeter and SAMson tools are flexible enough to provide a variety of alternative search profiles. The full capabilities of these search engines have not been exercised by the current set of examples. This gives some assurance for the ability of these tools to cope with larger or more complex examples.

The SHARPE reliability evaluation tool is the product of many years academic and industrial expertise. The tool has been presented with a variety of reliability models to solve during the production of the illustrative examples introduced in this thesis. A check by hand of a number of simple models was undertaken and the results confirmed the SHARPE output. A plausibility check for more complex models was also undertaken. SHARPE is able to produce measures, such as availability and performability, that have not yet been employed in X-Topmeter. This allows extensions to the X-Topmeter tool to be made in the future.

The hierarchical modelling and evaluation capabilities of SHARPE have proved particularly important as they facilitate the sub-task, task, network and Service structure presented in this thesis. The ability to produce a reliability model library and automatically generate reliability models that SHARPE can solve has also proved to be an effective part of the architectural topology selection process. This can be shown by consideration of CABV1. Different reliability models are employed for different sensing tasks in the system. More than one Service model is employed with, for instance, Service 1 being able to employ the results of the Service 0 sensing task.

The schedulability tool is a research tool. It was also the tool that was ‘changed’ the

most to conform with the allocation problem formulation. The results produced have been checked to ensure that offsets larger than period, or WCRT, are not produced. Transactions were also checked to ensure that the WCRT of messages were not greater than the offset of subsequent sub-tasks. A number of unusual features were found if no feasible schedule could be found. This is primarily due to the truncation of the convergence process for such allocations. The X-Alloc evaluation function was amended to take these features into account. Results for the CABV1 case study show that schedulability of a realistic sub-task DAG, resulting from a particular architectural topology, can be analysed using this schedulability tool.

Results produced in this thesis indicate that existing tools can be incorporated within the framework. There appears to be no reason why other specialist tools cannot be used to evaluate each proposed topology for characteristics such as maintainability.

5.5.2 Evaluation Functions

Two evaluation functions are employed in this thesis, one for the architectural topology problem and one for the allocation problem. They take into account both system goals and constraints on the quality of a topology. They were produced from perusal of the literature and domain knowledge from a number of industrialists. Formal analysis using the techniques presented by Prasad [131] was not undertaken, due to the lack of application specific knowledge. However, the results produced are encouraging and provide information on the relative merits of different topologies to designers.

The evaluation functions have been used in a variety of illustrative examples and the case study. The X-Topmeter evaluation function has also been employed to evaluate a sequence of solutions of known quality, see Table 3.13. These solutions differ by a single value of the gene string in the accompanying search engine. A coverage factor was added as a result of this analysis. A similar exercise was undertaken for X-Alloc. Results show that the evaluation function is well behaved in that a better solution produces a smaller value. This approach was used on simplistic examples, there is no reason however, to believe that this should not also be the case for more complex examples.

The example generation tool introduced in Section 3.3.1, was used to produce a number of example architectural topologies. These examples were then subjected to the reliability algorithms employed in X-Topmeter and the results analysed. The reliability values produced were consistent with the expected results, given the reliability models employed. The reliability models used were chosen to be as simple as possible, while still providing the ability to distinguish between different proposed topologies. The suitability of the models employed is an issue not addressed explicitly in this thesis.

The evaluation functions employed in this thesis are open to change, both within and

between projects. A single universal evaluation function for every iteration of the approach and for each project is not possible. In particular the weighting factors will evolve as a design progresses. However, the quantitative values are employed to aid designers make qualitative decisions and therefore some inaccuracy in the evaluation values can be tolerated. For instance, although the allocation employed in CABV1 may not be optimal it was able to show the deficiencies in the design of this variant. Furthermore, the evaluation function values provided information that was used to determine a new design for the processing tasks in the CAB system.

Designers will need to undertake a more rigorous modelling and evaluation exercise once a topology has emerged to assure themselves that reliability targets have been met. However, results produced by the evaluation functions employed in the illustrative examples and case study are encouraging. They indicate that using quantitative evaluation techniques to select an appropriate topology is feasible.

5.5.3 Interactions

The X-Topmeter / X-Alloc approach can accommodate analysis of interactions, and trade-offs, at a number of levels. X-Topmeter makes explicit trade-offs between reliability, platform size and cost. The value of these trade-offs is shown in illustrative example 3 for the architectural topology problem. In this example a number of scenarios are generated, and evaluated, during one iteration of the architectural topology problem. The results of these scenarios feed back into the topology selection process and provide information to the designers of the system. X-Alloc makes trade-offs between WCRTs, platform size and cost. Together these tools produce a topology. The effect of employing this topology on other design factors can then be analysed. The set of acceptable trade-offs by the tools can be changed in the light of this analysis.

The architectural topology resulting from X-Topmeter has a significant effect on the size and cost of the processing platform predicted by the X-Alloc tool. It also has an effect on the complexity, and difficulty of the allocation problem presented to X-Alloc. The designers may use results from iterations of the X-Alloc tool to place restrictions on the set of admissible architectural topologies. This *feedback loop* is a valuable addition to the data available to the designers of a system. The interaction between design, and topology selection is particularly evident in the decision to abandon CABV1 in favour of CABV2.

No explicit analysis of the trade-off sequences made by each tool has been undertaken. However, analysis of the topologies resulting from the illustrative examples and case study shows that acceptable topologies are produced. The author found the results of one iteration of the tools to be extremely helpful in producing the next refinement of the search. This indicates that the results of the trade-off sequences within a sub-

problem are transparent and can be used to aid the emergence process.

The link between an iteration of the allocation problem and the architectural topology it employed is clear. For instance, under-utilisation of a processor could often be linked directly to an architectural topology resource allocation decision. Similarly, the size of the communication network could be linked back to a network topology decision. This allows attention to be focused on particular elements of the proposed architectural topology.

5.5.4 Speed and Scalability of the Algorithms

Automated traversal of the topology design space is provided by the X-Topmeter and X-Alloc tools. The speed and scalability of the algorithms employed forms part of the evaluation of any tool based approach. X-Topmeter and X-Alloc are prototype tools employed to show that the concepts presented are viable. The code is in no way optimised for speed, neither with respect to the efficiency of code or the parameter settings for individual experiments. The experiments were undertaken on relatively slow machines by modern standards, namely a Sparc classic and a 133Mhz Pentium. However, a number of features are apparent. For instance, the allocation problem is relatively quick to resolve in comparison with the architectural topology problem.

Consider first the speed of an X-Topmeter experiment. The most time consuming experiments were those for CABV1 which took up to 6 hours to execute. Other less complex illustrative examples took considerably shorter time; of the order of 3 hours. The deciding factor on the speed of solution appears to be the time to predict the reliability of services. Some of the more complex topologies employing multiple copies per task have taken up to 25 minutes to evaluate the reliability of a single service. This is particularly the case for services 1 and 5 of CABV1, which employ a kofn model within each copy of the sensing task employed. Furthermore, service 1 employs results from another sensing task, the reliability of which is factored into the reliability calculations for the service.

To place the execution time of X-Topmeter into context a brute-force search was undertaken for a very simple example. The brute-force search took three weeks to run on a sun Sparc classic workstation. The best evaluation function value was 4860. The X-Topmeter tool found a solution within 0.062% of the optimal solution in 0.03% of the cpu time.

X-Topmeter is never going to be a *fast* tool due to the nature of the search space it is traversing. However, the assumptions used and the level of abstraction at which this approach sets a topology appear to give a good balance of accuracy versus speed. An experiment that can be completed overnight would appear to be practicable as a design aid. Activities can be undertaken to increase the speed of an X-Topmeter experiment.

In control system design a single Fourier transform evaluation can take many hours to evaluate [127, 167]. A run time of many days for a GA search engine to complete an experiment is not uncommon. This has led to a great deal of work to ensure convergence to reasonable solutions in a small number of evaluations. It is not unusual to see runs of 400 to 500 evaluations. Elements of this work could be incorporated into the X-Topmeter search process when the evaluation functions becomes very time consuming.

Perusal of the statistics files produced by the X-Topmeter tool for the illustrative examples and case study experiments show that solutions within 10% of the best solution were found after only 36% of the final number of evaluations made. This information can be employed to truncate the experiments to save time. Finally, obvious solutions such as parallelisation and faster processors can be employed. Stand-alone PCs running linux now run at more than twice the speed of the machines employed here.

The X-Alloc examples have shown that for smaller problems a rapid convergence to a solution occurs. In CABV1 a much more difficult problem was presented to the tool. Nevertheless, convergence occurred in 40 minutes. Scalability problems are reduced by the use of restrictions on the admissible allocations implied by the architectural topology employed. The variable-size allocation approach helps to overcome some of the scalability problems experienced with other SA approaches by allowing the tool to escape from local minima by adding extra hardware units. The addition of TS restrictions should also reduce scalability problems. Scalability issues remain a fundamental feature of X-Alloc due to the NP-hard nature of the allocation optimisation problem.

5.5.5 Flexibility and Extensibility

Flexibility is one of the main criteria for the usefulness of a tool to aid the design process. This flexibility must be allied with the ability to re-use as much information and designer effort as possible. Of course, good topologies must also be produced.

X-Topmeter and X-Alloc were specifically designed to allow a great deal of flexibility and re-use. In particular the use of generic architectures, data-libraries, generic reliability models and standard fault-tolerance techniques minimises the effort required by the designer, while allowing a wide variety of systems to be evaluated.

The Illustrative examples and case study show that the X-Topmeter / X-Alloc approach is extremely flexible. For instance, task level models can be adjusted to take account of sub-models, such as the need for 4-out-of-4 wheel rotation sensors for ABS. The service level models can similarly be adjusted to model the use of sensing results from multiple sensing tasks. The set of resources employed can also be changed. At the allocation level the schedulability analysis employed can be changed and the set of

possible allocations can be tailored to the particular architectural topology.

Flexibility is built in by the use of evaluation functions that can employ both *goals*, such as platform size and cost, and side constraints, such as reliability. The evaluation functions are only restricted to those functions which are calculable, provide an appropriate level of granularity and are well behaved. A well behaved function increases in value as the quality of a solution falls.

The search engines employed have also be chosen with flexibility in mind. Guided search techniques, such as GAs, are very good at traversing poorly defined search spaces. Search techniques, such as SA, are able to produce optimal results. Both of these approaches can be parameterised to improve the search undertaken for a particular example. A wide variety of alternative select, creation and merge functions are available in both the search engines employed in this thesis.

Flexibility, in itself, is not adequate for an approach which aims to provide a framework to aid a designer resolve the topology selection problem. The approach must be extendible to consider different generic architectures and accompanying dependability and timing evaluation techniques. To show the extendibility of the X-Topmeter / X-Alloc approach a third version of the CAB case study is presented in Section 5.5.6.

5.5.6 CAB Version 3: Independent Architecture

CAB variants 1 and 2 employ the GUARDS generic architecture and priority based pre-emptive scheduling of processing elements. To show that the X-Topmeter / X-alloc approach can be extended to consider systems that do not employ this framework CAB Version 3 is presented in this Section. Considerable effort would be required to implement this Version of CAB in X-Topmeter and X-Alloc as the schedulability analysis, and some of the reliability analyses, employed in Versions 1 and 2 will no longer be relevant. This Section therefore takes the form of a discussion of the implementation required for Version 3.

In CABV3, see Figure 5.15, a processing hardware ‘node’ is placed at each wheel. Braking demands arrive from the central computing sub-platform and the local computing element employs appropriate braking activities. Each distributed wheel-braking action may be assumed to be independent of other wheel-braking actions. This is clearly hazardous and may lead to asymmetric braking of the vehicle. To introduce ‘dependence’ between the braking activities at each wheel cross-communication between computing hardware units on the same axle is required.

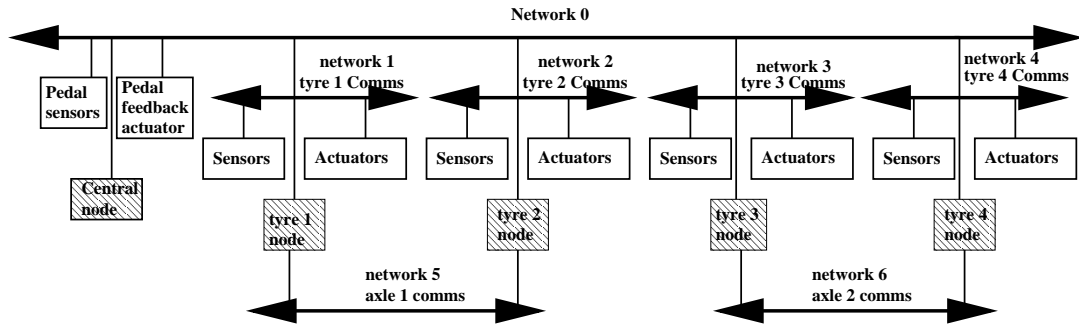


Figure 5.15: CABV3: System Description

Each ‘tyre node’ implements a BASIC and an ABS algorithm, which determines the braking-pressure to employ. Enhanced braking is implemented as a separate set of services. Load balancing is undertaken by each wheel node based on the data from load sensors on each axle. The logical architecture for this system is given in Figure 5.16. The central computing node captures the pedal pressure sensor values, pre-processes them and passes a value to the tyre modules. It also provides the driver feedback function. The tyre nodes deal with all other sensor values, processing and actuating. Seven communication networks are employed in CABV3. Network 0 links the central node to the tyre nodes. Networks 1 to 4 facilitate inter-node communication. Finally, networks 5 and 6 allow cross axle communication.

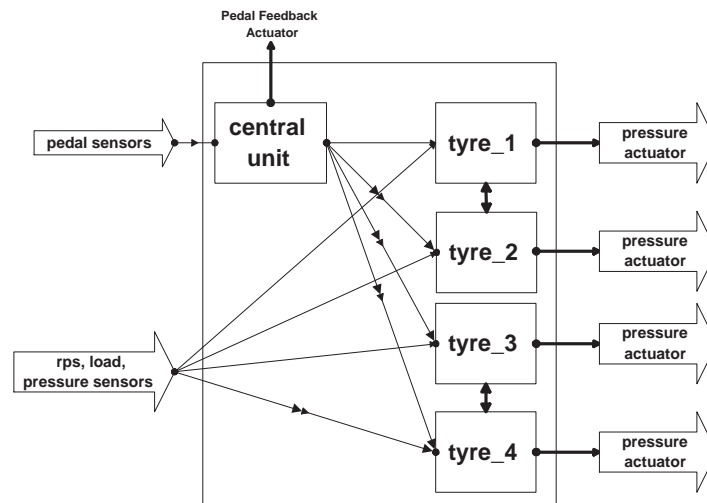


Figure 5.16: CABV3: Logical Architecture

Nine Services are to be implemented to provide the required braking activity on this multi-node platform:

Service 0: braking front left	Service 1: braking front right
Service 2: braking rear left	Service 3: braking rear right
Service 4: cross check across front axle	Service 5: cross check across rear axle
Service 6: LOAD (F) - front axle load balancing	Service 7: LOAD(R) - rear axle load balancing
Service 8: DRIVER - Feedback to driver	

In this Version an alternative generic architecture, the Time-Triggered Architecture (TTA) developed at the Technical University of Vienna [145], is to be employed. Particularly affected by this change are the processing task and network topologies.

In a TTA all system activities are initiated by the progression of a globally synchronised timebase. The key advantages are claimed to be composability and the transparent implementation of fault-tolerance. An autonomous communication controller provides de-coupling between the host sub-system and the communication sub-system. Together these sub-systems form the platform. Communication between electronic units is performed using the Time-Triggered Protocol (TTP/C). The communication sub-system decides autonomously, according to a static schedule, when to transmit a message and whether a received message is relevant for a particular electronic unit.

An electronic unit has the general structure shown in Figure 5.17. It consists of a host sub-system, a communication sub-system, and a Controlled Object Interface (COI) to the sensors and actuators. The host sub-system executes the application software. The communication sub-system employs the TTA communication controller and executes the TTP/C protocol.

During operation, each communication controller synchronises its local clock to generate a common time base of known precision. Whenever the common time reaches an instant that is contained in the message descriptor list (MEDL) the actions specified in the MEDL are carried out. Clock synchronisation is performed by the fault-tolerant average algorithm [87].

One or more electronic units can be combined to form a fault-tolerant unit (FTU) to provide a specified control service. An FTU consists of two or more electronic units, see Figure 5.18. The FTU layer is responsible for the management of the replicated electronic units within an FTU, i.e. it performs redundancy management. An FTU can be employed as an architectural component. For example,

- A single unit (Figure 5.18(c)) can be configured if redundancy is maintained at the system level (e.g. four wheel nodes in the braking system of a car).
- Two units (Figure 5.18(a)) can be grouped into an FTU that will provide the specified service as long as one of the two units is operating. Fail silence in the value domain has to be ensured by the host.

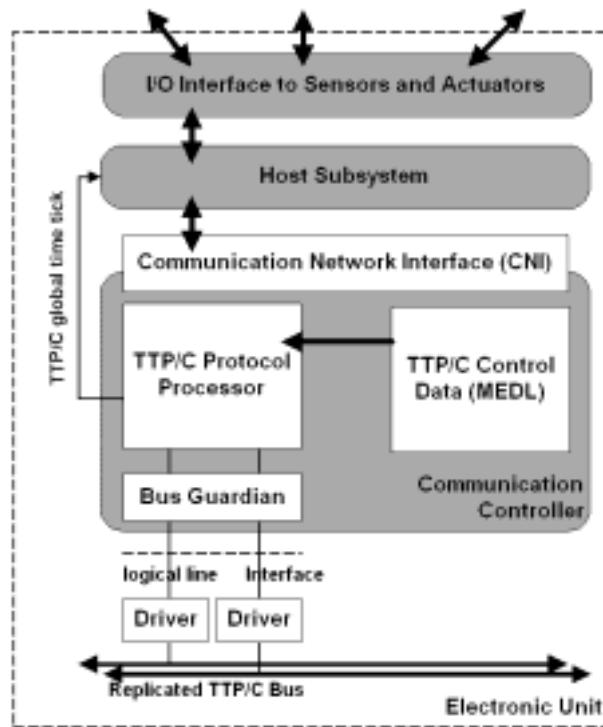


Figure 5.17: Structure of an Electronic Unit

- Three units (Figure 5.18(b)) can be grouped into a Triple Modular Redundancy FTU. This FTU will tolerate a single failure of any one of its units.
- Four units can be configured if the system has to be available in a TMR mode, even if one of the units is in maintenance (Figure 5.18(d)).

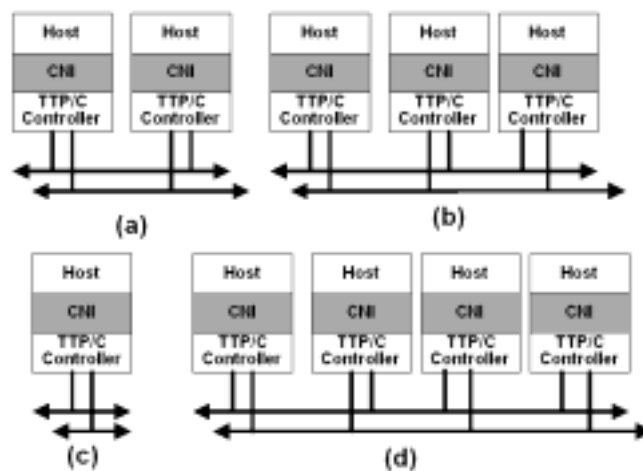


Figure 5.18: FTU Configurations: (a)2 units (b)TMR units (c)Single unit (d) 4 units

It is possible to form FTUs of software units executing on the hosts of different electronic units. Thus, allocations of multiple software units to the same processor can be undertaken. Furthermore, different FTUs may employ the same or different buses.

Possible Implementation of X-Topmeter

This version of CAB has not been implemented in X-Topmeter or X-Alloc, mainly due to the coding effort required. However, the required extensions to these tools to select a topology for CABV3 can be envisaged. A number of features relating to the extendibility of the topology selection approach are introduced.

In Chapter 2 a generic guided search algorithm was introduced. This required: a representation of the problem and solution, a select function, a create function, and a merge function. One version of these elements has been used to produce a GA to address the architectural topology problem for a GUARDS architecture. This GA can be adapted for the TTA employed in CABV3.

The logical architecture for CABV3 can be represented as a set of Services. The sensors and actuators employed can be modelled as tasks. The differences come in the representation of a processing task and the interface between the three tasks in a service.

Each electronic unit employed in a TTA platform employs a COI and can therefore be connected directly to the appropriate sensors and actuators. Two electronic buses are used to transfer data between electronic units in an FTU. Different FTUs may employ the same physical buses. The number of networks in the system is a function of the processing tasks that share buses. In CABV3 seven networks are required. Each network may employ a gene structure consisting of two integers denoting the resource employed for the two buses.

An FTU forms an architectural topology for a processing task. Integral to an FTU is a set of redundancy management protocols, including a set of decision mechanisms. Only hot sparing is employed in a standard FTU. Therefore the topologies for each processing task can be expressed using three integers:

1. an integer with value 0 to 3 to indicate the number of processing resource alternate 1 employed,
2. an integer with value 0 to 3 to indicate the number of processing resource alternate 2 employed, and
3. an integer with value 0 to 3 to indicate the number of processing resource alternate 3 employed.

Thus in X-TopmeterV2.3 27 integers would be required to represent the set of task topologies and 14 integers to represent the communication networks.

Provision can be made to employ shadow units. These are units that may be employed by a number of different FTUs. In other words they are shared cold spares. If shadow units are employed then the representation of a processing task needs to be extended. Suppose up to 3 shadows could be employed per FTU. Each FTU gene would require 3 extra integers, each of which would take on a value between 0 and the number of admissible shadow resources. A zero indicates the FTU cannot employ that shadow unit.

The X-TopmeterV2.1 select and create functions may be employed. The merge function is based on an evaluation function, which must be able to distinguish between good, and bad, TTA architectural topologies. This function uses measures of cost, size and reliability. A cost measure can be easily constructed to take into account the implementation of each electronic unit. A size measure that takes into account the requirement for five processing nodes in CABV3 can also be constructed. The minimum number of buses used in a TTA system is dependent on how the seven logical networks are implemented. Existing reliability models need to be changed, or extended, to ensure that an accurate measure of the reliability of each service is produced.

The reliability model for an electronic unit should take note of the hardware resources being employed and the communication interfaces. Since processing and bus hardware form an integral part of an electronic unit a single detailed reliability model can be produced. This will probably be state-based. The variable element of the reliability of a unit is provided by the different type and number of processing and bus resources that may be employed.

The processing task (FTU) level reliability model must take account of the possibility of common cause electronic unit failures on the reliability of the task. This can be done simplistically using a set of coverage values for the different configurations of shared electronic unit hardware.

If FTUs use shadow nodes then there is a dependency between failures in the system. That is whether an FTU has access to a shadow is dependent on the set of failures in the system. This will require a Markov model similar to that used for shared repair engines [152]. Hidden Markov models, such as that employed by HARP [46], may be used.

The existing generic Service reliability models must be altered to reflect the configuration of networks employed in each service. Thus, the main change required to implement X-TopmeterV2.3 is the running of a reliability model generator to produce a set of electronic unit, FTU and Service reliability models for the proposed TTA topologies, see Section 2.4.

Possible Implementation of X-Alloc

Once an architectural topology has been selected an allocation process can be undertaken. A TTA platform employs a single global static schedule [53]. This schedule is based on a tree-search of the set of possible schedules for a given allocation of sub-tasks to processors and messages to buses. The tree is based on the DAG of precedence constrained sub-tasks for the allocation.

The trade-off for this scheduling approach is a reduction in the flexibility of the system. Sub-tasks have allotted slots on the processor and can only run at this time. Thus the maximum utilisation of a processor, or bus, using this scheme is reduced relative to the priority based scheme employed in CABV1 and CABV2. This implies a larger platform may be required. Since TTA does not employ a priority based approach, only a single integer is required to indicate the allocation of each sub-task. A second integer indicates the allocation of each message.

The select and create functions for X-AllocV2.3 may be identical to those employed for X-AllocV2.1. However, two attributes of the merge function need to be investigated. First, the probability of undertaking a move of a particular type. Second, the fragmentation of the platform due to the use of nodes needs to be taken into account.

The use of computing nodes places tight restrictions on where a sub-task and message may be placed, which can be taken into account by adjusting the setP and setB parameters for each sub-task and message. This may lead to a problem with *deception*. To overcome this elements of TS may be employed. In this circumstance illegal moves may only be proposed if the move is not currently tabu. The probability of an illegal move being accepted is determined by the X-Alloc cooling schedule.

The resource constraints employed in X-AllocV2.1 can be used in X-AllocV2.3. The same set of resource constraints can be employed. The cost and size elements of the evaluation function will change, especially if TS methodology is employed. TTA employs a different scheduling approach to that used in the current X-Alloc. Therefore, a different schedulability algorithm is required. The scheduling algorithm introduced by Fohler & Koza [53] employs a schedulability test and generates a *measure* of the quality of the resulting schedule. This measure should be incorporated into the X-AllocV2.3 evaluation function to guide the search process.

Summary

The topology for CABV3 is somewhat different to that of the other two CAB versions. A distributed rather than centralised platform is employed. Both X-Alloc and X-Topmeter can be extended to consider a set of possible topologies for this new version. The discussions for this variant have shown that the TTA and TTP/C communication

protocol are compatible with the approach taken in this thesis to resolving the topology selection problem. The changes required to provide guidance for selection of an appropriate topology centre on reliability and timing analysis.

The TTA is a very prescriptive generic architecture in comparison to GUARDS. Thus, the reliability models employed can reflect this extra information. The trade-off is increased work on producing the models, potentially extra layers to the model to incorporate common cause effects, and increased computation time. The plus side is an increase in the ability of the evaluation function to distinguish between alternatives and a reduction in the size of the architectural topology search space.

Timing analysis requires a schedulability analysis. Fortunately, TTA employs a static schedule which can be analysed for the schedulability of its elements off-line. Unfortunately, the approach used is computationally expensive. A number of heuristics may help reduce the time taken by the scheduling algorithm. This is important as the scheduling algorithm is run many times during an X-Alloc experiment.

5.6 Summary

In this Chapter a case study has been presented to bring together the elements presented in Chapters 3 and 4. This case study shows that the topology search space and evaluation functions can be customised for particular examples. It also shows that detailed computing task designs can be incorporated into the architectural topology selection process. Finally, it shows that the allocation problem can be addressed relatively early in the design process.

The evaluation of the approach undertaken in this Chapter has addressed three aims of this thesis. First, to show that quantitative measures of the quality of a topology can be produced. Second, to show that automatic search techniques can be employed to traverse the topology design space. Finally, to show that the approach presented in this thesis can aid the designer resolve the topology problem for safety-critical real-time systems that employ the GUARDS generic architecture.

Overall, the flexibility, extendibility, evaluation function and speed factors evaluated in this Chapter indicate that the X-Topmeter / X-Alloc approach can provide support to the system designers so that the topology of a GUARDS system can emerge during the design life-cycle.

It is also clear from the evaluation and results of the case study that the approach is flexible, extendible and fast enough to provide support for the development of SC-RT systems that employ different generic architectures, such as TTA or shared memory systems.

Chapter 6

Conclusions

This thesis is concluded with:

- a summary of achievements
- a comparison of the work with related research
- a set of proposals for future work

6.1 Summary of Thesis

Selecting a topology is a fundamental issue for the designers, procurers and maintainers of a safety-critical real-time control system. A topology has hitherto been set early in the design process. Considerable effort is then expended on three aspects of the system-topology interface:

- Provision of an appropriate level of control functionality within the topology framework
- Allocation of software units to hardware units on a given platform
- Evaluation of the dependability and timing attributes of a full system design

Any problems arising from this effort may cause costly redesign and an increase in system complexity.

This thesis has introduced an approach that allows a topology to co-evolve with the design of a control system. The approach indicates a topology, if one exists, that can meet timing and dependability, as well as functional, requirements. The designer then decides on the appropriateness of the proposed design and accompanying topology.

The approach utilises existing quantitative evaluation techniques within an Analysis - Synthesis - Evaluation framework.

The approach is first employed once a generic architecture and initial high level design (logical architecture) have been produced. Procurement information is used to determine a library of admissible hardware and software resources. Design information is used to determine a set of control services that must be provided by the system and the dependability requirements on these Services. A set of admissible architectural components are selected by the designer. Finally, a Reliability Model Generator is used to determine a set of reliability models that will be used to predict the dependability of proposed architectural topologies. This data is extensive, but most of it is employed within existing design processes or can be obtained from domain knowledge.

The resulting design space is traversed using a tool, X-Topmeter, that employs an appropriate evaluation function. This tool was developed as part of this thesis and employs a GA, selected as the appropriate version of the generic search algorithm for this problem, to choose an architectural topology. An analysis is then made of the acceptability of the proposed topology and, by inference, the proposed design. This analysis may include a more detailed reliability analysis of the system. In later stages of the process analysis of the reliability of the system will need to be reassessed in the light of the proposed allocation of processing resources.

A new functional design is proposed and the topology design space analysed in the light of the new design and previous iterations of the approach. This may lead to a change in the set of admissible resources, architectural components, control services or requirements on these control services. X-Topmeter is then re-applied. The ability to analyse, and change, the characteristics of both the design space and what constitutes a valid solution to the topology problem within a single framework, is a new and novel feature of this approach. This Analysis-Synthesis-Evaluation approach is repeated throughout the design process.

Once a set of processing sub-tasks is produced for a particular design the timing attributes of the control system may be predicted. An evaluation of the timing properties of a topology can only be undertaken once an allocation of the processing sub-tasks to particular processing units has been postulated. A second tool, X-Alloc, is employed to determine this allocation. X-Alloc employs a Simulated Annealing algorithm, which was selected as the appropriate version of the generic search technique for this problem. Since an assignment of software resources to hardware resources is an integral part of X-Topmeter the allocation problem can be addressed relatively early in the design process.

A postulated solution to the architectural topology problem and the functional units given in the system DAG are used to determine an allocation search space. Each

allocation is evaluated with respect to its timing and resource usage attributes. X-Alloc determines an allocation of sub-tasks to processing units and an allocation of the accompanying messages to communication hardware. It also determines the processing platform implied by this allocation.

This allocation process is undertaken each time a new architectural topology, or set of functional sub-tasks, is postulated by the designer. Analysis of previous iterations and the system design allow the allocation search space, and hence the processing platform, to evolve as design progresses.

The framework, and topology selection techniques, developed in this thesis provide a means of co-evolving the dependability and timing characteristics of a given control system with the functional design. It does so using quantitative evaluations of the timing and reliability characteristics of each proposed topology.

The efficacy and flexibility of this approach has been demonstrated using a set of illustrative examples and a case study. The case study showed a snapshot of how this approach could be employed on a real-world problem. It also showed the current variety of generic architectures, scheduling regimes and fault-tolerance approaches that can be accommodated. The framework remains the same but the set of solutions and evaluation techniques involved are changed.

One side-effect of building the data libraries and models required for this approach has been a greater understanding of the system being developed and the impact of design decisions on the non-functional attributes of a design. It also allows issues of re-use, both in terms of resources and analysis models, to be explicitly addressed. Provision for re-use is essential if this approach is to be employed in industrial practice.

In short, this thesis has shown that quantitative measures can be used to guide the topology selection process for a hard real-time safety-critical system. The prototype tools show that automated help can be made available to the designer throughout the design process. The work undertaken within this thesis is transferrable, within limits (Section 6.4), in that it can be adjusted to the needs of developers of particular projects.

6.2 Comparison with Related Work

Several limited attempts have been made in recent years to develop approaches to design that incorporate reliability and timing attributes of a system. In this Section the approach presented in this thesis is compared with research undertaken in five areas:

1. Topology selection

2. Architectural topologies
3. Allocation
4. Interactions with other design issues
5. Industrial practice

6.2.1 Topology Selection

Research on topology selection as a design activity is sparse. The author is aware of three relevant studies by Thompson [159], Draber [42] and at Siemens [59].

Thompson presents a 13 stage design cycle for distributed fault-tolerant systems based on industrial experience, see Section 3.1. Thompson’s approach focuses almost exclusively on hardware issues and attempts to determine a set of “show-stoppers”, such as an inability to meet reliability targets. Show-stoppers investigated include safety and weight issues. Thompson addresses safety in a very simplistic manner by mandating a replicated bus-based architecture.

Safety show-stoppers can be addressed in the X-Topmeter / X-Alloc approach by altering the system design, or the set of admissible topologies or the evaluation functions. Similarly, weight factors may easily be added to the evaluation functions either as a numerical goal or a side constraint. RBDs are employed by Thompson to “perform a quick combinatorial reliability comparison of redundant configurations” and a simple fixed cyclic schedule is employed to schedule control functions.

The X-Topmeter / X-Alloc approach is compatible with, and subsumes, most of the aspects of the approach presented by Thompson. The use of simulation techniques to evaluate maintainability characteristics of a proposed topology may be a useful addition to the X-Topmeter / X-Alloc approach. X-Topmeter / X-Alloc improve on Thompson’s work primarily in the set of alternative topologies that may be considered, the flexibility of the approach, and the use of evaluation functions to guide the topology selection process.

Draber’s approach is also industrially based. Considerable effort is expended in reducing the number of admissible topologies for a system to a level where an exhaustive search can be employed. Draber’s approach is reliant on a detailed design, extensive knowledge of the consequences of employing different topologies and the availability of FMEA data. The set of restrictions employed by Draber appear to be realistic and may be of help in restricting the topology design space late in the design process.

The X-Topmeter / X-Alloc approach could be employed in a manner similar to Draber. For instance, FMEA data for the resources in the resource library may be used to rule-out particular topologies. The topology evaluation criteria employed by the two

approaches are similar. X-Topmeter / X-Alloc can be given appropriate parameter values to act as simple neighbourhood search techniques or even exhaustive search engines. Furthermore, TS hybridisation may employ variants of the rules used by Draber to rule out particular alternatives. However, X-Topmeter / X-Alloc is a more flexible approach. It may be employed from an earlier stage in the design process to give design guidance. It also allows variants that appear to be unacceptable at one level of design decomposition to be re-evaluated at later stages in the process.

The work at Siemens is also industrially based. In stage one a set of evaluation criteria are produced based on customer requirements, project requirements and quality attributes. In stage two system implementations are proposed and evaluated using the criteria produced in stage one.

The X-Topmeter / X-Alloc approach assumes that an evaluation function for a topology can be produced. The work at Siemens shows one way in which current industrial practice may facilitate the production of this function. Furthermore, the function can be upgraded at different levels in the design process. X-Topmeter / X-Alloc specifically addresses stage 2 and provides a considerable improvement on the existing simplistic approach used by Siemens. It is odd that so much effort should be taken to produce an appropriate evaluation function, but relatively little use made of the result during the design process.

6.2.2 Architectural Topologies

A number of authors have employed GAs to address problems similar in form to the architectural topology problem, that is to select the level of redundancy to employ in a fault-tolerant system. For instance, Painton & Campbell [125] employ a GA to investigate how a system can be improved during its lifetime by adding to, or changing, hardware components in the system. Coit & Smith [31] also produce a GA to address the redundancy allocation problem for a series-parallel system.

The X-Topmeter / X-Alloc approach subsumes these approaches and extends the problem into the design arena. It is able to employ a multiplicity of system and reliability models, including combinatorial (RBD) and state based (Markov) approaches. Furthermore, SHARPE can evaluate these models to produce a wide variety of performance and dependability measures. For instance, although the work in this thesis has concentrated on reliability measures the models could be easily adjusted to model system availability and SHARPE used to produce appropriate availability measures.

A significant improvement of the X-Topmeter / X-Alloc approach is the explicit use of a reliability model generator as part of the framework for the approach. This allows models for proposed architectural topologies to be produced as required, although not during an X-Topmeter experiment. Joint software - hardware reliability models may

also be employed.

6.2.3 Allocation

The allocation problem is well known and has been addressed by a wide variety of authors. In particular the work of Tindell [163] and Alternberd [3] is representative. Tindell employs an SA approach, while Alternberd employs a hybrid approach. Both authors address the problem late in the design life-cycle for given hardware platforms. Good results are produced for a variety of allocation problems. Effectively the quality of an approach to solving the allocation problem is based on three factors; the pessimism of the schedulability analysis employed, the number of items to be allocated relative to the size of the platform and the effect of restrictions on the search space.

The X-Topmeter / X-Alloc approach solves a slightly different problem to those addressed by these authors: produce a hardware platform of minimum size, and accompanying software allocation, that meets a set of timing requirements. This approach increases the set of feasible alternatives. As a result, the effect of restrictions on the search space appears to be less pronounced than for other SA based approaches. The effect on the search engine of trying to place large numbers of sub-tasks onto a small platform is also reduced.

X-Alloc produces results of a similar quality to that produced by Tindell and previous tools produced by the author. In fact the WCRT predictions are less pessimistic as they use a schedulability analysis designed to improve on the work of Tindell. No comparative results are available for the work of Alternberd.

6.2.4 Interactions with other Design Issues

Interactions between design issues remains an area of great debate, but little research. Trade-offs are typically qualitative rather than quantitative. Research into quantitative trade-offs which is compatible with, and complementary to, the work presented in this thesis has been produced by Keeney & Raiffa [83] and Prasad [131].

The X-Topmeter / X-Alloc approach allows interactions between the topology problem and other design issues to be addressed via a set of labelled DAGs and through the use of evaluation functions that quantify the value of a particular set of trade-offs. The evaluation functions employed result from consultation with industrial partners, perusal of the literature and discussion with colleagues working in similar research areas. It has proved impossible to compare these functions with those used in industry, as industry has only recently begun to evaluate design trade-offs quantitatively, rather than qualitatively.

Results for the case study however, are in accordance with a qualitative evaluation

undertaken by a wide variety of students with industrial expertise attending a module of the MSc in safety critical systems. This case study has also been used as an example on industrially based courses.

6.2.5 Industrial Practice

Industrial practice has lagged behind the research community in the use of modelling and evaluation techniques to determine an appropriate level of redundancy for fault-tolerance and allocation of sub-tasks. Modelling and evaluation of the reliability aspects of a system is typically undertaken by a separate group to the designers once a full system design has been proposed. Assumptions are made in the modelling process on a second-hand basis, with very little checking from the designers, from reports produced by the designers. Reliability measures are used as a qualitative measure of the dependability of the design. There is often a great deal of pressure on the analysts to produce the “correct” result.

Recently some efforts have been taken to introduce reliability modelling and prediction into the early stages of the design process. One industrial partner has sought guidance on the use of RBD models, and evaluation techniques, as part of an analysis of a proposed system design at a very early stage in the design process. This is a first for them, but shows that early predictive analysis is becoming more important in an industrial setting. More common is to use quantitative techniques to allocate failure rate requirements to sub-systems. This approach can be seen in proposed standards, such as ARP 4761 [150]. These failure rates are then passed on as requirements to sub-contractors.

One aim of the X-Topmeter / X-Alloc approach is to make a first step towards allowing the designer to produce appropriate system and reliability models, with the guidance of specialists. A second aim is to promote a greater understanding of the impact of design decisions on the size of platform required and the reliability / timing aspects of a control system. To achieve this a more extensive GUI for the production of the resource and reliability model libraries will be required in future implementations of X-Topmeter and X-Alloc.

The X-Topmeter / X-Alloc approach is *blue-sky* research as far as industrial practice is concerned. However, as an approach it has received a good response when presented to industrialists. In the short term aspects of this approach are likely to be taken up, such as reliability modelling during design. In the long term it provides a framework for future process improvement activities.

Guided search techniques are beginning to emerge as solution engines for control problems [127]. However, they have not been used as a design aid. Recently, Rolls Royce and Associates have recognised the need to consider trade-offs as part of a design

process and have included an investigation of the ability to perform trade-off analysis as one of its mission statements for a new University Technology Centre. The X-Topmeter and X-Alloc prototype tools have shown that guided search techniques are strong candidates for use in this role.

6.3 Future Work

The approach presented in this thesis attempts to bring together a number of research strands. A set of prototype tools have been produced. Both these factors mean that significant work would be required to bring this approach up to a state where it could be used in industry. Three areas of future work are therefore highlighted:

- Use of this approach during Procurement and the Lifetime of the system
- Extensions to the effectiveness of the search tools and the evaluation functions employed
- Extensions to the usability of the tools

6.3.1 Procurement and System Lifetime

The current version of the X-Topmeter / X-Alloc approach is heavily biased towards aiding the design process. However, it may be extended up the system life-cycle to aid procurement and requirements decisions. Procurement decisions are becoming particularly important in industries with long development cycles or system lifetimes. Procurers can use a high level model of the system to investigate the set of hardware resources required to fulfil dependability and timing requirements. They can employ appropriate cost and maintenance models to investigate different combinations of hardware resources. Most important of all it may be possible to produce a set of requirements for any hardware used in a particular system. This may help guard against obsolescence since any unit that meets these requirements can be used in the system topology.

The X-Topmeter / X-Alloc approach may also be extended into the operating lifetime of the system to gauge the effect of a mid-life update on the dependability and timing properties of a topology. It may also be used to evolve a new topology or to track a topology during operation.

The ability to track a changing environment may be helpful in resolving the “in-service” Topology Problem. That is, to reconfigure the topology to get the best reliability/availability out of non failed units. Cobb and Goldberg [30] postulate two diversification operators to track changing environments: *Random Immigration* and

Hyper-mutation. In the random immigrants mechanism the *replacement rate* specifies the fraction of the population that is replaced each generation by randomly generated genes. The triggered hyper-mutation mechanism uses uniformly distributed mutation. When the adaptive mechanism is triggered due to a degradation in performance, the level of mutation is dramatically increased. Once better performance is achieved the mutation rate drops to a background level.

6.3.2 Extensions to the Search Tools

Human operators interfacing with a complex system will affect the flow of information between units and will themselves have specific timing requirements [67]. Operators have both a rate of output and a level of reliability which depends on factors such as

- decision making requirements,
- training and experience and
- amount of information received.

Suppose that an operator has three attributes; a name, a rank and a skill level per skill. Operators can be assigned to jobs based on these attributes and an evaluation function produced to analyse each such assignment based on a set of SDFs. The SDFs to incorporate into the topology evaluation function would be:

- Required number of operators
- number of simultaneous (concurrent) users
- user intensity measured as system processing time over user response time
- system performance with respect to interaction with operators.

Thus, an extension of the topology problem to incorporate assignment of operators can be envisaged. The design space increases by the number of operators times the number of positions that the operator can be placed. Operators interact with other elements of a topology by placing constraints on them. For instance, a set of output devices may need to be in close proximity to the operator restricting the permissible allocations of hardware to particular physical locations. The operator will also have an impact on the reliability of the system. It is likely that this extension will be more useful for sensitivity analysis, rather than during “real” design. A number of unconvincing measures of operator reliability have been produced [112].

The X-Topmeter / X-Alloc approach can be extended to include a measure of the quality of the control functionality. GA searches have already be employed to determine

the optimal form of control algorithms for a fixed platform control system. Additions to the evaluation function to allow limited trade-offs between functionality and reliability can be envisaged.

Performability allows a trade off between reliability and performance of a (sub)-system to be undertaken as a single measure. A state based reliability model of a (sub)-system is produced and the transitions between states are labelled with a performance measure. A measure of the *risk* attached to a solution could be used or a measure of timing characteristics of the solution. In the latter case a multiplicative timing and reliability evaluation function element may be employed. The existing X-Topmeter library of state based reliability models may be extended to incorporate reward measures. The SHARPE tool is able to produce expected value and expected accumulated reward measures for the reward models.

Maintainability measures of a proposed topology may be employed. This extension may require a simulation to be run of the topology, which is particularly time consuming. It therefore appears better to stick with applying a maintainability measure to the best topology produced by an X-Topmeter experiment.

A variety of TS style restrictions on the search space, and accompanying additions to the evaluation functions of both X-Topmeter and X-Alloc, can be envisaged. The prototype tools employ a very limited number of such restrictions and naive extensions to the evaluations functions. This extension will be a valuable addition as the size and complexity of examples increases.

Horn argues that GAs can be used to deal with multi-objective optimisation by adding the concepts of Pareto domination to the genetic operators and by applying *Niching pressure* to spread the population out along the Pareto optimal trade-off surface. A Pareto surface consists of all solutions whose combinations of characteristic values produce the *same* overall utility. Niches [70] are zones of the search space. Particular elements of a solution are kept intact and allowed to compete as a group against other zones in the search space. It may be possible to introduce topological zones, such as groups of sensors at the same physical location, to the topology problem [160].

So, one major piece of work is to extend X-Alloc to explicitly include sensor and actuator allocation. Allocation in this case takes the form of a grouping of sensors and actuators, both logically and to physical locations. For instance, a logical sensing group consists of sensors that provide data for a number of control services. A physical sensor / actuator group takes the form of a set of units placed in the same part of the system environment. This becomes particularly important if the ability to compare integrated and distributed topologies is required, see CABV2 and CABV3.

6.3.3 Extensions to the Usability of the Tools

The prototype X-Topmeter tool requires a great deal of information to be produced, so that a set of resource and reliability model libraries can be produced. Even more data on the characteristics of the proposed system is required by the X-Alloc tool. The current approach is to produce an input file, with a line per data point. This is not satisfactory.

One area of fruitful research would be to produce a set of Graphical User Interfaces to allow the library data to be input. Of particular interest would be a GUI version of a reliability model generator. The front-end could be similar to the front-ends employed by X-Topmeter and X-Alloc. Interfaces of this type are not easy and require considerable programming effort.

6.4 Final Comments

In this thesis emphasis has been placed on the use of quantitative techniques to support the topology selection issue for safety-critical real-time systems. Production of an ‘industrial quality’ topology selection process, that may be used to investigate the trade-offs implicit in topology selection, poses a challenge for the research community. This process would require a synthesis of numerous strands of the existing research into SC-RT systems. Furthermore, the problems addressed are NP-hard optimisation problems. A first attempt at producing an approach that incorporates such a synthesis has been presented.

In principal, the X-Topmeter / X-Alloc approach may be employed by any system designer to investigate the set of admissible topologies for his, or her, proposed system. In practice, a great deal of domain knowledge and analytical skill would be required to repeat the work undertaken in this thesis. This is inherent in the nature of the problem.

If the ultimate aim of allowing the designer to undertake topology selection is to be fulfilled three enhancements of the current situation in industry need to be made. First, the process for the production of safety-critical systems will need to explicitly consider the topology selection problem. Second, significant research into Reliability Model Generators is required. A great deal of the complexity in this thesis arises from the need to generate dependability models to evaluate the set of alternative architectural topologies. Finally, a suite of tailored GUIs will be required to ‘lead’ the designer through the elements of the approach.

Although much remains to be done, this thesis has clearly shown how computer aided support can be effectively given to complex activities, such as architectural design.

Bibliography

- [1] J. T. Alander. An Indexed Bibliography of Genetic Algorithms. Technical Report 97-3, University of Vaasa, 1997. Available via ftp - ftp://ftp.uwasa.fi/cs/.
- [2] M. Ali and C. Storey. The Optimal Control of Vehicle Suspension Systems. In I.C. Parmee, editor, *Adaptive Computing in Engineering Design and Control*, pages 167–173, University of Plymouth, UK, March 1996. PEDC.
- [3] P. Alternberd. Multiprocessor Allocation of Periodic Hard Real-Time Tasks. Technical Report 10/96, C-LAB, C-LAB Furstenallee 11 D 33095 Paderborn, July 1996.
- [4] J. Arlat, K. Kanoun, and J. Laprie. Dependability Modelling and Evaluation of Software Fault-Tolerant Systems. *IEEE Transactions on Computers*, 39(4):504–513, April 1990.
- [5] M. Armstrong. Joint Reliability-Importance of Components. *IEEE Transactions on Reliability*, 44(3):408–412, 1995.
- [6] N.C. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed Priority Pre-emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.
- [7] L. Baccouche. Efficient Static Allocation of Real-Time Tasks Using Genetic Algorithms. Technical report, Imag Institute, Laboratoire de Genie Informatique, 1995.
- [8] J. Bannister and K. Trivedi. Task Allocation in Fault-Tolerant Distributed Systems. *Acta Informatica*, 20:261–282, 1983.
- [9] M. Barborak, M. Malek, and A. Dahbura. The Consensus Problem in Fault-Tolerant Computing. *ACM Computing Surveys*, 25(3):171–220, 1993.
- [10] I. Bate and A. Burns. Schedulability Analysis of Fixed Priority Real-Time Systems with Offsets. In *9th Euromicro Workshop on Real-Time Systems*, pages 153–160, Toledo, Spain, 1997.
- [11] S. Bavuso, J. Dugan, K. Trivedi, N. Mittal, and M. Boyd. *HARP Introduction and Guide for Users*. NASA, 7.0 edition, 1993.
- [12] D. Beasley, D. Bull, and R. Martin. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing*, 15(2):58–69, 1993.
- [13] D. Beasley, D. Bull, and R. Martin. An Overview of Genetic Algorithms: Part 2, Research Topics. *University Computing*, 15(4):170–181, 1993.
- [14] D. Beasley, D. Bull, and R. Martin. Reducing Epistasis in Combinatorial Problems by Expansive Coding. In *Proc. Fifth Int. Cong. on GAS*, pages 400–407, 1993.
- [15] J. Bland and G. Dawson. Tabu Search and Design Optimisation. *Comput. Aided Des.*, 23(3):195–201, 1991.
- [16] J. Bowles. A Survey of Reliability-prediction Procedures for Microelectronic Devices. *IEEE Transactions on Reliability*, 41(1):2–20, 1992.

- [17] E. Brehm. System Dependability Assessment Tool. In A. Stoyenko, editor, *2nd ICECCS*, pages 116–120, Montreal, Canada, October 1996. IEEE.
- [18] S. Brilliant, J. Knight, and N. Leveson. Analysis of Faults in an N-version Software Experiment. *IEEE Transactions on Software Engineering*, 16(2):238–47, 1990.
- [19] C. Brind, C. Muller, and P. Prosser. Stochastic Techniques for Resource Management. *BT Technol. J.*, 13(1):55–63, 1995.
- [20] D. Buede. Aiding Insight II - Decision Analysis Software Survey. *OR/MS Today*, 21(3):62–71, 1994.
- [21] A. Burns. Preemptive Priority-Based Scheduling: An Appropriate Engineering Approach. In *Advances in Real-Time Systems*, page 239. Prentice-Hall, 1995.
- [22] A. Burns, A. Hutcheon, B. Sharp, and A. Wellings. The Design and Development of Safety Kernels. Technical Report CI/GNSR/27, York Software Engineering for H.S.E., 1994.
- [23] A. Burns and A. Lister. A Framework for Building Dependable Systems. *The Computer Journal*, 34(2):173–181, 1991.
- [24] R. Butler. The SURE Approach to Reliability Analysis. *IEEE Transactions on Reliability*, 41(2):210–218, 1992.
- [25] D. Carman and A. Dolinsky et al. Software Reliability Engineering Study of a Large Scale Telecommunications Software System. In *6th ISSRE*, pages 350–359, Toulouse, France, October 1995.
- [26] R. Chapman, A. Burns, and A. Wellings. SPATS - A New Toolkit for High-Integrity Ada Development. In *1995 Ada UK International Conference*, 1995.
- [27] M. S. Chern. On the Computational Complexity of Reliability Redundancy Allocation in a Series System. *Operations Research Letters*, 11:309–315, 1992.
- [28] C. Y. Choi, B. W. Johnson, and J. A. Profeta III. Safety Issues in the Comparative Analysis of Dependable Architectures. *IEEE Transactions on Reliability*, 46:316–322, 1997.
- [29] G. Ciardo, J. Muppala, and K. Trivedi. On the Solution of GSPN Reward Models. *Performance Evaluation*, 12(4):237–254, 1991.
- [30] H. Cobb and J. Grefenstette. Genetic Algorithms for Tracking Changing Environments. In *Proceedings of the International Genetic Algorithms Conference*, 1993.
- [31] D. Coit and A. Smith. Reliability Optimisation of Series-Parallel Systems Using a Genetic Algorithm. *IEEE Transactions on Reliability*, 45(2):254–260, June 1996.
- [32] M. Coli and P. Palazzari. A New Method for Optimisation of Allocation and Scheduling in Real-Time Applications. In *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, pages 262–269, 1995.
- [33] D. T. Connolly. An Improved Annealing Scheme for the QAP. *EJOR*, 47:93–100, 1990.
- [34] I. Crabtree. Resource Scheduling - Comparing Simulated Annealing with Constraint Programming. *BT Technol. J.*, 13(1):121–127, 1995.
- [35] J. Dainter. BR710 Timing and Memory Analysis. Technical Report DNS 13230, RoSEC Derby, Rolls-Royce plc, Derby, 1997.
- [36] C. Das, J. Kreulen, and M. Thazhuthaveetil. Dependability Modelling for Multiprocessors. *IEEE Computer*, pages 7–19, October 1990.
- [37] L. Davis. Adapting Genetic Operator Probabilities in Genetic Algorithms. In *Proc. 3rd Int. Conf. on GAs*, pages 61–69, 1989.

- [38] R. Diekmann, R. Luling, B. Monien, and C. Spraner. A Parallel Local-search Algorithm for the k-partitioning Problem. In *Proceedings of 28th Hawaii International Conference on System Science*, 1994.
- [39] M. DiNatale and J. A. Stankovic. Applicability of Simulated Annealing Methods to Real-Time Scheduling and Jitter Control. In *Proceedings of the 7th Euromicro Workshop on Real-Time Systems*, 1995.
- [40] K. Dowsland. Variants of Simulated Annealing for Practical Problem Solving. In *Proc. Adaptive Computing and Information Processing Conference*, pages 115–133, 1994.
- [41] K. Dowsland and J. M. Thompson. Variants of Simulated Annealing for the Examination Timetabling Problem. *Annals of O.R.*, 1995.
- [42] S. Draber. Reliability-Orientated Design of a Distributed Control System for High-Voltage Switchgear Operations. In *FTCS-27*, pages 385–389, Seattle, Washington, US, June 1997. IEEE Computer Society.
- [43] K. Driscoll and K. Hoyme. The Airplane Information Management System: an Integrated Real-Time Fight-deck Control System. In *Real-Time Systems Symposium*, pages 267–270. IEEE, December 1992.
- [44] J. Dugan. Fault Trees and Imperfect Coverage. *IEEE Transactions on Reliability*, 38(2):177–185, June 1989.
- [45] J. Dugan, S. Bavuso, and M. Boyd. Modelling Advanced Fault-Tolerant Systems with HARP. In *Proc. of Reliability and Maintainability Symposium*, pages 1–25, 1991.
- [46] J. Dugan, S. Bavuso, and M. Boyd. Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems. *IEEE Transactions on Reliability*, 41(2):364–377, September 1992.
- [47] J. Dugan and R. Van Buren. Reliability Evaluation of Fly-by-Wire Computer Systems. *J. Systems and Software*, 25:109–120, 1994.
- [48] P. D. Ezhilchelvan and S. K. Shrivastava. A Classification of Faults in Systems. Technical report, Computer Science, University of Newcastle-upon-Tyne, 1991.
- [49] J. Ferland, A. Hertz, and A. Lavoie. An Object-Oriented Methodology for Solving Assignment-Type Problems with Neighbourhood Search Techniques. *Operations Research*, 44(2):347–359, March 1996.
- [50] J. Ribeiro Filho, P. Treleaven, and C. Alippi. Genetic Algorithm Programming Environments. *Computer*, 27(6):27–45, 1994.
- [51] C. A. Fleurent and J. C. Ferland. Genetic and Hybrid Algorithms for Graph Coloring. Technical Report 94-1, Universite de Montreal, 1994.
- [52] C. A. Fleurent and J. C. Ferland. Object-Oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique and Satisfiability. Technical Report 94-2, Universite de Montreal, 1994.
- [53] G. Fohler and C. Koza. Heuristic Scheduling for Distributed Real-Time Systems. Technical Report 6/89, Institut für Technische Informatik, Technische Universität Wien, Wien, Austria, April 1989.
- [54] S. Forrest and M. Mitchell. What Makes a Problem Hard for a Genetic Algorithm? Some Anomolous Results & Their Explanation. *Machine Learning*, pages 1–38, 1993.
- [55] B. Fox. Integrating and Accelerating Tabu Search, Simulated Annealing and Genetic Algorithms. *Annals of OR*, 41:47–67, 1993.
- [56] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to NP-Completeness*. W. H. Freeman, 1979.

- [57] R. Geist and K. Trivedi. Ultrahigh Reliability Prediction for Fault-Tolerant Computer Systems. *IEEE Transactions on Computers*, 32(12):1118–1127, December 1983.
- [58] R. Geist and K. Trivedi. Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques. *IEEE Computer*, pages 52–61, July 1990.
- [59] M. Glocker, S. Jockush, and N. Weber. Assessment and Optimisation of Systems Architectures: Experience from Industrial Applications at Siemens. In A. Stoyenko, editor, *2nd ICECCS*, pages 400–407. IEEE, October 1996.
- [60] F. Glover, E. Taillard, and D. deWarra. A User’s Guide to Tabu Search. *Annals of OR*, 41:3–28, 1993.
- [61] D. Goldberg, K. Deb, and J. Horn. Genetic Algorithms, Noise, and the Sizing of Populations. *Complex Systems*, 6:333–362, 1992.
- [62] D. Goldberg and C. Todd. Genetic Algorithms: A Bibliography. Technical Report 92008, Illinois Genetic Algorithms Laboratory, 1992.
- [63] D. A. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley, 1989.
- [64] J. W. Greene and K. J. Supowit. Simulated Annealing Without Rejected Moves. *IEEE Transactions on CAD*, 5:221–228, 1986.
- [65] G. Greenwood, C. Lang, and S. Hurley. An Evolutionary Strategy for Scheduling Periodic Tasks in Real-Time Systems. In *Applied Decision Technologies*, pages 171–188, 1995.
- [66] J. Grefenstette. Parallel Adaptive Algorithms for Function Optimisation. Technical Report CS-81-19, Vanderbilt University, Nashville, 1981.
- [67] M. Harellick, C. Amaro, et al. Operator Resources for Large Complex Systems. In A. Stoyenko, editor, *2nd ICECCS*, pages 112–115, Montreal, Canada, October 1996. IEEE.
- [68] A. Hertz and D. de Werra. Using Tabu Search Techniques for Graph Coloring. *Computing*, 39:345–351, 1987.
- [69] C. Höhn and C. Reeves. Graph Partitioning Using Genetic Algorithms. Technical report, Control Theory and Applications Centre, Coventry University, UK, 1995.
- [70] J. Horn and N. Nafpliotis. Multiobjective Optimisation Using the Niched Pareto Genetic Algorithm. Technical Report 93005, Illinois Genetic Algorithms Lab., 1993.
- [71] J.J. Horning, H.C Lauer, et al. *LNCS 16*, chapter A Program Structure for Error Detection and Recovery, pages 172–187. Springer-Verlag, 1974.
- [72] J.E. Hosford. Measures of Dependability. *Operations Research*, 8(1):204–206, 1960.
- [73] C.R. Houck, J. Joines, and M. Kay. Utilising Lamarkian Evolution and the Baldwin Effect in Hybrid Genetic Algorithms. Technical report, Dept. of Industrial Engineering, North Carolina State University, Raleigh, NC, US, 1995.
- [74] Aeronautical Radio Inc. Multitransmitter Data Bus: Part 1 - Protocol Description. Technical Report 629, ARINC, Annapolis, MD, USA, 1993.
- [75] L. Ingber. Simulated Annealing: Practice versus Theory. *Mathl. Computer Modelling*, 18(11):29–57, 1993.
- [76] ISO. Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication. Technical Report ISO DIS 11898, Draft for International Standard, 1992.
- [77] B.W. Johnson. *Design and Analysis of Fault-Tolerant Systems*. Addison-Wesely, 1988.

- [78] D. Johnson. Integrated Modular Avionics: A scheme for autonomous dynamic system reconfiguration. *Computer Syst. Sci and Eng*, 11(3):125–133, May 1996.
- [79] D. Johnson, C. Argon, L. McGeoch, and C. Schevon. Optimisation by Simulated Annealing: An Experimental Evaluation; part 1, Graph Partitioning. *Opns. Res.*, 37:865–892, 1989.
- [80] Z. Kalbarczyk and J. Christmansson. Technical Approaches for Reducing the Probability of Common-Cause / Common Mode Failures - A Survey. Technical Report 237, LDC, Chalmers University of Technology, Göteborg, Sweden, May 1995.
- [81] K. Kanoun and J.C Laprie. *Predictably Dependable Computing Systems*, chapter Software Reliability Trend Analysis: from Theoretical to Practical Considerations, pages 373–406. Springer, 1995.
- [82] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based Analysis of Software Architectures. *IEEE Software*, pages 47–53, November 1996.
- [83] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley & Sons, 1976.
- [84] J. Kelly, T. McVitte, and W. Yamamoto. Implementing Design Diversity to Achieve Fault-Tolerance. *IEEE Software*, pages 62–71, July 1991.
- [85] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimisation By Simulated Annealing. *Science*, 220:671–680, 1983.
- [86] H. Kopetz and A. Damm. Distributed Fault Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*, pages 25–40, 1989.
- [87] H. Kopetz and W. Ochsenreiter. Clock Synchronisation in Distributed Real-time Systems. *IEEE Transactions on Computers*, 36(8):933–40, 1987.
- [88] J. H. Lala and R.E. Harper. Reducing the Probability of Common-Mode Failures in the Fault-Tolerant Parallel Processor. In *12th IEEE/AIAA Digital Avionics Systems Conference*, pages 221–230, 1993.
- [89] T.G. Lane. Studying Software Architectures Through Design Spaces and Rules. Technical Report CMU/SEI-90-TR-18, Carnegie-Mellon University, 1990.
- [90] J. Laprie, J. Arlat, and C. Beounes. Definition and Analysis of Hardware and Fault-Tolerant Architectures. *IEEE Computer*, pages 39–51, July 1990.
- [91] J.C. Laprie. Dependability: A Unifying Concept for Reliable Computing and Fault-Tolerance. In T. Anderson, editor, *Dependability of Resilient Computers*, chapter 1, pages 1–28. BSP Professional Books, 1989.
- [92] J.C. Laprie, editor. *Dependability: basic concepts and terminology: in English, French, German, Italian and Japanese*, volume 5 of *Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, Wien, etc., 1992.
- [93] E. L. Lawler. *Recent results in the Theory of Machine Scheduling*, pages 202–233. Springer Verlag, 1983.
- [94] Y. Leung. Processor Assignment and Execution Sequence for Multiversion Software. *IEEE Transactions on Computers*, 46(12):1371–1377, December 1997.
- [95] N. Leveson. The Challenge of Building Process-Control Software. *IEEE Software*, 7:55–62, 1990.
- [96] Y. Li and Y. Jiang. Localised Simulated Annealing in Constraint Satisfaction and Optimisation. In *Applied Decision Technologies*, pages 221–232, 1995.

- [97] C. Liceargo and D. Siewiorek. Automatic Specification of Reliability Models for Fault-Tolerant Computers. Technical Report 3301, NASA, July 1993.
- [98] C. Lindemann, M. Malhotra, and K. Trivedi. Numerical Methods for Reliability Evaluation of Markov Closed Fault-Tolerant Systems. *IEEE Transactions on Reliability*, 44(4):694–703, December 1995.
- [99] R. Logendran and A. Sonthinen. A Tabu-Search Based Approach for Scheduling Job-shop Type Flexible Manufacturing Systems. *J. Opl. Res. Soc.*, 48(3):264–77, 1997.
- [100] S. Mahfoud. Finite Markov Chain Models of an Alternate Selection Strategy for the Genetic Algorithm. Technical Report 91007, Illinois Genetic Algorithms Laboratory, 1993.
- [101] O. Maimon and D. Braha. On the Complexity of the Design Synthesis Problem. *IEEE Transactions on Syst., Man, and Cybern.*, 26(1):142–151, January 1996.
- [102] M. Malhotra. *Specification of Dependability Models for Fault-Tolerant Systems*. PhD thesis, Graduate School of Duke University, USA, 1993.
- [103] J. Mann. *SAMson v1.5 User Manual*. School of Information Systems, University of East Anglia, 1996.
- [104] J. Mann. *GAMeter v2.0 User Manual*. School of Information Systems, University of East Anglia, 1997.
- [105] J. Mann, G. Smith, and A. Kapsalis. *Applications of Modern Heuristic Search Techniques*, chapter The GAMeter Toolkit. Alfred Walker, 1995.
- [106] J. McDermid, M. Nicholson, and D. Pumfrey. Experience with the Application of HAZOP to Computer-based Systems. In *COMPASS 95: National Institute of Standards and Technology, Gaithersburg, MD*, page 1, 1995.
- [107] J.A. McDermid. On Dependability, its Measurement and its Management. *High Integrity Systems Journal*, 1(1):17–26, 1994.
- [108] J. McWha. The Boeing Company - 777 Systems Overview. In *Royal Aeronautical Society Symposium*, pages –, 1993.
- [109] MISRA. *Development Guidelines for Vehicle Based Software*. Motor Industry Research Association (ISBN 0 9524156 0 7), 1994.
- [110] MISRA. Report 2: Integrity. Technical Report 2, MIRA, 1995.
- [111] MISRA. Report 3: Noise, EMC and Real-Time. Technical Report 3, MIRA, 1995.
- [112] MISRA. Report 8: Human Factors in Software Development. Technical Report 3, MIRA, 1995.
- [113] P. Mohapatra, C. Yu, and C. Das. Allocation and Mapping Based Reliability Analysis of Multistage Interconnection Networks. *IEEE Transactions on Computers*, 45(5):600–606, May 1996.
- [114] T. Nara, M. Nakata, and A. Ooishi. Software Reliability Growth Analysis - Application of NHPP Models and its Evaluation. In *6th ISSRE*, pages 250–255, Toulouse, France, October 1995.
- [115] M. Nicholson and A. Burns. Emergence of an Architectural Topology for Safety-Critical Real-Time Systems. Technical Report YCS-97-292, Department of Computer Science, University of York., 1997.
- [116] M. Nicholson and A. Burns. Structuring Architectural Topologies for Real-Time Safety-Critical Systems. Technical Report YCS-97-284, Department of Computer Science, University of York., 1997.

- [117] M. Nicholson, A. Burns, K. Tindell, and N. Zhang. Allocation of Safety Critical Hard Real-Time Tasks on a Parallel Processing Platform. Technical Report YCS-94-238, Department of Computer Science, University of York., 1994.
- [118] M. Nicholson and J. McDermid. Analysis of Dependable computer Systems. Technical Report YCS-94-245, Department of Computer Science, University of York, 1994.
- [119] M. Nicholson and J. McDermid. Quantification in Hazard Analysis. Technical Report Hazard Analysis of Safety-Critical Systems Course, April 1996, Department of Computer Science, University of York, 1996.
- [120] M. Nicholson, J. McDermid, and A. Burns. Analysis and Design Synthesis for Hard Real-time Safety Critical Systems. Technical Report YCS-94-237, Department of Computer Science, University of York, 1994.
- [121] M. Nicholson and D. Prasad. Design Synthesis Using Adaptive Search Techniques and Multi-criteria Decision Analysis. In A. Stoyenko, editor, *Second ICECCS*, pages 523–525, Montreal, Quebec, Canada, October 1996. IEEE Computer Society, IEEE.
- [122] M. Nicholson and D. Pumfrey. Hazard Analysis of a Computer Assisted Braking System. In *Hazard Analysis Course for the SVRC, Brisbane, Australia*, page 1, 1995.
- [123] A. Nikora and M. Lyu. Applying Reliability Models More Effectively. *IEEE Software*, pages 43–52, 1992.
- [124] A. Nikora and M. Lyu. Software Reliability Engineering Study of a Large Scale Telecommunications Software System. In *6th ISSRE*, pages 250–255, Toulouse, France, October 1995.
- [125] L. Painton and J. Campbell. Genetic Algorithms in Optimisation of System Reliability. *IEEE Transactions on Reliability*, 44(2):172–178, 1995.
- [126] C. Y. Park and A. C. Shaw. Experiments with a Program Timing Tool Based on Source-Level Timing Schema. In *Proceedings Real-Time Systems Symposium*, pages 73–81, 1989.
- [127] I.C. Parmee. Adaptive Computing in Engineering Design and Control. In *Proceedings of 2nd International Conference*. PEDC, March 1996.
- [128] S. Porto and C. Ribeiro. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints. Technical Report 93-03-porto, Catholic University of Rio, 1993.
- [129] J. Potts, T. Giddens, and S. Yadav. The Development and Evaluation of an improved GA Based on Migration & Artificial Selection. *IEEE Transactions on Sys. Man, & Cybernetics*, 24(1):73–86, 1994.
- [130] D. Powell. Preliminary Definition of the GUARDS Architecture. Technical report, ES-PRIT Project 20716, 1997.
- [131] D. Prasad. *Dependable Systems Integration Using Decision Analysis*. PhD thesis, Department of Computer Science, University of York, 1998.
- [132] D. Prasad, J. McDermid, and I. Wand. Dependability Terminology: Similarities and Differences. *IEEE Aerospace and Electronic Systems Magazine*, 11(1):14–21, January 1996.
- [133] D. Pumfrey and M. Nicholson. Hazard Analysis of a Computer Assisted Braking System. In *Hazard Analysis Course for MSc in Safety Critical Systems*, page 1, 1996.
- [134] S. Punnekkat. *Schedulability Analysis for Fault-Tolerant Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1997.
- [135] B. Purohit, T. Clark, and T. Richards. Techniques for Routing and Scheduling Services on a Transmission Network. *BT Technol. J.*, 13(1):64–72, 1995.

- [136] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. In *10th International Conference on Distributed Computing Systems*, pages 108–115, 1990.
- [137] K. Ramamritham. Allocation and Scheduling of Complex Periodic Tasks. *IEEE Transactions on Parallel and Dist. Sys.*, 6(4):412–420, 1995.
- [138] B. Randell. System Structure for Software Fault Tolerance. *IEEE Transactions on Software Engineering*, pages 220–232, 1975.
- [139] V. Rayward-Smith, editor. *Applications of Modern Heuristic Methods*. Alfred Waller, 1995.
- [140] C. Reeves. Diversity and Diversification in GAs: Some Connections with Tabu Search. In *Proc. of Int. Conf. on ANN and GAs*, pages 344–351, 1993.
- [141] C. Reeves. *Modern Heuristic Techniques for Combinatorial Optimisation*. Blackwell, 1993.
- [142] C. Reeves. Integrating Local Search into Genetic Algorithms. In *Applied Decision Technologies*, pages 261–276, 1995.
- [143] D. Rehman, E. Lock, and C. Nguyen. A Generalised Methodology for Evaluating and Optimizing System Design Factors. In A. Stoyenko, editor, *2nd ICECCS, Montreal, Canada*, pages 212–215. IEEE, IEEE, October 1996.
- [144] A. L. Reibman and A. M. Veerarghavan. Reliability Modelling: An Overview for System Designers. *IEEE Computer*, pages 49–57, 1991.
- [145] J. Reisinger. *MARS Operating System User Manual: Version 2.1*. Institut fur Technische Informatik, 1993.
- [146] F.S. Roberts. *Measurement Theory, with Applications to Decision-making, Utility and the Social Sciences*. Addison Wesley, 1979.
- [147] D. Rowe. Development of a Systems Based Architecting Process for Computer Based Systems. In A. Stoyenko, editor, *2nd ICECCS, Montreal, Canada*, pages 200–203. IEEE, IEEE, October 1996.
- [148] D. Rowe and J. Leaney. Evaluating Evolvability of Computer Based Systems Architectures - An Ontological Approach. Technical report, Draft for ECBS 97 submission, 1997.
- [149] RTCA. *RTCA/DO-178B: Software Considerations in Airborne Systems and Equipment Certification*. Radio Technical Commission for Aeronautics, December 1992.
- [150] SAE. *Aerospace Recommended Practice 4761*. SAE International 400 Commonwealth Drive, Warrendale, PA , 12 edition, 1996.
- [151] SAE - report SP-108. *ABS/TCS and Brake Technology Development*. SAE, April 1994.
- [152] R. Sahner, K. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems. An example-based approach using the SHARPE software package*. Kluwer Academic Publishers, 1995.
- [153] W. H. Sanders, W. D. Obal, et al. The UltraSAN Modelling Environment. *Performance Evaluation*, pages 89–115, October 1995.
- [154] F.E. Sandnes. A Hybrid Genetic Algorithm Applied to Automatic Parallel Controller Code Generation. In *Proceedings of National the 8th Euromicro Workshop on Real-Time Systems*, pages 70–75, 1996.
- [155] B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfaction Problems. In *Proceedings of tenth National Conference on Artificial Intelligence*, pages 440–446, July 1992.

- [156] H. Simpson. Data Interaction Architecture (DIA) for Computer based System Engineering. In *Proceedings of JFIT Conference, Edinburgh, 1994*.
- [157] T. E. Smith and D. E. Setliff. Using Simulated Annealing to Synthesise Resource Bounded Software. *Automated Software Engineering*, 1:155–176, 1994.
- [158] E. Talbi and T. Muntean. General Heuristics for the Mapping Problem. Technical report, Institut IMAG - Laboratoire de Genie Informatique, 1992.
- [159] H. A. Thompson. Fault Tolerant System Design - Tool Integration. Technical Report RRUTCShef/R/95014, Rolls-Royce Control and Systems Engineering UTC, Department of Automatic Control and Systems Engineering, University of Sheffield, January 1996.
- [160] H. A. Thompson. Multi-Disciplinary Optimisation Environment for an ASTOVL Aircraft. Technical Report RRUTCShef/R/98003, Rolls-Royce Control and Systems Engineering UTC, Department of Automatic Control and Systems Engineering, University of Sheffield, March 1998.
- [161] F. Tillman, C. Hwang, and W. Kuo. Determining Component Reliability and Redundancy for Optimum System Reliability. *IEEE Transactions on Reliability*, 26:162–165, July 1977.
- [162] K. W. Tindell. Holistic Schedulability Analysis for Distributed Hard Real-Time Systems. Technical Report YCS-94-197, Department of Computer Science, University of York., 1993.
- [163] K.W. Tindell, A. Burns, and A.J. Wellings. Allocating Real-Time Tasks: An NP-Hard Problem Made Easy. *Real Time Systems Journal*, 4:145–165, 1992.
- [164] C. A. Tovey. Simulated Simulated Annealing. *AJMMS*, 8:389–407, 1988.
- [165] C. Tudge. *The Engineer in the Garden*. Jonathon Cape, 1993.
- [166] US Department of Defence. *MIL-HBDK 217E: Reliability Prediction of Electronic Equipment*. US Department of Defense, 1986.
- [167] Various. Genetic Algorithms in Engineering Systems: Innovations and Applications. In *GALESIA '95*, pages 1–548, Halifax house, Sheffield, UK, 1995. IEE, IEE.
- [168] A. Villemeur. *Reliability, Availability, Maintainability and Safety Assessment*. John Wiley and Sons Ltd., 1992.
- [169] G. Waligora. A Tabu Search Algorithm for Some Discrete-Continuous Scheduling Problems. In *Applied Decision Technologies*, pages 253–260, 1995.
- [170] T. Warwick and E. Tsang. Using a Genetic Algorithm to Tackle the Processors Configuration Problem. In *Symposium on Applied Computing*, 1994.
- [171] S. Welke, B. Johnson, and J. Ayler. Reliability Modelling of Hardware / Software Systems. *IEEE Transactions on Reliability*, 44(3):413–418, September 1995.
- [172] S. R. Welke. A Unified Model of Hardware/Software Reliability. Technical report, University of Virginia, Charlottesville, Virginia, 1988.
- [173] P. White. Use of Markov Probability and Reliability Model Generation Methods in the Analysis of Reliability of a Fault Tolerant, Hardware and Software Based System. In *AGARD 493*. NATO, 1993.
- [174] D. Whitely. A Genetic Algorithm Tutorial. Technical Report CS-93-103, Department of Comp. Sci., Colorado State University, 1993.
- [175] D. Whitely and J. Kauth. GENITOR: A Different Genetic Algorithm. In *Proc. Rocky Mountain Conf. on AI*, pages 118–130, 1988.

- [176] K. Wong. A New Framework for Part Failure-Rate Prediction Models. *IEEE Transactions on Reliability*, 44(1):139–145, March 1995.
- [177] F. Xia, S. Velastin, and A.C. Davies. Evaluation of the Data Interaction Architecture Demonstrator by Means of a Multiple Mobile Robot Workspace Simulation. *Microproc. & Microsyst.*, 19(1):23–33, 1995.

Appendix A

Implementation of X-Topmeter & X-Alloc

A.1 Reliability Model Library

The reliability models employed in X-Topmeter are presented below. In Section A.2 an example of a service reliability model for a topology that employs the *standbyu2* and *parallel* models is presented. This library of models may be extended or changed by the designer as required. For instance, in CABV1 the sensing tasks for Services 1 and 5 have been amended to use a kofn model, as four wheel sensors are employed in these tasks.

Model	Name	Description
0	simple(FR1)	Single component model
1	standbyu(FR1,FR2, FR3)	Redundancy with 2 units of possibly unequal failure rates & software resource to detect failure of task
2	activeu(FR1,FR2)	Redundancy with 2 active components with possibly unequal failure rates and no decision mechanism. First to arrive is taken
3	accept1(FR1,FR2, FR3)	2 components in parallel and an acceptance test of first to arrive
4	majn(numdefaults, numspares, FR1,FR2,FR3)	Copies in parallel provide input to a voter.
5	parallel1 (numdefaults, numspares, FR1, FR2, FR3)	Parallel RBD model. First result to emerge used
6	standbyu2 (numdefaults, numspares, FR1, FR2, FR3)	Extension of standbyu for n standby copies of two resources
7	acceptn(numdefaults, numspares, FR1,FR2,FR3)	Version of accept1 in which two resources are used and each copy has an acceptance mechanism in series.
8	simplex (FR1,FRS1)	2 state Markov model: 1 copy of a resource
9	wtmr (numdef, numspares, coverage1, coverage2, FR1, FR2, FR3, FRS1, FRS2, FRS3)	Markovian TMR model for 3 copies which may be from different resources
10	standsh (numdef, numspares, covdef, covsp1, lh0, lh1, lsd, lsp1)	Markovian standby sparing model. Active replication. Software allowed.

Model	Name	Description
11	twintmr (numdef, numsp, covdef, covsp1, lh0, lh1, lsd, lsp1)	Markovian TMR model for 3 copies of each resource. 2 single resource TMR models are placed in parallel in a RBD model. First result to arrive taken
12	parallel2 (numdef, numsp1, numsp2, FR1, FR2, FR3)	Three separate resources in parallel. Each resource can have any number of copies of itself in parallel.
13	standbyu3 (numdef, numsp1, numsp2, FR1, FR2, FR3, FR4)	n standby copies of three resources. Cold Spared. Each copy has an acceptance test in series.
14	accept3 (numdef, numsp1, numsp2, FR1, FR2, FR3, FR4)	Three resources employed in parallel. Hot spared. Each copy has an acceptance test.
15	maj3 (numdef, numsp1, numsp2, FR1, FR2, FR3, FR4)	Three resources each with multiple copies and a majority voter. First to produce acceptable value is used.
16	threetmr	Markovian TMR consisting of 3 single resource TMR models placed in parallel in a RBD model. First result to arrive taken
17	binomial	RBD model employing k-out-of-n approach. Used if $numop > 1$
18	nmr (numop, numdef, numsp1, numsp2, l0, l1, l2, l3, accept) ¹	N-Modular Redundancy model which employs kofn and a voter. no_op gives k. General mechanism for hardware.

A.2 Reliability Model Evaluation

The SHARPE tool is used to predict the MTTF of services in a SC-RT system. Each service reliability model is generated using a failure rate data library of individual resources and the architectural topology employed for each task. An example SHARPE input file is presented below for one proposed topology for service 1 of the CABV1 case study. This example employs the sensor results from service 0. Therefore there are four task reliability models. There are also 3 network models. The service model is hierarchical in that it uses the failure distribution results of these tasks and networks to produce a MTTF measure for the overall service.

Task 0

```

block Fs3
comp a3 exp(.0001)
comp b3 exp(.000067)
comp c3 exp(.000025)
parallel top a3 a3 b3 b3 c3 c3
end
poly ep3() cdf (Fs3)

```

Task4

```

block Fs5
comp a5 exp(.0000046)
comp b5 exp(.0000077)
comp c5 exp(.000011)
parallel top a5 a5 b5 b5 c5 c5
end
poly ep5() cdf (Fs5)

```

Network 0

```

block net0
comp x0 exp(.00006)
comp y0 exp(.00006)
comp z0 exp(.00006)
parallel top x0 x0 y0 y0 z0 z0
end
poly epnet0() cdf(net0)

```

block service1

```

comp ax0 ep3()
comp ax1 ep4()
comp ax2 ep5()
comp ax3 ep0()
comp bx0 epnet0()
comp bx1 epnet1()
comp bx2 epnet2()
series joint0 ax0 ax3
series top joint0 bx0 ax1 bx1 bx2 ax2
end
expr mean(service1)
end

```

Task3

```

block F4
comp a4 exp(.0000023)
comp b4 exp(.0000020)
parallel top a4 a4 a4 b4 b4 b4
end
poly ep4() cdf (F4)

```

Task5

```

block standbyu20
comp a0 exp(.000040)
comp b0 exp(.000080)
comp c0 exp(.00000045)
series d1 c0 a0
series d2 c0 c0 b0
series d3 c0 c0 c0 b0
parallel top a0 d1 d2 d3
end
poly ep0() cdf (standbyu20)

```

Network 1

```

block net1
comp x1 exp(.000060)
parallel top x1 x1
end
poly epnet1() cdf(net1)

```

Network 2

```

block net2
comp x2 exp(.000060)
parallel top x2 x2
end
poly epnet2() cdf(net2)

```

The service reliability model is a RBD containing six components in series: the joint sensing task sub-model, the processing task sub-model, the action task sub-model and the 3 network sub-models. The failure distribution for the components of the service

model are calculated from evaluation of the task and network sub-models. The failure rates of the components of the sub-models are calculated using failure rate data from the resource data library and the appropriate coverage factor. The service model is evaluated by SHARPE to produce a MTTF of 13100 hours.

A.3 Data Structures for X-Topmeter and X-Alloc

In Table A.1 the data structures employed in X-Topmeter and X-Alloc to represent a library of hardware resources is presented. Four categories of hardware resource may be employed: sensor(0), processor(1), actuator(2) and bus(3). The coverage factor employed is determined by the decision mechanism used by a given task.

Name	Description
int id;	Unique identifier for hardware
int type;	Category 0-3
double FR;	Resource failure probability per second
double coverage[3];	Probability of correctly detecting failure
int speed;	Units per second e.g. number of instructions
double cost[2];	Ownership cost for 1st unit, % marginal cost of extra units
double utilisation;	Total utilisation of hardware of category 1 or 3.
int number;	Number of copies of this resource employed
int capacity;	Max number of sub-tasks, number of buffered messages
int memc;	Memory capacity for processors, bandwidth per second for buses

Table A.1: Hardware Data Structure

In Table A.2 software resource data is presented. Two variants and one default software are indicated for each software resource. Three categories of software resource have been identified: logical(0), variant(1) and decision(2). This allows the designer to employ up to N(=3)-modular redundancy if desired.

Name	Description
int id;	Unique identifier for software resource
int type;	Software category: 0-2
double FR;	Predicted or estimated failure rate
int variants[MAXVAR];	Variant component
int C;	Computation time of component in instructions
int periodic;	0=no 1=yes
double T;	Minimum inter-arrival time, in seconds
int setP[MAXHARDWARE];	Processor residency restrictions, -1 = non
int cost;	Ownership cost of this sware resource

Table A.2: Software Data Structure

Tasks are the primary structuring elements of an architectural topology. The service and system topologies are formed from the set of task topologies. A task structure contains data on whether software is to be employed and the set of hardware resources that may be employed to implement each task. Four types of tasks are used: sensing(0), processing(1), communication(2) and action(3).

Name	Description
int id;	Unique identifier for task
int type;	Type
int hardware;	Default hardware resource
int no_hardware;	Number of default hardware employed
int software;	Default software resource employed, none =-1
int no_spare[2];	Number of extra hardware resources employed
int spare[2];	Ids for spares
int no_variants[2];	Number of extra software units employed
int variants[2];	Software variants used in this task
int operational;	Number of seconds task must remain operational
double mttf;	Predicted mean time to failure
double WCRT;	Predicted worst case response time
int service;	Service task belongs to
int mes_total[6];	Total bytes of messages sent from task
int no_bits;	Number of bits/integers to represent task in chromosome
int tabu[MAXRES];	Next iteration when restrictions are lifted
int trans[3][MAXTRANS];	Strings of precedence constrained sub-tasks
int met;	Binary indicating whether WCRT requirement is met

Table A.3: Task Data Structure

Since no specific allocation of sub-tasks to processor units is made during an X-Topmeter experiment the proportion of inter-processor communication is unknown. A rate of 50% inter-processor traffic is budgeted for. This proportion can be adjusted as the design progresses. Each network is characterised by six bits or 3 integers. At present only two network arrangements are used: three networks and a single IMA network.

Name	Description
int id;	Unique identifier for network
int type;	Category of network
int hardware;	Default bus resource employed
int no_hardware;	Number of default buses employed
int no_spares[2];	Number of extra bus resources employed
int spares[2];	Ids for spares
int model;	Reliability model employed on this network
int operational;	Seconds network must remain operational
double mttf;	Mean time to failure
int WCRT;	Worst case response time of network in secs
double no_bytes[3];	Bytes per second that needs to be carried
int no_bits;	Number of bits/ integers to represent in gene
int tabu[MAXRES];	Iteration when restrictions lifted

Table A.4: Network Data Structure

A service is the highest level of system abstraction employed in the X-Topmeter and X-Alloc tools. A service provides a set of control actions within specified WCRT and reliability targets.

Name	Description
int id;	Unique identifier for a service
int no_tasks;	Number of tasks in service
double T;	Period of a service
double D;	Deadline of a service
int operational;	Seconds service must remain available
int tasks[3];	Set of tasks in service
int no_hardware[MAXHARDWARE];	Min number of each type of hardware
double mttf;	Mean time to failure of service
int joint[MAXTASKS];	Data shared with other tasks?

Table A.5: Service Data Structure

A set of weighting factors are employed by X-Alloc to determine the relative importance of meeting each allocation goal and constraint. The parameters also guide the search process as they are used by the allocation evaluation function to determine the quality of any proposed allocation. Eleven weighting factors are required:

double kparallel;	Factor for parallel sub-tasks
double knumwhare;	Factor for size of platform
double kmemp;	Factor for processor memory utilisation
double kmemb;	Factor for bus memory utilisation
double knumesp;	Factor for number of messages on one processor
double knumes;	Factor for number of messages
double ktransmes;	Factor for max bytes of messages in task constraint
double kwertt;	Factor for tasks failing to meet WCRT targets
double ktwcr;	Factor for total WCRT of tasks in the system
double kwrtsp;	Factor for messages failing to meet WCRT targets
double knumpstp;	Factor for number of sub-tasks per processor

Sub-tasks represent the set of software units to be allocated to specified (by X-Topmeter) processing resources. Sub-task attributes are:

Name	Description
int id;	id of sub-task
int taskid;	id of parent task
int host[3];	id of [resource, proc,no] sub-task is resident on
int setP[MAXPROC];	Set of processors to choose from
int message[6];	Data, network, bus, receiving sub-task, no, hardware
int memu;	Processor memory usage by sub-task
int priority;	Priority of this sub-task
int C;	Computation time in instructions
int WCET;	C in millisecs Based on speed of proc
int T;	Period of sub-task in milliseconds (ms)
int B;	Blocking of sub-task in ms
int deadline;	Deadline for sub-task in ms
int O;	Offset for sub-task
int jitter;	Jitter experienced by sub-task in ms
int WCRT;	Worst case response time in ms
int shared[MAXSUBS];	Set of predecessor messages. none=-1
int parallel[MAXSUBS];	Array of sub-tasks in parallel with sub-task
int met;	Has sub-task met its deadline?

Table A.6: Sub-task Data Structure

Processor structures represent individual processing units to which a sub-task may be allocated. Each processing resource may spawn a number of such units. The elements of a processor structure are:

Name	Description
int id;	id for processor
int hwareid;	id of parent processor resource
int memu;	Memory usage by resident sub-tasks
int maxmes;	Maximum number of messages
int numsubtasks;	Number of sub-tasks resident on this processor
int resident[MAXSUBS];	Resident sub-tasks
struct pst *primap[MAXSUBS];	Priority ordered table of pointers
int met;	Has sub-task met deadline?

Table A.7: Processor Data Structure

A message structure indicates the attributes of messages that must be transmitted between sub-tasks either on a bus (for sub-tasks on separate processors), or via local data areas (for sub-tasks on the same processor). The attributes of messages are:

Name	Description
int id;	id of message
int priority;	Priority of message
int bus;	Bus message is resident on
int setB[MAXBUS];	Set of buses to choose from
int no;	Number of buses to choose from
int parallel[MAXMES];	Array of messages in parallel with this message
int size;	Bytes: number of packets can be calculated later
int source;	Index into sub-tasks
int dest;	Index into sub-tasks
int deadline;	Deadline for message
int jitter;	Jitter experienced by the message in millisecs
int B;	Blocking for messages
int WCET;	WCET in millisecs for message on resident bus
int WCRT;	Worst case response time in millisecs
int O;	Offset for message
int T;	Period of message
int met;	Met deadline?

Table A.8: Message Data Structure

Finally, a bus structure indicates the attributes of the bus units that may be used in the hardware platform. A number of units per bus resource may be employed. The attributes of a bus structure are:

Name	Description
int id;	id for bus
int hwareid;	id of parent hardware resource
double memu;	Bandwidth usage by resident messages
int maxs;	Max number of messages at each processor
int numes;	Number of messages on this bus
int resident[MAXSUBS];	Resident messages
struct message *primap[MAXSUBS];	Priority ordered table of pointers

Table A.9: Bus Data Structure

A.4 Data on Resources Employed by X-Topmeter

Resource Data for Illustrative Example 3

Id	Type	WCET	FR	Coverage	Speed	Cost
0	sensor	-	.000031	0.642, 0.707, 0.925	107000	527, .006
1	sensor	-	.000046	0.667, 0.767, 0.969	166000	785, .009
2	sensor	-	.00019	0.674, 0.759, 0.945	187000	838, .048
3	processor	-	.00017	0.610, 0.721, 0.963	137000	978, .045
4	processor	-	.00014	0.651, 0.717, 0.984	152000	195, .016
5	processor	-	.00028	0.670, 0.728, 0.973	128000	434, .015
6	bus	-	.000076	0.681, 0.744, 0.914	1030000	501, .029
7	bus	-	.00013	0.658, 0.742, 0.995	1023000	454, .016
8	bus	-	.0000045	0.629, 0.702, 0.935	1047000	318, .029
9	actuator	-	.0000082	0.622, 0.736, 0.945	116000	631, .0073
10	actuator	-	.000025	0.659, 0.753, 0.900	186000	279, .013
11	actuator	-	.000023	0.648, 0.783, 0.955	136000	570, .053
12	software	77	.000018	-	-	1616
13	software	188	.000010	-	-	1962
14	software	100	.000014	-	-	1072
15	variant	83	.000015	-	-	1720
16	variant	199	.000010	-	-	1865
17	variant	206	.000010	-	-	2121

Resource Data for CAB Case Study

id	Failure rate	Coverage	Speed	Cost
0	.00002	.4,.5,.6	-1	10.40,.01
1	.00004	.4,.5,.6	-1	10.15,.01
2	.00006	.4,.5,.6	-1	11,.01
3	.00003	.3,.5,.7	-1	45,.04
4	.00002	.3,.5,.8	-1	64,.06
5	.00001	.4,.5,.8	-1	68,.09
6	.0000017	.4,.5,.8	-1	20,.01
7	.000001	.4,.5,.8	-1	30,.01
8	.000005	.4,.5,.8	-1	40,.01
9	.000012	.4,.5,.7	-1	4.35,.04
10	.000016	.4,.5,.8	-1	5.79,.04
11	.00002	.45,.55,.75	-1	6.32,.04
12	.0000015	.6,-1,-1	-1	3,.01
13	.00001	.6,-1,-1	-1	4,.01
14	.00002	.6,-1,-1	-1	5,.01
15	.0000025	.7,-1,-1	-1	7,.01
16	.000002	.7,-1,-1	-1	8,.01
17	.000003	.7,-1,-1	-1	9,.01
18	.000003	.65,-1,-1	-1	60,.04
19	.000005	.65,-1,-1	-1	68,.04
20	.000007	.65,-1,-1	-1	76,.04
21	.00008	.65,.75,.9	2000000	35,.02
22	.000085	.6,.75,.9	2000000	40,.02
23	.0000657	.6,.7,.85	1600000	35,.02
24	.000005	.5,.65,.9	110000	50,.02
25	.000005	.5,.7,.9	125000	60,.02
26	.000005	.55,.75,.85	130000	50,.02
27	0.0	1,1,1	-1	0,0

Appendix B: Satisfiability

B.1 Architectural Topology Problem as SAT

Satisfiability problems are known to be NP-complete decision problems. The mathematical formulation for the architectural topology problem as a satisfiability problem is

Minimise $F(x) = \sum_{cl \in CL} Clause_{cl}$		-Feasibility goal
subject to		Clauses:
$\min mttf_s - mttf_s \leq 0$	$1 \leq s \leq S$	Dependability
$\sum_{u \in U} cost_u - \max cost \leq 0$		Cost
$\sum_{u \in U} num_u - \max num_u \leq 0$		Size
$num_u - \max num_u \leq 0$	$1 \leq u \leq U$	Size
$\sum_{c \in C_t} x_{ct} = 1$	$1 \leq t \leq T$	Component-Task
$\sum_{c \in C_n} x_{cn} = 1$	$1 \leq t \leq T$	Component-Network
$\forall r \in R_{(ct)} x_{(ct)} \geq 1$	$1 \leq t \leq T$	Resource-component-task
$\forall r \in R_{(cn)} x_{(cn)} \geq 1$	$1 \leq n \leq N$	Resource-component-network

where

$F(x)$	number of unsatisfied clauses for assignment x
x	Set of proposed assignments, $x = [x_{ij}]$
cl	clause
$Clause_{cl}$	0 if clause cl satisfied, 1 otherwise.
u	unit: smallest indivisible item, such as processor, bus, or sensor.
c	architectural component
n	network
r	resource. That is type of unit that may be selected for a topology.
s	logical control action to be provided by the system.
$cost_u$	ownership cost of unit u
$\max cost$	maximum cost permitted for the system
$depend_s$	measure of the dependability of service s
$\min depend_s$	minimum level of dependability required
$\max num_u$	maximum number of units allowed in the system.
$\max num_u$	max allowable number of unit u in the topology
num_u	number of unit u used in the system.