Trade-offs in System of Systems Acquisition

Frank Royston Burton

Doctor of Engineering
University of York
Computer Science
September 2014

Abstract

Large organisations tend to have multiple organisational goals. Example goals for organisations that perform search and rescue might be being able to search large areas quickly, and to provide, for the speedy recovery of survivors. To satisfy these goals, organisations will acquire different resources such as new systems, training programmes, infrastructure and processes. These different resources when combined to meet the same organisational goals, can be considered as a *System of Systems* (SoS). Organisational goals can be satisfied by completely different resource combinations with each resource combination satisfying the individual goals to varying degrees and with different overall costs.

Since organisations only have limited resources available to them, there is an incentive for organisations to find the most efficient resource combinations to satisfy their goals. This can be considered as performing trade-offs in SoS acquisition.

There are several open research gaps in performing trade-offs in SoS acquisition. The first is that the resources involved are heterogeneous. How do you compare the benefits of new equipment against new training programmes or organisational structures? The second is the multi-objective nature of the problem with the different organisational goals competing for the same limited budget. The third is managing the problem through-life and maintaining the satisfaction of organisational goals as old system retire and new systems come into service.

This thesis presents a model-based technique (with prototype tool support) that combines techniques from the fields of through life capability management, goal modelling, search-based software engineering and model-driven engineering. This technique addresses the three problems stated above allowing decision makers to more efficiently consider the trade-offs involved when performing SoS acquisition.

The technique has been evaluated on a realistic case study and on a standard problem found in the field of search-based software engineering.

Contents

List of Figures

List of Tables

Acknowledgements

Firstly, I would like to thank my two academic supervisors Richard Paige and Simon Poulding for their provision of support, advice, guidance, feedback and 'word smithing' throughout the EngD programme. I'm grateful to my two industrial supervisors Simon Smith & Dick Whittington for provided me with such an interesting research topic to work on.

I would like to thank my internal examiner Rob Alexander for providing general guidance on shaping the research and tales of experience from his PhD and thank my external examiner Tony Clark for suggesting improvements to the thesis. I would also like to thank my family and friends for their continuing support.

Lastly, I appreciate the organisation of the Large Scale Complex IT System programme for allowing me to meet such talented researchers from across the country and further abroad.

Author Declaration

Except where stated, all of the work contained in this thesis represents the original contribution of the author.

Parts of the work described in this thesis have been previously published by the author in:

- F. Burton, R. Paige, S. Poulding, and S. Smith, "System of systems acquisition trade-offs," in 2014 Conference on Systems Engineering Research (CSER 2014), Mar. 2014. [1]
- F. R. Burton and S. Poulding, "Complementing metaheuristic search with higher abstraction techniques," Proc. 1st International Workshop on Combining Modelling and Search-Based Software Engineering, 2013. [2]
- F. R. Burton, R. F. Paige, L. M. Rose, D. S. Kolovos, S. Poulding, and S. Smith, "Solving acquisition problems using model-driven engineering," in Modelling Foundations and Applications, pp. 428 - 443, Springer, 2012. [3]

Parts of the work described in this thesis have been previously published with the author acting as a secondary contributor in:

- J. R. Williams, F. R. Burton, R. F. Paige, and F. A. Polack, "Sensitivity analysis in model-driven engineering," in Model Driven Engineering Languages and Systems, pp. 743 - 758, Springer, 2012. [4]

The author provided as a secondary contribution to the paper, the Capability Acquisition Technique with Multi-objective Search (CATMOS) prototype tool, the application of the CATMOS prototype tool to the Airport Crisis Management Scenario and assistance in both using the CATMOS prototype tool and in interpreting results generated by the primary author's sensitivity analysis.

- R. F. Paige, P. J. Brooke, X. Ge, C. D. Power, F. R. Burton, and S. Poulding, "Revealing complexity through domain-specific modelling and analysis," in Large-Scale Complex IT Systems. Development, Operation and Management, pp. 251 - 265, Springer, 2012. [5]

The Through Life Capability Management section of the paper summarises some of the original research shown in this thesis done by the author.

Chapter 1

Introduction

1.1 Motivation

Large organisations tend to have multiple organisational goals. For example, an organisation that performs search and rescue might want to be able to search large areas quickly, and provide speedy recovery to survivors. To satisfy such goals, organisations will acquire different resources (such as new systems, training programmes, infrastructure and processes) that when combined may go some way towards achieving them.

For large organisations, a set of goals may be satisfied - to varying degrees - by any one of a large number of potential resource combinations. This makes reasoning about the optimal ways for large organisations to meet their goals, via the acquisition of resources, a difficult and arduous task. This tends to result in high-level acquisition trade-offs being neglected for being too difficult, or with only a few of the full range of options being considered. It is also very unlikely that all of an organisation's goals will be fully satisfied by any one solution, due to factors such as budgetary limitations, unavailability of solutions and general problem complexity, meaning that in practice trade-offs are necessary. This thesis presents a model-based technique for allowing organisations to reason about the trade-offs involved during such acquisitions to enable more efficient decision-making.

Acquisition problems are made difficult by a number of characteristics. One of these is that the resources involved can be heterogeneous. For example, new equipment, new training programmes, new organisational structures, new infrastructure, etc. may all be part of a solution for an acquisition problem. To perform high-level acquisition trade-offs, comparisons needs to be made between these different types of resources, which traditionally are considered to be incomparable. How does one compare the benefits of a new piece of equipment against the benefits of using a new organisation structure? Another characteristic is the management of the multi-objective nature of the problem, with different organisational goals competing with each other over the limited acquisition resources. How does one support decision makers in making trade-offs between competing organisational goals? There is also a temporal aspect of the problem with the overall satisfaction of the organisational goals needing to be maintained through-life, as the individual systems in the organisations are retired and new systems introduced.

This thesis is motivated by two sources. The first source is the Large-scale Complex IT System programme [6]. The aim of this programme is to provide insight into the problems faced by Large-scale Complex IT Systems (section 1.1.2) development. This thesis deals with the acquisition of Large-scale Complex IT Systems, which due to the underlying nature of the problem is similar to the acquisition of Large-scale Complex Systems in general. The second source is the UK Ministry of Defence's transition to Through Life Capability Management (TLCM) (section 1.1.1) [7], an acquisition technique designed to deal with Large-scale Complex System procurement [8]. With the UK Ministry of Defence's push towards Network Enabled Capability (NEC) [9, 10] it is intending to use TLCM as an enabler for Large-scale Complex IT System acquisition [9–12].

The thesis presents a technique that deals with multiple competing organisational goals, evaluating trade-offs in acquisition over limited resources, the heterogeneous nature of the involved resources and scheduling acquisitions through life. This will be discussed further in the following sections.

1.1.1 Through Life Capability Management

The UK Ministry of Defence has recently switched to *Through Life Capability Management (TLCM)* for managing its military acquisitions [13]. Previously, the UK Ministry of Defence used *Through Life Management (TLM)* for managing its military acquisitions [13]. Before discussing TLCM, we consider both TLM and the issues that lead to the transition.

Through Life Management (TLM) involves the periodical updating of all equipment/platforms in the UK Ministry of Defence [13]. When a piece of equipment/platform nears the end of its lifespan, TLM ensures that a new replacement equipment/platform will be acquired on time to replace it [13]. The limitation of this technique is that it is heavily equipment focused; this has led to serious and high profile problems for the UK Ministry of Defence [13]. A recent example is the acquisition of the Apache helicopter fleet, which was acquired on time to replace the existing helicopter fleet. However, the supporting training programmes for teaching pilots how to fly the Apache helicopter were delivered late [7,14]. This resulted in a large number of the newly acquired helicopters being grounded. The UK Ministry of Defence needed to pay to maintain the existing helicopter fleet in service and pay again to keep the Apache helicopters in storage, leading to £19.9 million in additional costs [7]. Similar problems have occurred in the delivery of the Bowman radio system and the nuclear submarines jetties where either the proper training or infrastructure was not provided [7].

This has led to growing recognition within the UK defence community that acquisition should be carried out with respect to military capabilities [15]. Military capabilities are the abilities that the acquired assets of the UK Ministry of Defence provide to front line military commanders [7]. Military capabilities are created from compositions of heterogeneous resources, categorised by the Defence Lines of Development (DLoD) [12, 16]. The Defence Lines of Development are Training, Equipment, Personnel, Information, Doctrine and Concepts, Organisational Structures, Infrastructure and Logistics [17]. The intention of TLCM is to deal with the same issues as TLM but to consider them across all of the DLoD [7].

The transition to TLCM has the potential to enable new opportunities. One of these comes from acquisition in terms of capability rather than equipment, which in turn allows consideration of alternative solutions to military problems. For example, previously, an acquisition problem could be expressed in terms of acquiring a scouting tank. The purpose of a scouting tank is to go to the front lines and scout the enemy positions so that other tanks can fire on their locations. When the acquisition is rewritten in terms of capability, this becomes not the acquisition of a scouting tank but the acquisition of a scouting capability. This allows the consideration of new types of solutions for providing the capability, specifically those that are *not* tanks. For example an Unmanned Aerial Vehicle could also be used to gather the necessary information for scouting at a cheaper price and without placing the life of the driver of the scouting tank in danger. [1]

The use of capabilities rather than equipment as the principal concept of TLCM allows consideration of high-level acquisition trade-offs. These types of high-level trade-offs at the system of systems level can lead to much better capabilities being acquired at potentially reduced cost. These types of high-level trade-offs are the focus of this thesis.

Before this opportunity can be taken advantage of, there are some unsolved problems to be addressed. In TLCM the heterogeneous resources that can be acquired to produce military capabilities are categorised by the *Defence Lines of Development (DLoD)* (section 2.3) [12, 16]. The relationship between the DLoD and military capabilities is known to be many to many [12]. That is to say that the same military capabilities can be produced from different combinations of resources and the same combinations of resources, give rise to multiple military capabilities. The actual relationship between the two - how to go from military capabilities to heterogeneous resources in the DLoD or vice versa - is undefined and is an unsolved problem within TLCM [12]. An implication of TLCM is that the heterogeneous resources from the DLoD are comparable in a meaningful objective way; this is to say that the acquisition of a new piece of equipment can be compared objectively to the acquisition of a new organisation structure or the acquisition of a new training programme, etc. By objective it is meant that there should be a structured method for performing the comparison rather than guess work. This is because all of the resources in the DLoDs can be purchased using the same resource (money) and all effect the produced capabilities. How to perform such a comparison is open research question and has been referred to as the 'Apples and Wednesdays' problem [18]. The name is reflective of the type of problem involved: how does one compare the benefits of having more apples to the benefits of having more Wednesdays? A major contribution of this thesis is the provision of a technique to partially address this problem.

Due to the many-to-many nature of the relationship between military capabilities and the DLoD, there can be multiple valid solutions in fulfilling any set of military capabilities. Each solution using different combinations of heterogeneous resources to achieve the same effect at different costs. Solutions can even achieve different fulfilment of the various military capabilities at different costs, leading to a complex trade-off space.

Another unsolved problem is within the through life *management* of capabilities. A problem that occurs within in the UK Military is that when components in the DLoD are retired from service and new components in the DLoD come into replace them, there can be capability gaps [16]. Capability gaps arise when military abilities are lost and certain military operations cannot be carried out until the replacement resources are fully acquired. This usually results in incurring costs to either maintain the old components or alternatively incurring costs to speed up the acquisition of new components [16].

Currently, partly due to the timescales of military acquisition (which can be in the decades),

[1]Private discussion at the Integrated Enterprise Architecture Conference 2010

whether the concept of TLCM is an improvement over the previous acquisition paradigm is still mostly unproven. However after the Enabling Acquisition Change Report [7] in 2006 on the ability of the UK MoD to adopt TLCM, work has now started on the adoption of TLCM by the Ministry of Defence [19].

1.1.2 Large-scale Complex IT Systems

The Large-scale Complex IT System research initiative [6, 20] studies the unique issues found in large-scale complex IT systems due to their scale and complexity. This thesis falls under this research programme. Though Life Capability Management (TLCM) is a technique for performing generic large-scale system acquisitions and through it has a military motivation, it is not military specific.

The UK Ministry of Defence is attempting to use TLCM for the purpose of acquiring new large-scale IT systems under its Network Enabled Capability (NEC). It is recognised that the IT systems cannot be gained in isolation from the other Defence Lines of Development [9, 11, 12]. At the system of systems level it is no longer sufficient to consider IT systems in isolation from the other components they interact with. Capability-based acquisition is a system of systems problem [12].

1.1.3 Relationship to Early Requirements Engineering

The capability management part of TLCM has an analogue within the field of goal driven requirements engineering.

A capability from TLCM is similar to the concept of a goal from goal driven requirements engineering. Both capabilities and goals are desired abilities and can be decomposed and satisfied by components. A technical difference is that a capability must have associated benchmark measurements [8] to describe how well a solution satisfies the capability; a goal *may* have associated benchmark measurements. For all intents and purposes they are very similar concepts, suggesting that capability is not military specific.

The similarity extends further. The National Air Traffic control in the UK, who are using capability-based acquisition, have introduced a "new" concept called a Hazard for their work [2]. The Hazard concept already exists in KAOS [21] goal modelling under the name of an Obstacle.

In capability-based acquisition, the capabilities are eventually satisfied by the acquiring things categorised by the DLoD. In goal driven requirement engineering, goals are similarly satisfied by acquiring things categorised by systems, processes and people. These categorisations have substantial overlap. Systems is equivalent to equipment and infrastructure from the DLoD, people is equivalent to people and training from the DLoD and processes is equivalent to doctrine, concepts and logistics from the DLoD. The choice of categories makes no significant difference; it is done merely to indicate the types of things that are likely to be acquirable in the acquisition solutions and help structure solutions.

There are multiple styles of goal modelling. The most similar approach to capability-based acquisition is KAOS [21] goal modelling, because it considers the goals to belong to the system

[2]Private discussion with National Air Traffic control

as a whole rather than to individual actors (like i* [22]). A major difference is that KAOS has formal underpinnings for verifying its goal models are internally consistent, whereas capability-based acquisition is supported by architecture frameworks such as MODAF [23] for verifying its solutions against the actual problem.

The high-level trade-offs being researched in this thesis correspond to the evaluation of alternative goal tree derivation paths in goal modelling. While alternative goal tree derivation paths has been recognised as a research area by Lamsweerde [24], the work that has been done in this area is minimal. As a consequence of solving the research gaps in terms of TLCM, the research will also be addressing most problems of alternative goal tree derivation in the field of goal modelling.

1.2 Formal Statement of the System of Systems Acquisition Trade-off Problem

This thesis is addressing problems found in the acquisition of system of systems, largely motivated by the challenges facing the UK Ministry of Defences Armed Forces. The Australian and Canadian militaries have similar acquisition programmes using similar concepts but with different terminologies [25, 26].

In this thesis, we will be considering the system of systems acquisition problem using the perspective of TLCM, which is a major approach being used to address system of systems acquisition in practice [7], and therefore encounters the major common problems of system of system acquisition.

A large organisation engages in various scenarios representing what it does or what it wants to do. From these scenarios, the abilities required to be able to perform the scenarios are called capabilities (section 2.2).

Capabilities can be satisfied in various ways that (for military acquisition) are categorised by the Defence Lines of Development (DLoD). The categories are: training, equipment, personnel, information, doctrine and concepts, organisation, infrastructure and logistics [17]. These DLoD come together to satisfy the capabilities to some degree. The relationship between the DLoD and capabilities is known to be many to many but the exact relationship is unknown [12].

The acquisition decision makers aim to increase the satisfaction of existing capabilities of the organisation or add new capabilities to the organisation as new needs arise. To do this, the acquisition decision makers have to decide what things, from the various DLoD, to acquire. Some of the desired things will be acquired in-house, and acquisition of other things will be contracted out to various vendors. The acquisition is likely to be brown-field, and as such the decision makers should consider if existing systems have the potential to be used together with new systems to produce the wanted capabilities when formulating their acquisition solutions.

There is a finite budget available for these acquisition programmes, which is received over time. As such, the various acquisition options will need to be considered and some trade-off between satisfying the various capabilities with respect to the limited resources will need to be established. For simplicity, we will refer to the things that can be acquired from the DLoD as components from now on. There are complex dependencies between the various components that can be acquired. Some components cannot be acquired without acquiring other components and some components need to work together with other components to provide the wanted capabilities. The acquisition decision makers need to decide which components to acquire and what trade-offs to make in doing so.

The acquisition decision makers also need to manage the acquisitions through life. This means considering how the acquisition of the chosen components will be scheduled considering budgetary limitations, dependencies between programmes and needs for having capabilities at different times. These considerations can affect the initial choices of which components to acquire. Changes in capability must be carefully managed to prevent *capability gaps* from forming, where certain capabilities cannot be used because systems have been retired and the replacement systems have not yet been introduced. Capability gaps may in some cases be acceptable; in other cases they need to be identified in advance so arrangements can be made to cover the gap; and in all cases it is desirable to be aware of when the capabilities gaps will happen to allow appropriate action to be taken.

The overall aim of the thesis is to provide techniques for helping decision makers address the problem of system of system acquisition trade-offs. The first research gap focuses on the missing link between the DLoD and capabilities, the second research gap deals with all the problems stated but without consideration of the time and scheduling, while the third research gap deals with adding time and scheduling to the technique.

The main challenges are as follows:

1. Going between the concrete concepts of the things in the DLoD to the very abstract notions of military capability.
2. Providing a technique that can manage the complex dependencies between the various DLoD.
3. Making trade-offs between satisfying the various capabilities whilst considering the complex dependencies between the various DLoD and also considering the budgetary limitations.
4. Handling the brownfield nature with existing systems already in place.
5. Attempting to manage these challenges not just in a single one-off acquisition but instead managing it in a continuous acquisition that takes place over long time periods.

Challenges 1, 2 & 4 are covered by research gap 1, challenges 1, 2, 3 & 4 are covered by research gap 2 and challenges 1, 2, 3, 4 & 5 are covered by research gap 3. The research gaps are presented below.

1.3 Research Gaps

There are three significant research gaps that this thesis will address:

1.3.1 Research Gap 1 - Bridging the gap between the Defence Lines of Development and Military Capabilities

There is currently no objective method that works for mapping the programmes in the Defence Lines of Development (DLoD) to the capabilities that they can produce. The current state of the art in the field is no more advanced than using weighted sum [18,25], which by trivial inspection is known to be incorrect. Therefore, this thesis will provide a working objective technique for mapping the Defence Lines of Development (DLoD) to capabilities. This is conceptually equivalent to solving the Apples and Wednesdays problem from the UK military acquisition community [18]: An example of this problem is whether it would be preferable to acquire a new piece of equipment or a new organisational structure. Both cost money to acquire, and when they have been acquired they will contribute to the overall military capability; and so it should be possible to decide which of the two is the best to acquire. Without first establishing the link between the DLoD and military capabilities, there is no apparent way to compare them because they are two completely different types of things.

1.3.2 Research Gap 2 - Multi-objective Acquisition Trade-offs

The second research gap is addressing the multi-objective nature present in this type of acquisition problem. The different stakeholders involved in the acquisition have different goals and there is only a limited amount of resources that can be used during the acquisition. This means in practice that some of the stakeholders' goals will not be fully satisfied by the acquisition. Our approach will not identify which stakeholder's goals should be met and which should be abandoned, but instead will support decision makers to effectively explore the various possible trade-offs between the individual stakeholder's goals and the overall costs involved.

1.3.3 Research Gap 3 - Scheduling Acquisitions Through Life

The third research gap is focusing on the through life part of TLCM. Capabilities need to be maintained over time and capability still needs to be maintained during the retirement of existing resources and the acquisition of new resources to replace them. This research gap addresses how the acquisitions should be scheduled over time and how to identify possible capability gaps before they occur.

A capability gap is where for a short period of time capabilities are no longer functional due to the retirement of existing resources, which have yet to be replaced [16]. There is a need for these to be identified in advance to allow decision makers the ability to either accept the loss of capability during that time window or to extend the lifespan of existing resources or to provide temporary replacement resources or to hurry acquisition of new resources to deal with the capability gap [16]. The technique presented in this thesis needs to provide support for allowing the acquisition decision makers to identify these situations in advance.

1.4 Research Objectives and Research Hypothesis

Large organisations acquire system of systems over time to fulfil their organisational goals. System of systems are by necessity composed out of smaller systems that can either be pre-existing within the organisation, off the shelf from vendors or custom made for the organisation's specific needs. Large organisations can choose from multiple options when selecting these systems to be included within their system of systems and there are multiple valid combinations of systems that can fulfil the same needs in different ways.

The specific combination of systems selected by an organisation for inclusion within their system of systems affects the levels to which their organisation needs are fulfilled. It also affects the total cost that the organisation incurs in meeting its needs.

This is further complicated by the temporal aspects of system of systems acquisition, with systems composing the system of systems retiring and coming into active service through life changing the satisfaction of the organisational needs over time. It is additionally complicated by the heterogeneous nature of these systems as illustrated by the DLoD (section 2.3).

The aim of this thesis is to provide techniques (with prototype tool support) for decision makers to be able to explore and manage the trade-off space between the satisfaction of the different organisation goals and the costs incurred, whilst also considering the through life element of the problem. Additionally, the technique is validated against a realistic case study.

The research hypothesis of this thesis is: *The high-level trade-off decision space during system of systems acquisition can be effectively explored using a technique that generates an approximation of the Pareto front of the fulfilment of the various organisational objectives against the resources used in the context of acquiring systems for the system of systems.*

Each point on the generated approximation of the Pareto front is supported by a goal model that presents the corresponding acquisition plan in an objective and justified manner to the acquisition decision makers. For the context of this thesis, *objective* means that a structured method has been used and *justified* means that there is an argument for making the claim that the organisational goals have been satisfied by the acquired systems.

The technique draws upon research done in the fields of goal modelling, search based software engineering and model-driven engineering. The technique also supports the scheduling of acquisitions over time.

A Pareto front is a well-established concept in the field of economics (created by Vilfredo Pareto). A Pareto front is the set of solutions that are Pareto optimal. A solution is Pareto optimal when there is no other solution that is strictly better than it for all objectives. When performing trade-offs only the solutions on the Pareto front need to be considered as all other solutions are worse than a solution that is on the Pareto front. Pareto optimally is further explained in section 8.1.

1.5 Capability Acquisition Technique with Multi-Objective Search (CATMOS)

This thesis presents a new technique called Capability Acquisition Technique with Multi-Objective Search (CATMOS). CATMOS takes in a description of a system of systems acquisition problem in the form of a top-level goal model and takes in descriptions of the possible satisfying components; both existing and acquirable. The descriptions of the possible satisfying components contain the capabilities the components provide, the component's dependencies on other components and the component's costs to acquire. This can be further annotated with scheduling information on when the component is in-service for or how long it will take to acquire and how long it will be in service for after it has been acquired and when the costs for the component will need to be paid and if there are maintenance costs that need to be considered.

Using the basic annotations CATMOS produces an approximation of the Pareto front of the desired capabilities against the costs. Each point on the Pareto front is supported by a goal model that shows the acquisition plan for that trade-off, the components involved and how they work together to satisfy the desired capabilities. An approximation of the Pareto front is produced since it is computationally infeasible to produce the actual Pareto front on non-trivial problems. This part of the technique is used for satisfying research gaps 1 & 2.

With the further annotations, CATMOS can schedule the acquisition plans and take account of whether the plans satisfy the desired capabilities during the time they are wanted in evaluating plans given both the budgetary and time limitations. When used like this CATMOS produces the Pareto front and goal model for each point like before but further produces for each acquisition plan a capability over time chart that shows how the satisfaction of the various capabilities varies over time and a Gantt chart showing when each component is acquired. This part of the technique is used for satisfying research gap 3.

The CATMOS technique is presented in chapters 7, 8 & 9. The basic ideas behind the technique are introduced in chapter 7, the application of multi-objective search is explained in chapter 8 and the through life scheduling of acquisition plans is explained in chapter 9.

1.6 Novel Contributions in this thesis

The research sits between the research fields of through life capability management, goal modelling, search-based software engineering and model-driven engineering. The research contributes a number of novelties to and between these fields including:

- *Alternative Goal Tree Derivation.* The research provides an effective way to perform alternative goal tree derivations. For a single acquisition problem, hundreds of thousands of alternative goal tree derivations corresponding to different solutions can be effectively explored. There is little existing research in the area of alternative goal tree derivation [24] and the research that does exist can only explore two or three alternative goal tree derivations at once.

- *Finding Trade-offs using Goal Models.* Following on from the alternative goal tree derivation contribution, there is no existing work that automatically finds the alternative goal tree derivations that represent the best trade-offs that the decision maker can make. This allows the decision maker to effectively explore the various high-level trade-offs that can be made.

- *Combining Goal Models with Multi-objective Search.* The research combines techniques from the acquisition community (goal modelling) with techniques from the search based software community (multi-objective search) to produce a novel technique that addresses the research gaps.

- *Through Life Scheduling Support on Goal Models.* The research provides a technique that allows the management of through life constraints and issues that appear in goal modelling when applied to the system of systems problem. For example, dealing with systems in the goal-model retiring and needing other new systems to be introduced to cover the capability gap. It also provides the ability to schedule dependencies both sequentially and in parallel with each other for acquisition and to consider through life costs against a limited budget.

- *Automatic Evaluation of Dynamic Goal models.* The technique presented in this thesis automatically generates alternative goal tree derivations and then these alternative goal trees are automatically evaluated. There is no existing work that attempts to automatically evaluate a dynamically generated goal tree. The closest work is by Letier & Lamsweerde [27] that assumes a single static goal tree structure and their evaluation is carried out manually.

- *Objectively bridging the gap between the Defence Lines of Development and capabilities.* The approach provides a valid working technique for translating between acquisition programmes within the Defence Lines of Development and the produced capabilities or vice versa. Previous objective techniques are known not to work (see section 4.4). Additionally, this helps address the 'Apple's and Wednesday's' problem, which is an open research problem [18].

- *Quantification of goal modelling agents.* In goal modelling, agents are derived to satisfy goals. In the technique presented in this thesis, this is further extended to allow multiple copies of the same agent to work together to satisfy goals that cannot be fulfilled by single agents alone.

- *Identifying the equivalent between Capability Based Acquisition and Goal Modelling.* One of the first contributions of this thesis was identifying that capability-based acquisition and goal modelling contain equivalent concepts and ideas under different terminologies.

The technique was applied to the simpler problem of the Multi-objective Next Release Problem (MONRP) as a case study. This was done in published work [3] by the author and is also shown in section 8.3. The technique was shown to offer a number of contributions to the MONRP. The full details of these are given in section 8.5 but in summary the main advantages of our technique (on the simpler problem of the MONRP) are:

- *Continuous Release Support.* Existing work on the MONRP considers either a single release or a few set release dates. Our technique includes support for handling continuous software releases. [3]

- *Visualisation.* An identified issue in the MONRP is to explain to the stakeholders why a found solution is good [28]. Our approach adds partial support for this by allowing the generated solutions to be visualised and presented to the stakeholders [3].
- *Continuous variable requirements.* Our approach to the MONRP allows the usage of continuous variable requirements in MONRP. [3]

1.7 Thesis Structure

The next chapter introduces some of the major concepts that are used during the thesis. Chapter three, problem discussion, covers the literature surrounding the problem of system of systems and military acquisition. The fourth chapter, current practice, covers literature on existing solutions that are being currently used to address this problem today. The fifth chapter, applicable research fields identifies relevant techniques that could be used to address the research gaps. The sixth chapter, explains the problem in more detail and shows an application of some the existing work in the field on the problem. The seventh chapter introduces a technique for bridging the gap between the DLoD and capabilities. The eighth chapter deals with performing multi-objective trade-offs and performs a case study on the MONRP. The ninth chapters deals with the through life aspect of the problem, performs a case study on a realistic military scenario and summarises how the presented technique relates to other research. The tenth chapter deals with the implementation of the technique's prototype tool. The last chapter evaluates the work, concludes, and provides insights to future paths of research.

1.8 Research Context

This thesis is for an Engineering Doctorate in Large-scale Complex IT Systems. The Doctorate of Engineering is similar to the Doctorate of Philosophy in that it contains a research component of the same length as the Doctorate of Philosophy. However the research component has an additional requirement to have an engineering focus in that it solves real problems found in industry.

Therefore this research project is being taken out under guidance from MooD International. The nature of solving real industrial problems almost always by necessity requires taking a cross disciplinary approach. Instead of contributing to a single academic field, the research performed in this thesis contributes to multiple academic fields and uses solving the industrial problem as the focus. The research fields used by this thesis are goal modelling, search-based software engineering, model-driven engineering and through life capability management.

This thesis has been funded by the MooD International and by the Engineering and Physical Sciences Research Council under grant number EP/F501374/1 as part of the Large-scale Complex IT Systems programme.

Chapter 2

Literature Review - Introduction to the Terminology

2.1 Introduction

This chapter will give definitions for some of the key concepts that will be used in the thesis. This chapter is only an introduction to the key concepts and aims to provide basic definitions to help the reader understand the key concepts. Most of the key concepts will be explained in more detail in subsequent chapters.

First the concept of capability is defined and described to show how it can be used as an abstraction. We then summarise the key implications for the use of the concept of military capability during acquisition. This is followed by an explanation of the DLoD and definitions for each of the lines of development. Then, Measures of Performance (MoP) and Measures of Effectiveness (MoE) and the differences between the two are explained. Lastly, the concept of a model is described for use later on in the thesis.

2.2 Capability

The Oxford Dictionary defines capability as:

> "*capability* noun (pl. capabilities) (often capability of doing/to do something) the power or ability to do something: he had an intuitive capability of bringing the best out in people — the company's capability to increase productivity.
>
> - (often capabilities) the extent of someone's or something's ability: the job is beyond my capabilities.
> - a facility on a computer for performing a specified task: a graphics capability.
> - forces or resources giving a country the ability to undertake a particular kind of military action: their nuclear weapons capability." [29]

The notion of capability being used in this research project is the "extent of someone's or something's ability" [29]. Capability is an interesting abstraction for system engineering. A simple way to explore this abstraction is to look at the graphics capability example from the definition.

The term graphics capability implies there is some way available to display graphics. The abstraction leaves two things undefined about the way to display graphics. Firstly, what is providing the capability is left out; in this case it could be the composition of several components of a computer, e.g. the graphics card, the motherboard, the software driver, the CPU, etc. However, none of these components will give a usable graphics capability on their own. Secondly, the intended application of the capability is not defined. In this case it may be to edit pictures, play 3D games, make movies, etc. An issue with this abstraction is how do we measure the quality of the capability we have. A hypothetical graphics card may be good at rendering 3D animations but bad at rendering 2D movies. This suggests that capability needs to be measured within the context of the capabilities intended use. Capability is also restricted by its environment because the components which make up the capability function only under certain constraints, so in this example the motherboard may only function between 0°c - 30°C. The other thing to note is that for a capability to be used it needs to have some embodiment, in this example the computer. In the case of military capabilities this is Force Elements, which are the military units that are assembled out of the things that are acquired [30].

2.3 Defence Lines of Development

The Defence Lines of Development (DLoD) are categories of components that when composed together produce military capability [12, 16]. The relationship between the components in the DLoD and the military capabilities is known to be many to many [12]. The importance of the DLoD is that they describe the full breadth of the things that may be acquired during system of systems acquisitions.

The UK MoD Defence Lines of Development (DLoD) [17] are, in summary:

- *Training* - The means to facilitate the practical learning of military doctrine.
- *Equipment* - Systems, platforms and weapons.
- *Personnel* - The supply of capable and motivated people.
- *Information* - Information is considered to be data when applied with context to a situation. Data is considered to be merely raw facts without meaning.
- *Doctrine and Concepts* - Doctrine are the principles used to guide military forces and includes the current methods for performing military activities. A concept is a new idea on how to perform a military activity in the future.
- *Organisation* - "Relates to the operational and non-operational organisational relationships of people. It typically includes military force structures, MOD civilian organisational structures and Defence contractors providing support." [17]
- *Infrastructure* - The management of permanent buildings, land and utilities.
- *Logistics* - The planning of and carrying out of the movement and maintenance of forces.
- *Interoperability* - Handling the interoperability issues that arise between the different DLoD.

Through there have been moves by some military contractors to have Industrial Readiness, which is the ability of industry to supply the wanted DLoDs to the MoD, also included in the DLoD; this has not been done [16].

The Australian Department of Defence, the American Department of Defence and the Canadian Department of National Defence have their own versions of the Defence Lines of Development, which are called the Fundamental Inputs to Capability [31], DOTLMPF [32] and PRICIE [26] respectively. While they are fairly similar to the DLoD they have slight differences in the way they have categorised the things that they acquire.

2.4 Measures of Performance

In the TLCM methodology, capabilities are considered to have attached measurements that can either be Measures of Performance (MoP) or Measures of Effectiveness (MoE) that allow the abstract capability to be treated in a concrete way [8].

A MoP is a measurement that can be taken on a system of interest directly. It is a measurement that is independent of any scenario the system may be used in. For example, the distance travelled by a car on a full fuel tank is a MoP. A military example is the maximum range at which an artillery piece can fire. MoP describe what systems are capable of doing in a quantifiable manner [33]. MoP are used in TLCM to describe the performance of systems independently of their performance in scenarios [8].

2.5 Measures of Effectiveness

A Measure of Effectiveness (MoE) is a measurement that is defined in terms of a scenario and measures how well a system performs within that scenario [33].

A military example would be the probability that the target fired at by the artillery piece will be successfully destroyed. This is different from a MoP as it is not a direct measurement of a property of a system. For example, an artillery piece can have range and destructive power as MoP and a MoE towards a ‘Long Range Fire’ capability of chance the target is destroyed. High MoP does not necessary imply high MoE whilst an artillery piece may have high range and destructive power if it has low accuracy it may still score badly on its MoE of destroying its target. MoE describe the extent to which a capability meets stakeholder needs [33]. MoE are used in TLCM to describe how well systems can meet the desired capabilities [8]. Both MoP and MoE are general system engineering terms rather than being TLCM specific.

2.6 Model

The word model has many possible meanings. The sense of the word model being discussed in this thesis can be defined as “a simplified description, of a system or process, to assist calculations and predictions” [29].

Later on in the thesis, we will be reviewing techniques that can be used to create models of systems and therefore it is important to know the distinction between a model of a system and the system itself. A model is an abstraction of the system and therefore only captures a subset of the details about system. Having a model of a system is a similar concept to having a map of a territory. A quote from Alfred Korzybski is that “A map of a territory is not the territory” [34]. The point being that the map can be inconsistent with the territory and because something will work according to the map does not mean it will work in the territory. The same applies to models of systems.

Another observation that can be made from the quote is that you can have multiple maps of the same area of land each with its own purpose. One map may contain the names of towns and cities build on the land and another may instead contain information on the height of the land. For a person trying to travel to a certain town the first map is useful and for a person who is attempting to climb a mountain the second map is useful. However a person who is trying to find a certain town will not find a height map of the land useful and a mountain climber will find limited usage in a map without the heights of the land marked. Since the map is not a full description of the territory it needs to be made with some purpose in mind. The same applies to the modelling of systems. This issue is considered in work on model validity for use in simulation in that a model is considered valid with respect to specific purposes [35].

In this thesis, modelling will be used both in the context of goal modelling and in the context of model-driven engineering. Goal modelling creates a structured argument that acquiring certain systems, people or process will satisfy an organisations goals [21]. A human expert can then manually check a goal model to make sure that the argument it presents is correct [21]. Model-driven engineering aims to use models instead of code for the development of software systems [36–38]. In this thesis, model-driven engineering techniques and tools are used for the manipulation of goal models.

2.7 Summary

In this chapter, we have briefly defined some of the key concepts that will be used in the thesis. In the next chapter, the literature review covers the research into the problem area being studied by the thesis.

Chapter 3

Literature Review - Problem Discussion

This chapter discusses the literature relating to the problem area of large-scale complex system acquisition. The purpose of this chapter is to help understand the considerations that need to be made whilst attempting to address the stated research gaps. The chapter begins with discussing military capability before moving on to TLCM. Then 'Wicked problems' by Rittel and Webber [39] are discussed. 'Wicked problems' are important because it describes the issues that face large-scale acquisition projects and any technique that is created to help address large-scale acquisition problems should be aware of them. Lastly, for completeness with respect to the TLCM motivations of the thesis, nature's equivalent of warfare, competition between species, will be briefly discussed to determine if there are any lessons that can be learned to aid in human warfare.

3.1 Military Capability

The UK Ministry of Defence is moving from equipment based acquisition to capability-based acquisition [19]; similar change has been made within the Australian Department of Defence [31]. This is because the previous acquisition techniques, which focused on renewing equipment, had various short comings due to their heavy equipment focus and ignoring the other DLoD that help make up military capabilities. This led to high profile failures in the Apache helicopter programme, Bowman radio system programme and the nuclear submarines jetties programme [7].

With military capability the things that compose the capability are left undefined, as is the intended use of the capability. This allows multiple and different solutions to be considered in attempts to meet the capability demands.

As with the definition of capability shown in the previous chapter, military capabilities are composed from different things, which are identified as the Defence Lines of Development (section 2.3). Recognition is also given to military capability being restricted by the environment in, which they operate [23]. For example, equipment can stop working in different terrains and climates. Military capabilities designed for use in the sea are unlikely to be effective in a land locked country and tanks designed for use on flat land are most likely ineffective on heavy mountainous terrain. Military capability is physically embodied in Force Elements At Readiness [12].

The Enabling Acquisition Change Report describes military capability needing to be made from the "most cost effective mix of components" [7], which suggests that cost effectiveness is part of the motivation for the adoption of capability-based planning. This is getting the most capability possible for the amount spent.

The more traditional use of the term military capability before its use in TLCM has been summarised by Newsome [40]. Traditionally the term was used interchangeably with the concept of military power. Newsome classifies the academic theories on military capability as either "power" theories or "dialectic" theories. In power theories, militaries have some level of power, which can be determined by looking at the outcomes of wars between military forces. In dialectic theories, started by Carl von Clausewitz [41] cited in [40], military capability is the amount of military resources available minus some "friction" that accounts for human factors and other non-material constraints. [40]

The military capability discusses by Newsome and Carl von Clausewitz is a very abstract and vague notion of the military's ability to project its power in warfare. Moving on to Through Life Capability Management, military capability is considered to be measurable via the use of Measures of Effectiveness [8]. These are real world measurements that can be taken from a system of interest in some scenario of interest [33]. For example the distance that an aircraft is able to drop supplies off to troops without having to refuel or land at an airstrip. These types of military capabilities are a lot more precise than the earlier uses of the term. Whilst Newsome and Carl von Clausewitz discuss military capability and Through Life Capability Management gives a way to measure it, how to actually obtain the military capability is left undefined.

3.2 Through Life Capability Management

Through Life Capability Management (TLCM) is a progression of Through Life Management (TLM), through the addition of Capability Management [7]. Through Life Management primarily focused on periodically updating in-place equipment to improve the armed forces [7]. TLCM extends this by considering all of the Defence Lines of Development (DLoD), rather than just equipment, and focusing on the capabilities given to the front line commanders that they can use during their operations [42]. Capabilities are usually created by multiple acquisitions over the various DLoD [42] (section 2.3) working together to create effective abilities that the front line commander can draw upon [12, 42].

The Acquisition Operating Framework gives the definition of Through Life Capability Managements as:

> "Through Life Capability Management (TLCM) interprets the requirements of Defence policy into an approved programme that delivers the required capabilities, through-life, across all Defence Lines of Development (DLoDs)." [8]

A similar definition is given by McKane:

> "TLCM is an approach to the acquisition and in-service management of military capability in which every aspect of new and existing military capability is planned and managed coherently across all Defence Lines of Development (DLOD) from cradle to grave." [7]

The Enabling Acquisition Change Report [7] is the major report introducing TLCM and discusses both its ideas and the practicalities of implementing them within the UK Military. The report justifies both the Through Life part of TLCM by stating that it is a common theme in reports on defence acquisition and justifies the capability part of TLCM by highlighting some of the high profile failures, which have occurred due to purely focusing on the equipment in their acquisition. The report [7] lists a number of possible advantages of adopting TLCM:

- A single person being able to control the funds for a capability, potentially ensuring a coherent delivery across the Defence Lines of Development. Previously, the different parts (equipment, training programmes, etc.), which work together to create the capability were controlled by different people.
- The enabling of trade offs between the Defence Lines of Development, potentially leading to better value for money solutions.
- Upfront investment to potentially gain cost savings in the operating and support costs.
- Opening up the ability to contract for capabilities.

The report also describes as a limitation of TLCM that it can be difficult to take the separately acquired capabilities and combine them back into force elements for deployment. The reports states that it is currently hard to create force elements in the Navy and Air force cases where force elements are based around ships and airplanes and even more difficult in the Army's case where force elements are based around people with much less well defined roles. The report argues that currently even though the ideas of TLCM conceptually has benefits, implementing them at the moment would be challenging. [7]

The Acquisition Operating Framework [8] gives a summary of the current Through Life Capability Management process. The process starts with Capability Planning.

In the first stage of Capability Planning, the Head of Capability takes the capabilities desired by the sponsor and identifies the characteristics of the capability along with making them measurable and solution independent. These are then assigned to Capability Management groups and Capability Planning groups and interdependences between groups are made clear. [8]

In the second stage of Capability Planning, the vague capability characteristics are decomposed into capabilities associated with tangible military effects. Metrics for measuring how well the military effect has been met are created along with benchmarks in terms of the metrics for the most demanding expected scenarios. [8]

In the third stage, five different perspectives are considered across the DLoD are considered. The capability perspective looks at the difference between the current and expected military capability given the current acquisition programmes. From this, the current acquisition plan is assessed against the desired military capabilities in stage 2 of the plan to establish the current surpluses and shortfalls. The research perspective looks at whether the current research programmes are aligned with capability goals. The industrial perspective looks at the major delivering companies and identifies problems in the market place. The financial perspective determines what the financial pressures and constraints are. The commercial perspective looks at existing contracts and brings to attention the MoD's commercial position. [8]

In the fourth stage of Capability planning, the second stage, which gives the goals, and the third stage, which gives the different perspectives, are compared and the shortfalls and opportunities are considered. This can led to the writing of formal options containing the implications of decisions made in Capability Planning. This can also start an investigation to find generic solutions across multiple capabilities. [8]

The final stage of Capability Planning, involves prioritising the created options. The next part of the process is Capability delivery that turns the accepted options into a cohesive programme and manages the delivery of it. [8]

The capability planning does not however ensure that what is delivered to the front line commanders is fully operational. Usually this is not the case and a smaller iterative-based acquisition process is needed to fill in the gaps of what was missed by the capability planning process. Some degree of iteration is likely to be necessary for any large-scale acquisitions.

In the UK MoD case this iteration is done via Urgent Operational Requirements (UORs) [43]. UORs are used when current operations find gaps in what is available to the front line. The main focus in the acquisition of UORs is delivering the requirements quickly hence parts of the acquisition process are dropped. They are not expected to completely integrate with the rest of the normal acquisition process and have a residual risk associated to them. [43]

Urgent operation requirements will most likely always be necessary. This is because the TLCM process relies on the Defence Planning Assumptions, which being assumptions can be incorrect and incomplete. Errors in assumptions led to there being gaps between what is acquired and what is required for the real operations undertaken.

3.3 Wicked Problems

Wicked problems were first described by Rittel and Webber [39] in 1973 in the context of planning problems. Wicked problems are a type of problem that are encountered in the real world. They tend to have a social nature and the problem tends to therefore have many different aspects belonging to many different people.

Before we go any further there is the question of what do social planning problems have to do with the field of defence acquisition or software engineering? One of the main reasons for building large-scale complex IT systems is the notion of satisfying complex social needs [44]. Both work on building large-scale IT systems [45] and on the problem of defence acquisition [46] have argued that their fields contain these wicked problems. This is likely to be because both have a social component to the needs they are addressing. Therefore it would be remiss of us not to look in the research field of wicked problems for guidance.

There is no real limit on how far the effects of a solution implemented in the real world can propagate through the social structure of the real world. This means that any scope placed on a Wicked problem may be missing key parts of the problem. To give an example of such propagation, it could be imagined that building a library in a small village may result in someone in the village going on to higher education who then goes on to make some ground breaking contribution in some research field. Generally these knock-on effects are not considered when considering whether to allocate funds to building a library in a village. Unfortunately the knock-on effects of implementing solutions to Wicked problems in the real world are not necessary good and they can lead to greater problems than the problem that was originally being addressed. Normally, problems being solved by the natural sciences have clearly defined boundaries [39]. Rittel and Webber [39] classifies these problems as tame problems.

A key difference between tame planning problems and Wicked problems is the formulation of goals. The difference is observed by a realisation that it is not the inputs to the design of the system or how the system was designed that matter but instead only the outputs or effects of the system that matter. This is a change from nouns “What is the system made of?” [39] to verbs “What does the system do?” [39]. This is clearly the change currently now being made in defence acquisition 30 years later with the move from Through Life Management to Through Life Capability Management. [39]

A second key difference between tame and wicked problems is the problem of identifying what the problem actually is [39]. This is because with the inability to place a boundary on the problem the possible solutions are endless and each of these solutions can lead to its own problems. The equivalent to this in the field of IT is that when a solution is implemented it immediately changes what the requirements are [45]. This is expected to become more common as the size of IT projects increase [45]. This is mostly likely because the larger the IT project, the more likely the IT project is trying to tackle some underlying social problem. Large IT projects are often acquired for forcing some organisational change [1].

In social planning there is a belief of unrestricted malleability by members of the general public in what can be done [39]. Unrestricted malleability is the view that anything can be done in solving a problem [39]. Views in a report on challenges in IT [47] also indicate that in IT projects customers believe that software can do anything. In the case of IT projects it should be pointed out that real constraints do exist, for example cost, computational limitations, lack of pre-written libraries for tasks increasing costs, lack of available research, etc [47]. The report states the limitations tend to be hard for customers to understand due to their abstract and multidimensional nature [47].

The defining characteristics of Wicked problems according to Rittel and Webber are [39]:

- “There is no definitive formulation of a wicked problem”
- “Wicked problems have no stopping rule”
- “Solutions to wicked problems are not true-or-false, but good-or-bad”
- “There is no immediate and no ultimate test of a solution to a wicked problem”
- “Every solution to a wicked problem is a “one-shot operation”; because there is no opportunity to learn by trial-and-error, every attempt counts significantly”
- “Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan”
- “Every wicked problem is essentially unique”
- “Every wicked problem can be considered to be a symptom of another problem”

[1]Private Discussion with Ian Sommerville at LSCITS Social Technical Systems Lecture

- "The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem's resolution"

- "The planner has no right to be wrong"

3.3.1 Proposed solutions

Roberts [48] gives three general coping strategies for Wicked problems:

- *Authoritative*, where a few stakeholders hold all the power and enforce their view of the problem and solution.

- *Competitive strategies*, where multiple organisations attempt to solve the problem separately using their own view of the problem and the solution. The organisations that defined the problem 'correctly' are successful and the organisations that defined the problem incorrectly tend to take heavy financial losses.

- *Collaborative strategies*, where the stakeholders come together and attempt to find common ground in a solution between them.

Competitive strategies have the benefit that multiple attempts are tried and the most successful definition of the problem and solution wins. An example for this would be in the creation of any new product in the commercial marketplace. The companies that successfully determine their potential customers problems and create products that meet those requirements tend to succeed while companies that fail to determine their potential customers problems tend to fail. [48]

An example given by Roberts of competitive strategies going wrong is the war between Japan and the US. In the 1930s both had a perceived problem of national security. Japan needed the oil in the Pacific Ocean to ensure its national security, however the US considered Japan entering the Pacific Ocean as a threat to its national security and this arguably caused the war. [48]

Collaborative strategies involve bringing together all the stakeholders and attempts to find a common ground solution. The benefit of this method is that resources are not wasted in competition. The example given by Roberts is the organisation of relief in Afghanistan. This involves the distribution of food, water to the populous and the reconstruction of the essential utilities (power, water, gas, etc.). It was difficult and painful for the stakeholders to meet up together and discuss, however the end results were a large improvement over the previous state. The drawback is that the more stakeholders that are involved the slower the process. [48]

According to Rittel and Webber [39], a popular idealist solution to Wicked problems that has been circling in the field of planning is to create a feedback decision-making system where the unexpected consequences of introducing solutions to Wicked problems could then be addressed iteratively. This is perhaps an early prelude to the iteratively development in software engineering. Iterative development and responding to new created requirements near the end of a project has been picked up by Agile Methods [49].

There is work on the collaborative approach as a solution to Wicked problems, such as TRAiDE [50] (discussed later in section 4.1) and Strategic Kinetic [51]. The common theme in all these solutions is to create a shared problem representation between the stakeholders through some kind of problem visualisation technique. Techniques typically used in these solutions include problem representation, dialogue mapping and the ability to perform what-if scenarios. Most of work done later on in this thesis is intended to support these techniques.

The purpose of the collaborative techniques is to form a better problem formulation of a 'Wicked problem' by including more stakeholders into the decision-making. Using collaborative techniques can't get around the characteristics of 'Wicked problems'. It is hoped that by including more stakeholders to get a better problem representation that better solutions can be created.

3.4 Evolutionary Warfare

Warfare is carried out in the natural world for the purpose of survival between predator and prey species and has been well studied [52]. Since a major motivating factor in the research is TLCM, which was designed for performing acquisition in military scenarios, it makes sense to discuss evolutionary warfare and see if there are any lessons that can be learnt.

Symmetric warfare is where two armies procure the same types of things to be sent to the front line; for example both armies decide to procure the same type of tanks. In this case it is quite clear that the army who has the most resources (so they can build the most tanks) is the most likely to win. Since for symmetric warfare to happen it requires both parties to agree to it, the party with fewer resources are likely to turn to asymmetric warfare to gain a better chance of winning. Symmetric warfare in the natural world tends to be avoided by the two species going after different resources in the environment [52]. The UK's recent wars have been asymmetric warfare [53].

An interesting occurrence in the natural world found by Valen [54] cited in [55] is that, contrary to popular belief that a species will gain more and more beneficial adaptations over time making species less likely to go extinct over time, is that all species go extinct at the same rate. The effect was called the Red Queen Effect after the Red Queen in the sequel to Alice in Wonderland [56] who keeps running as fast as she can but she cannot progress, since the landscape itself keeps up with her. Valen [54] cited in [55] proposed hypothesis is that the effective environment, which includes the other species that can make counter adaptations, will always be the same as the other species in the environment will eventually create counter adaptations to any beneficial adaptations a species makes. Dawkins and Kerbs [52] proposed that on the introduction of a new adaptation by a species that it will enjoy a small amount of time before it's opponent species determine a counter for this adaptation.

The Future Air and Space Operational Concept report [57] shows a example of this in that it says Air power has given them a large advantage over the enemy and in response now the enemy is making the Air power less effective by hiding in complex terrains, the use of tunnels and by hiding within urban environments. The report goes on to say that the enemies are also trying to use information warfare to misrepresent air power as a "cruel overmatch" and "blunt instrument of power". The nuclear weapons used by the United States in 1945 were clearly an almost uncounterable adaptation and by using it they were able to gain victory. However now due to new social and political pressures using nuclear weapons in normal war zones is unlikely to be viable. [57]

According to Cartlidge and Bullock [58] there are three endings for a co-evolutionary race.

- An uncounterable adaptation is found by one side leading to the extinction of the other side.
- A temporary uncounterable adaptation can be found that makes any strategy the other side uses to be equally effective [59–61].
- The co-evolution may enter a cycle where each strategy is beaten by another strategy as occurs with the side-blotched lizard mating strategies [62].

In side-blotched lizard mating strategies, there are three versions of males with different coloured throats [62], which are:

- The aggressive males with orange throats that defend large territories.
- The sneakier males with yellow throats, which is the same throat colour as the female side-blotched lizard, which do not bother with defending territories.
- The non-aggressive males with blue throats that defend small territories.

According to Sinervo & Lively [62], an over abundance of aggressive males leads to an over abundant of sneakier males who are able to mate undetected by the aggressive males as the aggressive males guard large territories. An over abundant of sneakier males leads to an over abundant of non-aggressive males as they guard much smaller territories and therefore are able to catch the sneaker males. An over abundant of non-aggressive males leads to an over abundant of aggressive males who can contest their territories. This results in the side-blotched male lizard population cycling between the three types of male lizards. [62]

Dawkins and Krebs [52] purpose that species have a limited budget of resources that adaptations can be bought with resulting in species choosing between generalising for a large number of species badly and specialising against a few species well. This raises the possibility that an army could be tailored specifically to defeat another army, which in the general case may be the stronger army.

What should be taken away from the section is that asymmetric warfare is far more common than symmetric warfare in general. Living species also have a limited budget for acquiring their adaptations for warfare and they make the choice between adapting for a large number of opponents badly or specialising against a few opponents well. The UK MoD is likely doing the former whilst some of the other countries are likely to be doing the latter. The value of new capabilities, even apparently uncounterable capabilities such as nuclear weapons is likely to degrade over time as the surrounding environment adapts, if not by military means via political or economic means.

3.5 Summary

In this chapter, we discussed the literature relating to Through Life Capability Management and research fields that provide general guidance to the research such as wicked problems and evolutionary warfare. In the next chapter, we will move onto discussing existing solutions that are already used in practice to address the large-scale acquisition problem.

Chapter 4

Literature Review - Current Practice

This chapter describes the current practice in relation to defence acquisition. The chapter begins by describing the TLCM Robust Acquisition inclusive Decision Environment (TRAiDE), which is a workshop based process using tool support that aims to facilitate trade-offs in defence acquisition. The chapter then covers the more traditional high-level acquisitions techniques used by militaries, which are Enterprise Architecture Frameworks and military simulations, before covering some basic work on modelling capabilities and the work produced by the Network Enabled Capability Through Innovative Systems Engineering (NECTISE) project.

4.1 TRAiDE

TLCM Robust Acquisition inclusive Decision Environment (TRAiDE) [50] is a process for performing trade-offs in defence acquisition. Daw [63] defines defence acquisition as a Wicked problem (discussed in section 3.3). Under Roberts' [48] approaches to solutions to Wicked problems (discussed in section 3.3.1), TRAiDE falls under the collaborative type solution category. [50]

The TRAiDE process starts by bring the stakeholders for the trade-off together in a workshop. Information is collected from the stakeholders and is stored in an information manager acting as a central place for information. The information manager supports the manipulation of its information by external tools, including tools for risk management, formal requirement management, simulation and performance analysis. The information manager is used to produce visualisations for the stored information. Visualisations are tailored to the different aspects of the problem for different stakeholders and are created with the ability for them to be manipulated during presentations. They all draw from a common information source so the knock-on effects of making changes in one visualisation can be seen in others. [50]

TRAiDE uses the five main perspectives used in the TLCM process: capability, industrial, commercial, financial and research. Visualisations are produced of various different acquisition scenarios for demonstration to the stakeholders for discussion and comments. Feedback is taken on information gaps and information inconsistency. A test of TRAiDE was performed on the Future UK Mine Counter Measure Capability (FMCMC) and was run over a six months period. This resulted in the restructuring of the FMCMC programme. [50]

TRAiDE [50] makes use of a modified Gantt chart called the Integrated Management Plan. The Gantt chart contains the acquisition programmes with their start time, in-service time and retirement time. The acquisitions programmes that are in-service at any one time are aggregated together to produce a capability-over-time graph alongside the Gantt chart. This is to allow the changes in capability to be managed over time by identifying gaps in capability between the retirement and replacement of components. Once these gaps have being identified, the stakeholders can move the acquisition programmes forward or backwards in time to help cover the gaps, to extend the in-service time of the existing components, or can choose to ignore the gap. [50]

The approach employs the collaborative approach to addressing wicked problems in running as a workshop to bring the stakeholders together. According to Daw, simply employing a workshop-based process without using the common conceptual model had major issues, due to the various stakeholders in military acquisition speaking with different terminology. A major part of TRAiDE is that the viewpoints allow the various stakeholders to see how their view of the acquisition problem, and their attempts to solve it, have knock-on effects in other people's views. [1]

The benefits of the approach are that it takes a collaborative approach to large-scale acquisition and attempts to help the various acquisition decision makers understand each other's perspectives. The drawbacks are that it takes a long time to perform in practice due to being a workshop-based process and needing to collect all the various stakeholders together. A major problem found was that for the Integrated Management Plan there was no known objective manner in which the various acquisition programs could be aggregated to determine military capability that worked. One of the major motivations for this thesis is trying to solve this problem; this issue is research gap 1.

4.2 Enterprise Architecture Frameworks

Enterprise Architecture Frameworks are widely used in the military for acquisition. The UK Ministry of Defence has adapted the US Department of Defense's Architecture Framework (DODAF) to create the UK MoD's Architecture Framework (MODAF) [64]. MODAF covers more of the Defence Lines of Development than DODAF, which focus mostly on the equipment line [65]. Enterprise Architecture Frameworks were popularised by Zachman [66] who took building architecture blueprints and adapted them for use on organisations instead of buildings [66]. The first framework was the Zachman Framework [67].

Enterprise architecture frameworks contain a large number of predefined viewpoints. To use an enterprise architecture framework, a relevant subset of the viewpoints for the problem is chosen and those are drawn up for the organisation in question. In normal building architecture, the viewpoints would be concepts such as the outer wall drawings, the electrical wiring, the plumbing, etc. Enterprise architectures act as a repository in which information can be stored. Example viewpoints for enterprise architecture are goal lists, process lists, process models and entity relationship models [67], through the viewpoints vary depending on the enterprise architecture framework being used.

A study into Enterprise Resource Planning systems [68], which are systems that group together several other systems, found that a hidden benefit of Enterprise Resource Planning systems was decision support. This is likely to be due to Enterprise Resource Planning systems collecting infor-

[1]Private discussion with Andrew Daw

mation into a central repository like in Enterprise Architecture Frameworks and in TRAiDE. Some Enterprise Architecture Frameworks have an underlying meta-model which allows the querying of information contained within the views [69].

At the Integrated Enterprise Architecture Conference 2010, a leading conference on the usage of enterprise architecture it was announced by a conference organiser as a safety warning to practitioners that there was little benefit in using an Enterprise Architecture Framework on an enterprise in the hope that some useful information will come out of it; to which there was no objection to the announcement. This is because the information captured by the Enterprise Architecture Framework will not be at the same level of abstraction or the right type of information as if the Enterprise Architecture Framework was created to solve a specific problem. This relates back to the concept of modelling (discussed in section 2.6), where a model is acquired or produced with a purpose in mind.

Enterprise architecture is used for managing changes in large organisations. For example the Singapore MoD is using enterprise architecture to merge together 90% of the processes contained within their land, navy and air force [70] and the Swedish Armed Forces are using enterprise architecture to help manage its transition from a conscript army to a professional army [71].

MODAF the UK MoD's enterprise architecture framework has a large number of viewpoints split into six categories, which are strategic, operational, system, technical, acquisition and service-oriented [72]. MODAF has an underlying metamodel (metamodels are covered in section 5.4) that allows querying of the information stored inside of the viewpoints [69]. MODAF begin as the US DoD's DODAF and was adapted to by the UK to include the Defence Lines of Developments [65]. DODAF itself has a heavy equipment focus [65].

Enterprise architecture frameworks such as MODAF or DODAF are not tools for making high-level trade-off decisions; however they are commonly used tools for both recording the acquisition decisions made and performing the management of the individual acquisition programmes, once the decisions have been made.

4.3 Military Simulation

The US Department of Defence classifies military simulation into three types [73]. Live simulation, which refers to war-games using real people and real systems; Virtual, which refers to using real people but within a simulated world; and Constructive, which refers to using simulated people operating in a simulated world. [73]

OneSAF is a product line containing multiple products designed to be used together to create military simulations and is used by the American Department of Defense [74]. It was developed in response to a recognition of a duplication of effort in developing modelling and simulation tools [75] cited in [74]. Its product line has an architecture that has been designed to facilitate reuse of its components for different military simulations [74]. It uses an agent based model and has the concept of a physical model, which represent the effectors and preceptors of agents and a behavioural model, which deals with how the agents act according to the applicable doctrine [76]. A language is defined in terms of domain concepts for allowing non-developers to describe the behaviour of agents and works by composing primitive behaviours [76].

The High Level Architecture [77, 78] is a Department of Defense standard for its simulations to promote interoperability between them; it has been accepted as an IEEE standard. The standard mandates the use of a run-time infrastructure that the developed simulations can connect to. This allows them to share information with other simulations. Multiple simulations running together are called a federation. It uses a publish-subscribe system where objects in the simulations are published then other simulations can then listen to and changes the other objects present. Even with the high level architecture, when the simulations have been developed independently there are problems in using them together [79, 80].

Military simulation mostly sees usage in the development of individual systems [81]. Simulations will be discussed in more detail in the next chapter (section 5.2).

4.4 Weighted Sum Based Approaches

Work by Wyer & Long [25] looks at modelling capability through-life. It considers the Fundamental Inputs to Capability (Australian version of the DLoD) in the limited sense of being restrictions on the number of deployable platforms. It models platforms using a different set of Measures of Performance (MoP) for every military capability being considered and for every role the platform can take. The papers points out that the number of MoP required can become large for reasonable sized problems. To convert the MoP into Measures of Effectiveness (MoE), weightings are given by the stakeholders for each of the threat environments (high, medium, low). This weighted sum approach is normally implemented by using a matrix for the MoP measurements, and multiplying them by another matrix of weights to generate a matrix for the MoE measurements. The acquisition of the platforms and the costs against time is modelled and the result is graphed. The work is limited in that it performs many simplifications in not considering anything but platforms directly contributing to military capability, and by only considering the possible threat environments as high, medium or low.

Weighted sum solutions such as this are widely believed in the defence community to be incorrect. For a basic explanation of why consider a ‘Long Range Strike’ capability implemented using a F16 Bomber, an aircraft carrier, a pilot training program and a pilot. In the weighted sum method, these four concepts are assigned weights that are summed together to produce the capability. The loss of the pilot training program may result in reduced capability rather than no capability if the pilot knows how to fly a similar aircraft; this can be express in weighted sum. The loss of any of the F16 Bomber, the aircraft carrier or the pilot should result in zero capability, however this simply cannot be expressed using weighted sum as the other terms in the weighted sum will still have values. The relationship between the components and the produced capability is not a relationship that is amenable to the weighted sum based approach.

Another more complex example is a ‘Video conferencing’ capability. The capability is produced by two computers, with video conferencing software, two webcams and a network infrastructure. Whilst the reduction of video quality due to using cheaper webcams can be expressed using weighted sum, the loss of either computer, or the loss of the network infrastructure resulting in no capability cannot be expressed via weighted sum. Nor the fact that if the webcams produce higher quality video than the network infrastructure is able to send, the extra video quality is simply lost. In short, the relationship between the Defence Lines of Development (DLoD) and capabilities is usually domain specific and non-trivial.

4.5 NECTISE Architecture Framework

In the now completed NECTISE project, Webster et al [82] starts to develop what is essentially an architecture framework coupled with a process for evaluating how system architectures can meet Network Enabled Capabilities. Network Enabled Capabilities are capabilities that have being created with a network of systems (another term for system of systems) working together. The framework has a meta-model for representing its structure. The framework defines a set of documents to be provided by the user, which includes a definition of the desired capability, a definition of a scenario to evaluate the capability against and a configuration for the services in meeting the capability. An outline of the intended process is given, however it is incomplete and needs further research.

Venters et al [83] extend this work by giving examples of converting military scenarios into MoEs. Unfortunately the work is too preliminary to be useful. Rather than providing an architecture framework and process it provides a general outline of what such a thing could look like with the intention that the author would follow up the work in later publications, which has not been done so far.

4.6 Summary

In this chapter, several existing techniques used in large-scale acquisition have been discussed. The collaborate approach TRAiDE to solving the problem of large-scale acquisition trade-offs have been discussed along with its remaining issues that need to be solved. Enterprise Architecture Frameworks that are commonly used to document solutions to large-scale acquisition has been discussed. A naive solution to converting DLoD into capabilities using weighted sum has been discussed along with the inaccuracy of this method. Military simulation and some of the preliminary results of the NECTISE project have also been covered. In the next chapter, research fields that may be applicable in addressing the research gaps will be discussed.

Chapter 5

Literature Review - Applicable Research Fields

5.1 Introduction

This chapter explores the relevant research fields that may be drawn upon to help address the identified research gaps (section 1.3). The research fields that will be discussed are simulation, agent based modelling, goal modelling, metaheuristic search, model-driven engineering, sensitivity analysis, decision support and feature models.

5.2 Simulation

This section explores the general techniques and ideas behind simulation whereas section 4.3 discussed the military specific adaptations to simulation. Simulation in general is where a model of a system of interest is used in place of the actual system to perform experimentation and analysis. The model needs to be fit for the purposes of the simulation for the results to be valid (discussed in section 2.6). Both the final state of the simulation and how the model changes over time can be studied. [84]

The core idea in simulation is that it is cheaper, easier and more feasible to make changes to a model of a target system of interest than making the changes to the target system. This is normally the case with most large systems. Making the changes directly to the actual system will produce better and more reliable results than experimenting on a model of the system but it will be more expensive. When changes to the model of the system are made the effects of the changes are recorded and these are used to inform decisions for changing the actual system. [85]

Simulation in the context of this thesis is promising because in large-scale system acquisition the effects of making acquisitions or changes to the large-scale system tend to be highly expensive and irreversible. This is due to the nature of Wicked problems [39] (section 3.3), of which large-scale system acquisition is one example, each problem tends to be unique and the changes made by any potential solution irreversible. This means that it is not possible to undo erroneous changes or alternatively attempt to learn from our mistakes to attempt to improve future decisions made on the large-scale system. This could mean that simulation could be used to test some potential solution to some problems, through simulation probably cannot be applied to some of the more social problems. Another potential use of simulation for our work is in the measuring of the quality of a capability; a simulation could be used to evaluate how well a system of interest will perform in a scenario before the system of interest has been acquired.

5.2.1 Discrete event approach

Most current research on simulations uses the discrete event approach. In the discrete event approach, there is a system state composed of a set of entities each with their own attributes. Each entity has multiple actions. The actions can lead to events, which can change the state of the system. The discrete event approach supports the use of resources that represent things in the simulation, which have a constrained capacity; limiting how much of that resource can be used. The inputs to discrete event simulations tend to be partially stochastic meaning the results from the simulations are also partially stochastic. Typically, statistical techniques are applied to the output of simulations to make sense of the resultant partially stochastic results. [84, 85]

5.2.2 Agent Based Modelling and Simulation

Agent based modelling and simulation [86,87] is a way to perform simulation of a complex system by modelling agents or small autonomous parts of the system and how they interact with other agents contained within the system. This is different from the discrete event approach described above in that each agent is limited to acting based upon its own internal state and information it receives from the environment. No agent can directly interfere with another agent's state. An agent can represent a person or a small part of the system that acts only according to its own knowledge and rule set. In agent based modelling and simulation a large number of these agents work together to produce complex emergent behaviours. This is relevant because almost all large-scale systems are composed of large numbers of interacting components. Agent based modelling has also seen widespread use in the social sciences under the name Agent Based Social simulation [88].

Hoverd and Stepney [89] argues that agents in real world systems never communicate directly with each other but instead communicate via mediating fields contained within the environment. A given example of mediating fields by Hoverd and Stepney is where one agent sees the reflected photons from another agent and is therefore aware of its presence. They go on to argue that the agents have internal states, which the agent acts on, and that the information they receive from the environment into their internal state will be incomplete. To implement this approach they suggest using a client-server architecture where the server contains the external states of each agent within the environment and each of the agents is a client that requests the external states of the other agents through the server.

5.2.3 General Simulation Concerns

As simulation size grows, the computational power needed to perform a run of a simulation will increase. An obvious solution for large simulations is to run them in parallel on multiple computers. Whilst parallelising simulations is non-trivial, a significant amount of research has been carried out in pursue of this aim [90].

Whether a simulation is valid or not can only be asked with respect to the purpose of the simulation [35]. This is because simulations are based on models of the system, which only describe a simpler version of the system (see discussion in section 2.6). For each question the simulation is to be asked about the system, the validity of the model used by the simulation must be re-established [35]. Simulations are not normally meant to give the perfect behaviour of the actual system but instead an approximation within an acceptable range [91]. The distance the results can vary from the real system while the simulation remains valid depends on the purpose of the simulation [91].

Originally our research was heavily considering the usage of simulation. However, it was found that goal modelling is a better and simpler abstraction for dealing with capabilities and the issues of simulation validation can be avoided because goal models provide a structured argument for why acquiring things allows an organisations objectives to be fulfilled that can be checked by human experts for validity.

5.3 Goal Modelling

Goal modelling is a set of techniques used for early requirements engineering. A large number of requirement techniques begin with some physical system in mind that will fulfil the needs of the stakeholders [39]. Goal modelling instead begins with the 'how' the stakeholders wish to solve their problems and formalised it as goals and these are eventually decomposed into systems, people or processes that are the 'what'.

The Zachman Architecture Framework [67] considers all acquisition problems to have a 'why', 'how' and 'what' part: 'why' is the acquisition taking place, 'how' will the acquisition solve the problem and 'what' will be acquired. In TLCM, military capability represents the 'how's of how the operations will be performed leaving the 'what' of what the physical systems will be undefined. The 'whys' for capabilities are contained in the defence planning assumptions that include the military scenarios the military expects to face in the future.

Goals and capabilities are similar concepts. Satisfying a goal and satisfying a capability both represent having the ability to satisfy some objective. A goal and a capability is effectively a 'how' statement, of how something will be done to solve the acquisition problem. The DLoD from TLCM and system, people and processes from goal modelling are both categorisations of things that can be acquired in the satisfaction of capabilities and goals. These are also similar concepts. They both correspond to some 'what' statement of what will be acquired to solve the acquisition problem.

The first identified research gap in this thesis is bridging the gap between the military capabilities and the acquirable things, which are categorised by the Defence Lines of Development (DLoD). This is similar to bridging the gap between goals and people, processes and people, which goal modelling already performs. Goal modelling therefore would be potentially a good place to start and build upon for solving the first research gap.

Some techniques focus on modelling the goals from the individual actor perspective within the organisation [22, 92] allowing the internal goals of the actors to be considered while others focus on modelling the goals of the entire system then discharging them to actors for implementation

[21, 93]. The technique developed later in the thesis will use the latter approach since it is the approach taken in Through Life Capability Management (TLCM). Work has also been done on using goal modelling for arguing the safety of a system [94, 95]. Some of the main goal modelling techniques are i* [22], KAOS [21, 93], GSN [94, 95] and GBRAM [92].

Goal modelling has two major competing styles, KAOS [21] and i* [22]. The style of goal modelling being used in TLCM is much closer to the KAOS style, which has goals belonging to the system as a whole, rather than the i* style of goal modelling, which has goals belonging to individual agents. The KAOS goal tree starts with the high level objectives and then decomposes them as a AND/OR graph and eventually all the leaf goals will be satisfied by agents which are either people, systems or processes [21].

Goal modelling is used extensively in the CATMOS technique to justify why acquiring certain sets of components results in satisfaction of military capabilities. The use of goal modelling has the effect that all the produced acquisition plans by the technique can be checked manually by human experts for validity.

5.4 Model Driven Engineering

Model Driven Engineering (MDE) is a principled approach to software engineering based on the concept of using models instead of code for the implementation of software systems. The advantage of doing this is that the abstraction level of specifying the software can be raised to the domain concepts of the software system stakeholders. [36–38]

The aim is to increase productivity and to allow more complex software systems to be written. A precursor to MDE comes from research into Computer Aided Software Engineering tools, which specify programs using graphical diagrams. One of the principles of MDE is that a model is written using the concepts of the problem domain rather than concepts relating to the software implementation. MDE employs the use of model transformations to change the model into a form that allows execution. [38]

A similar approach is taken in the Object Management Group's Model Driven Architecture [96] that defines a Platform Independent Model (PIM), which is a model of a system without technical detail and a Platform Specific Model (PSM), which is a model closely tied to a specific implementation technology.

A widely used approach to MDE is Domain Specific Modelling Languages (DSMLs) [38]. These are based on Domain Specific Languages (DSLs). A DSL is a programming language that focuses on a narrow application domain [97]. This is opposed to a general-purpose language such as C++, Java, Python, etc. that aims to be applicable to any application domain. The use of DSLs allows people to create applications for the target application domain much faster than general-purpose languages [97, 98]. The drawback is that the DSL must first be developed for the application domain before it can be used [97, 98]. DSLs have an abstract syntax tree that defines the concepts and relationships between the concepts that can exist in the language and a concrete syntax tree that defines the grammar syntax that programs written in the language must follow.

In Domain Specific Modelling Languages (DSMLs), instead of an abstract syntax tree a metamodel is used [38]. A metamodel is a model that describes the concepts and the relationships between the concepts that can be used in other models that are said to conform to the metamodel [99]. Once defined metamodels have a variety of uses. One such use is allowing a set of constraints to be written in the terms of the metamodel and then checked on models that conform to the metamodel [38]. Another use of metamodels is for enabling model transformation. Model transformation rules can be written in terms of a source metamodel and of a target metamodel and then models that conform to the source metamodel can be converted into new models that conform to the target metamodel [99]. This can be extended to use multiple source models with multiple source metamodels and with multiple target metamodels to create multiple target models. Example transformation languages using the metamodel concept are the ATLAS Transformation Language [100], Epsilon Transformation Language [101] and any transformation languages that comply with the QVT standard [102]. Model transformation is the main approach to addressing the gap between the Platform Independent Models and Platform Specific Models [103]. The use of DSMLs and model transformation also allows additional models to also be produced from the DSMLs such as safety models and formal proofs [97, 104].

A standard usage of MDE for software development is to create a DSL for capturing the problem in the conceptual terms used by the stakeholders and then creating another DSL that describes the problem in the conceptual terms of the implementation details. A model transformation is then used to convert problems specified in the stakeholders DSL to the implementation DSL and a model-to-text transformation is then used to convert the model written in terms of the implementation DSL into source code that can then be compiled and executed as a program. [38]

One MDE tool is Epsilon [101], which is an extensible model management platform built on top of Eclipse [105] and the Eclipse Modelling Framework [106]. The concept of Epsilon is to provide a common framework with a base language that other model management languages can be written on top of. The base language is Epsilon Object Language (EOL) and this serves as a general-purpose model management language. On top of this, multiple task specific languages have being written including the Epsilon Transformation Language (ETL) [107], Flock [108] and EuGENia [109].

MDE is relevant because it provides techniques and tools that allow the easy manipulation of models such as the models used in goal modelling and simulations. This thesis uses MDE to support the automatic restructuring of goal models.

5.5 Metaheuristic search

Metaheuristic search is a technique that can be applied to combinational optimisation problems [110]. A combinational optimisation problem is where there is some goal to be achieved by configuring objects and the challenge is to find the 'best' configuration [111]. A generalisation of the technique is multi-objective combinational problems that deal with more than one goal at a time [112]. Search is applicable to this research project, as we will be looking for the best configuration of procurement projects with respect to various goals. It may be possible to improve on work done in TRAiDE (section 4.1) by using this technique on its scheduling of projects.

Metaheuristic search allows the exploration of search spaces, which would normally be impractical to exhaustively consider due to the size of the search space or the time taken for each

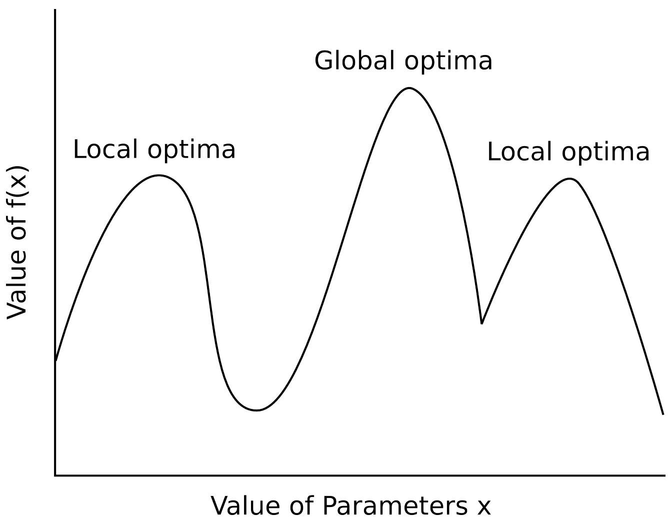


Figure 5.1: Example function f(x) with local and global optimum

evaluation of a potential solution [111]. They permit a trade-off between solution quality and the computational time needed to find it [113].

A general form of the metaheuristic search problem is to maximise a fitness function $F(x)$ where x is a set of parameters to $F(x)$ that can be changed to alter the value of $F(x)$ [113]. A function $F(x)$ normally contains local optima and a global optimum. The global optimum is the maximum value that a function can reach given the optimal input parameters. An example of this is shown in figure 5.5. The concept of a neighbourhood is some function that takes in x values and defines a new set of x values, which are considered to be nearby the x values. This leads to the concept of a local optimum. A local optimum is an x value for which $F(x)$ is greater than the $F(x)$ of all neighbouring x values, but x is not the global optimum. Metaheuristic search techniques need a method to leave these local optimal and reach the global optima. Briefly we will cover three of the metaheuristic techniques, which will be simulated annealing, tabu search and genetic algorithms. Simulated annealing and tabu search both explore one place in the search space at a time, as opposed to genetic algorithms that use a population based approach to explore multiple places in the search space at the same time [110, 113–115].

Simulated annealing [110] is based on the physical process of annealing, which aims to make certain materials (steel, copper, brass, glass) reach their ground state by heating them up and then slowly cooling them down. The ground states of a material are rare compared to all the other possible states they can hold and if the temperature is suddenly dropped the material finds a non-ground state.

A simulation proposed by Metropolis et al [116] for simulating interacting atoms within a material works by randomly creating displacements and accepting them if they led to less energy in the material and accepting them but only at a certain probability if they result in more energy in the material. Simulated annealing was invented by Kirkpatrick et al [110] who extended the algorithm of Metropolis et al by starting with a high temperature, which relates to a high probability of accepting higher energy configurations, and lowering it with time and having the accepting function to be a desired search goal. This was shown to find good results over a range of combinatorial problems. Kirkpatrick et al suggest that this is because the found state at high temperatures during the search got good gross features and at low temperatures the found state got good local features.

Tabu search [115] begins with a random solution to the combinatorial problem and then makes improvements to the solution by continuously taking the best locally optimal decision called a move. Locally optimal decision means by only changing one part of the current overall solution, which change leads to the greatest improvement. A tabu list is implemented that has a memory of previous moves taken and prevents these moves from being taken again for a short period of time. This leads to the search exploring more of the search space rather than just heading directly to the nearest local optima. Tabu search can make an exception to the tabu list called aspiration if a move would be a very large improvement.

Genetic algorithms [113, 117, 118] are metaheuristic techniques that explore a search space using a population of individuals that are scattered over the search space. Genetic algorithms use the concept of a chromosome. The chromosome is a representation of the solutions that is more suitable for using search techniques on like crossover and mutation. The chromosomes are then mapped to actual solutions, which can then be evaluated. The chromosomes are genotypes where the actual solutions are phenotypes. The genotype corresponds to the DNA in life that describes what a plant or animal will look like and the phenotype is the plant or animal itself. Genetic algorithms begin with an initial population of random chromosomes and then map them to actual solutions and evaluate all the solutions. The best solutions that are closest to the desired objective are then selected using a selection algorithm and then bred together using breeding operators to produce a new population replacing the old population. This process is repeated multiple times and usually causes the population to converge towards solutions that either satisfy or approximately satisfy the desired objective. Two normal breeding operations applied on members of a population to create new population members in a genetic algorithm are crossover and mutation. Crossover creates offspring chromosomes from two parent chromosomes by taking parts from each member. Mutation causes random changes within the chromosomes.

5.5.1 Multi-objective Search

Metaheuristic search was first extended to dealing with multiple goals by Schaffer [112] with his vector extension to genetic algorithms. There are three main ways to extend genetic algorithms to support multiple objectives, which are weighted sum, changing the objective function each round and using Pareto ranking [118]. Weighted sum is using the sum of the goals of interest as the goal. Changing the objective function is alternating between the various objective functions present. The most interesting for our work is Pareto ranking as it deals with making trade-offs in the allocation of resources. Pareto optimality is a well-known concept from the field of economics. A solution is considered to be Pareto optimal when to make any further gain to one objective it requires that another objective take a loss. The Pareto front is all the possible Pareto optimal solutions to a problem. This is useful because solutions that are not Pareto optimal do not need to be considered when making trade-offs because there will be another solution that is Pareto optimal that is strictly better than it. Goldberg [117] states that to get a good representation of the Pareto front requires both the use of ranking based on Pareto domination in the metaheuristic along with an anti-niching method to prevent all the solutions centring around a single point on the Pareto front.

A widely used multi-objective search algorithm is called NSGA-II [119]. NSGA-II attempts to evolve a population of members optimal in different ways by introducing the concept of a Pareto front to its fitness function and rewarding population members for how Pareto optimal they are. A solution is Pareto optimal when there are no other solutions that are better than the solution for all objectives. So for two objectives with two solutions (5,7) and (8,4) both are Pareto optimal since they are both better on an objective. If another solution (10,10) was introduced than it would be Pareto optimal and the two previous solutions would not be Pareto optimal since (10,10) is better than them both on both of the objectives. The NSGA-II algorithm penalises its population members proportionally to the number of other people members that are better than them for all objectives. NSGA-II also includes an anti-crowding algorithm, which penalises solutions for being close together. A main contribution of the NSGA-II algorithm was being able to rank the solutions in O (N^2) rather than O (N^3) time.

Finding the Pareto front of solutions for multi-objectives is useful because all the solutions on the Pareto front represent the trade-offs that can be made and no solution outside the Pareto front needs to be considered since it is strictly worse than a solution inside of the Pareto front. With the Pareto front, the decision maker can simply choose the trade-off they want to make from the front.

This is a useful technique for performing trade-offs as instead of giving one possible solution to the stakeholders it shows a large number of possible trade-offs between the stakeholders. The research field of multi-objective optimisation extends metaheuristic search techniques with the concept of the Pareto front. It is used in this thesis for addressing the second research gap, which is dealing with the multi-objective nature of the trade-offs involved (section 1.3.2).

5.6 Sensitivity Analysis

Sensitivity analysis in the context of modelling is where the effects of the inputs of a model are considered in respect to their effect on the output of a model [120]. A definition of sensitivity analysis is "The study of how the uncertainty in the output of a model (numerical or otherwise) can be apportioned to different sources of uncertainty in the model input" [121]. Sensitivity analysis can be broken down into local and global sensitivity analysis [121, 122]. In local sensitivity analysis the effect of varying one input variable at a time is considered on the outputs of the model whilst the other input variables remain constant [122]. In global sensitivity analysis the effects of varying one input is considered on the outputs of the model whilst the other inputs variable are also varied. [122].

Sensitivity analysis is a useful technique for checking the robustness of the outputs of a model or simulation based on the uncertainties present in the inputs to the model or simulation. Sensitivity analysis has been applied to the results of the to be presented technique (presented in chapters 7, 8 & 9) to check their robustness. This work was carried out in conjunction with another author and is published in [4].

5.7 Decision Support Systems

Decision support systems are computer systems designed to support people in the making of decisions. The types of decisions made using decision support systems are recognised to have both a structured part that is most easily handled by a computer and an ill-structured part that is mostly easily handed by a human [123–125].

In this thesis, the technique attempts to address the structured part of the problem. The ill-structured part of the problem, which is defining the desired capabilities and which trade-offs to make in satisfying them is left to existing techniques such as TRAiDE (section 4.1), which uses a workshop-based process to deal with the ill-structured part of the problem. The works leaves the most 'Wicked' (see section 3.3) part of the problem to existing techniques such as collaborative-based approaches and aims to address the more tamer structured part of the problem.

5.8 Product lines and Feature models

Techniques that aim to provide a configuration of a system such as product lines and features models are not applicable. This is because the number of different ways the components can come together to provide capabilities grows exponentially with the number of components. Whilst each individual acquisition plan in CATMOS is represented using a directed acyclical graph, a directed acyclical graph such as those used in feature models [126] cannot represent the entire search space or any significant portion thereof. Ignoring any notational issues or usability issues in using feature models for this type of problem, a feature model to represent the problem would require exponential space.

5.9 Summary

In this chapter, we have discussed various techniques that look promising in their application to the three research gaps identified in the introduction. In the next chapter, we will briefly recover the part of the problem identified in chapter 1 that the CATMOS technique will address.

Chapter 6

Problem Description

6.1 Introduction

In this chapter, the formalisation of the problem that the CATMOS technique intends to solve will be given. This is followed by discussion of how the CATMOS technique fits into existing research on solving acquisition problems. Finally, the current state of the art in solving the relationship between the Defence Lines of Development (DLoD) and capabilities is discussed and why it is unsuitable for the task.

6.2 Capability Management

Problem description 6.2 (related references [7, 8, 12, 16, 18]) gives an overview of the problem in Through Life Capability Management (TLCM) [7] that the CATMOS technique tries to address excluding the through life part of the problem. The through life part of the problem is discussed in the next section.

In TLCM [8], acquisitions are managed in terms of capabilities. These are abilities that are granted to front line commanders by performing acquisitions. The things that are acquired to create these capabilities are categorised by the DLoD (section 2.3).

It is known that the relationship between capabilities and the DLoD is many to many however the exact relationship is unknown [12]. This is related to the 'Apples and Wednesdays' problem.

The 'Apples and Wednesdays' problem, described by Barton and Whittington [18], is a problem in defence acquisition that the different acquisition programmes in the DLoD are completely different types of things. However, if they all contribute to the produced capabilities and all cost money, it should be possible to say that acquiring a new equipment programme is better or worse value for money then acquiring a new training programme. How to compare the benefits of the acquisition of new equipment programmes to the acquisition of new training programmes or performing comparisons between any of the DLoD is however unknown.

The acquisition programmes, categorised by the DLoD, can be part of the pre-existing System of Systems or they can be acquired for the System of Systems at cost. Even in large scale acquisitions, there are financial limitations for how much resources can be spent in performing the acquisitions. In the UK MoD's case, the acquisition programmes can be acquired either in-house or from industry [8].

The problem to be solved is that the acquisition makers want to maximise the fulfilment of the various capabilities whilst simultaneously reducing cost [16]. To do this, they need to make a trade-off between the satisfaction of the various capabilities and the costs involved. A sub problem of this is finding a method that can be used to go between the acquisition programmes and the produced capabilities. This is research gap 1 as described in detail in section 1.3.1. Providing the decision makers with the necessary information for trading-off between the various capabilities and the costs is research gap 2 described in detail in section 1.3.2.

Desired capabilities:

Long range strike capability

Surveillance capability

Scouting capability

Block route capability

Route clearance capability

Hard target removal capability

Anti IED capability

Air support capability

*

Many to many relationship.

Each programme can contribute to many different capabilities
and each capability can be contributed to by many different programmes.

The exact relationship between the desired capabilities
and the Defence Lines of Development is unknown.

*

Defence Lines of Development:

Training Programmes

Personal Programmes

Doctrine and Concepts Programmes

Infrastructure Programmes

Equipment Programmes

Information Programmes

Organisational Structure Programmes

Logistics Programmes

The programmes can be both sourced in-house and sourced from industrial vendors.

Budgetary limitations:

Whilst the budgets in large scale acquisition can be vast they are not infinite.

Problem:

Maximise the satisfaction of the desired capabilities by
selecting which programmes in the Defence Lines of Development to acquire.

Whilst considering the budgetary limitations.

Whilst considering the in-place brownfield systems.

Since all the desired capabilities can't be maximised whilst keeping to
budgetary limitations, provide support to the acquisition decision makers to allow
them to choose the trade-offs that they wish to make

Research Gaps:

Research Gap 1: Find a method to bridge the gap between
the capabilities and the Defence Lines of Development

Research Gap 2: Allow the decision makers to effectively explore the trade-offs in
choosing which capabilities to satisfy.

Prob desc 6.1: The Capability Management Problem Overview

6.3 Through Life Capability Management

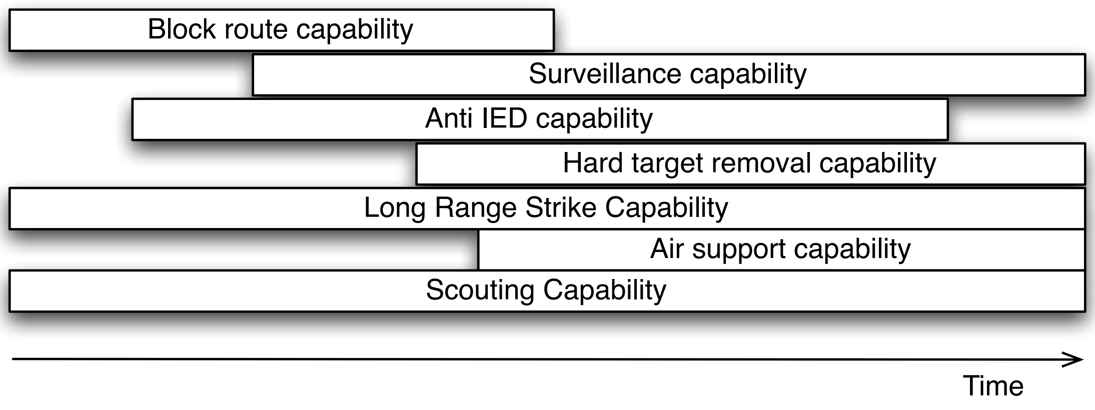
Problem description 6.3 (related references [7, 8, 12, 16, 18, 127]) shows the through life part of the problem the CATMOS technique intends to address in Through Life Capability Management (TLCM) [7]. The capabilities that are desired over time change as can be seen in MODAF StV-3 - Capability Phasing [127]. The desired capabilities change over time because the operations that the UK MoD intends to perform change over time.

The acquisition programmes that provide the desired military capabilities take time to first come into service and can eventually leave service. Whilst normally a replacement programme will be acquired to replace an acquisition programme leaving service, the change over can lead to capability gaps. It is desirable that these capability gaps are identified in advance, allowing either the existing acquisition programmes service time to be extended, the new programmes acquisition to be accelerated or the gap in capabilities to be accepted by the acquisition decision makers. [16]

The budget for acquiring new acquisition programmes and maintaining existing acquisition programmes is acquired over time by the UK MoD. The scheduling of the new acquisition programmes needs to be scheduled around the available budget and the acquisition programmes need to be ready for when the capabilities they support are needed and maintained until either the capabilities are no longer wanted or alternatively the acquisition programmes are replaced either with other similar acquisition programmes or more likely a different set of acquisition programmes fulfilling the same capabilities in a different manner.

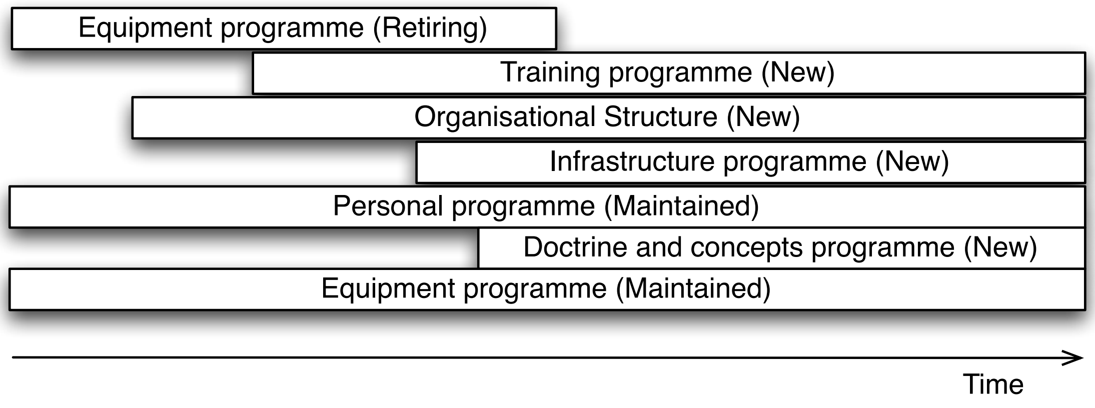
The main problem is to schedule the acquisition of the acquisition programmes around both when the capabilities they help provide are needed and around the budgetary limitations. In places where capability gaps can't be avoided the capability gaps need to identified in advance so appropriate actions can be taken if necessary [16]. Solving this problem, whilst simultaneously solving the problems from the previous section, make up research gap 3 as described in detail in section 1.3.3.

Desired capabilities:



Whilst some capabilities are wanted for the foreseeable future, other capabilities will eventually no longer be wanted as the way operations are conducted changes over time. New capabilities will also be introduced over time to allow new operations to be performed.

Defence Lines of Development:


Equipment programme (Retiring)
Training programme (New)
Organisational Structure (New)
Infrastructure programme (New)
Personal programme (Maintained)
Doctrine and concepts programme (New)
Equipment programme (Maintained)
Time

The individual programmes within the Defence Lines of Development can both enter and leave service. This can lead to capability gaps where programmes supporting capabilities leave service and the replacement programmes have not yet being acquired.

Budget:

◇ Incoming government funds
◇ Outgoing new acquisition programmes
◇ Outgoing maintenance costs and wages

Problem: Maintain the satisfaction of the desired capabilities through life as programmes retire and new programmes come into service.

Avoid gaps in capabilities where possible otherwise identify when the capability gaps will happen so they can be addressed properly.

Schedule the acquisition of new acquisition programs considering the capabilities they contribute to and the budgetary limitations.

Research Gap 3: Schedule the acquisitions of the programmes through life considering the capabilities they contribute to and the budgetary limitations

Prob desc 6.2: The Through Life Capability Management Problem Overview

6.4 Relationship with research on acquisition problems

Some general context is needed to clarify how the presented CATMOS technique will fit into the existing research for solving acquisition problems. A conceptual overview of this section is shown in problem description 6.4 with related references [39, 48, 67].

From Zachman's Architecture framework, three of the major factors involved acquisition problems are the 'why', 'how' and 'what' [67]. The 'why' factors are the reasons behind why the acquisition is taking place. This covers all the motivating reasons for why the acquisition has been decided on. The 'how' factors frame the problem as some objectives that need to fulfilled. The 'what' factor decides on the physical things that will be acquired to meet the problem. An illustrative example is the 'why' factor being 'Too many children passed the age of 10 are unable to read in the county', the 'how' factor being 'Provide more books to the local community' and the 'what' factor being a mobile library driving around the neighbourhood.

'Why' to 'how' is usually addressed by collaborative techniques such as TRAiDE [18]. TRAiDE, which was developed in conjunction with MooD International, was one of the motivating techniques for this thesis in that it was missing the underlying logic for performing 'how' to 'what'. In TRAiDE's case this is in connecting 'capabilities' to 'building blocks'. The technique presented in this thesis deals with going between the 'how' to 'what' in acquisition.

An influential paper in the area of the large-scale acquisitions is by Rittel & Webber who introduce the concept of a 'wicked problem' [39]. They describe problems faced in the field of large-scale social planning. They note that there is a change when problems become sufficiently large that they start to become defined by 'verbs' instead of 'nouns'. This is similar to the changes made by Through Life Capability Management [7] to Through Life Management (TLM) in that the UK MoD has recognised that their problems need to be defined by 'verbs' and not 'nouns'.

'Wicked problems' have numerous properties that make them intractable to solve by computational techniques (discussed in section 3.3). A standard approach for dealing with 'wicked problems' is to use a collaborative approach [48], which TRAiDE is one of. Dealing with the 'why' to 'how' of a problem using computational techniques would be incredibly difficult; however the 'how' to 'what' is mostly tractable because you can check whether or not a solution meets the desired objectives in the 'how'. Going from 'why' to 'how' mostly tames the problem.

Three solutions to 'Wicked problems' are the use of authoritative approaches, competitive approaches and iterative development [48]. In authoritative approaches, a few of the stakeholders decide on the problem and the solution by themselves, which tends to lead to failure [48]. In iterative development the solution to the problem is given in parts and user feedback is taken into account when producing the next part thus leading to the solution 'homing in' on the user requirements. Competitive approaches are where a large number of companies attempt to solve the same problem using authoritative approaches [48]. Many such companies go bankrupt; however a few of the companies will succeed in solving the problem [48]. In the TLCM case, iteration is provided by Urgent Operational Requirements (UORs) [43] however this means that there are problems with solutions that have been found while in usage on an active battlefield, and thus there is the goal to minimise the number of these that occur.

The CATMOS technique itself is designed to sit between the 'how' and the 'what' in Zachman's Architecture framework [67]. It takes in capabilities, which are wanted objectives (how)

Positioning CATMOS in the field of acquisition

Why *Why are we solving this problem?*

The four main methods for going from Why to How:

Collaborative Methods - Stakeholders come together have meetings and workshops. E.g. TRAiDE

Iterative Methods - Delivers solution in parts, takes user feedback into account before making next part

Competitive Methods - Lots of companies try to solve the problem, most go bankrupt, a few of them succeed

Authoritarian Methods - One person decides solution - Generally fails

There are trade-offs that can be made between why and how. CATMOS doesn't deal with these trade-offs.

In terms of wicked problems by Rittel & Webber, the why to how is very wicked

How *How are we going to solve this problem?*

This step appears when the problems get sufficiently large scale and people begin to recognise that what the solution is doesn't matter and how the solution was made doesn't matter but instead the only thing that matters is does the solution meet the wanted objectives.

The CATMOS technique sits between how and what. It takes in wanted capabilities (the how) and parts of possible solutions (the what) and it brings them together to produce acquisition plans.

CATMOS relies on there being an existing method like TRAiDE for going from why to how.

CATMOS deals with the possible trade-offs between the how and what.

In terms of wicked problems by Rittel & Webber, the how to what is less wicked. To deal with the fact that the problem is still slightly wicked, CATMOS provides justified goal models that can be examined by domain experts.

What *What are we going to acquire to solve this problem?*

References: Rittel & Webber 1997, Roberts 2000, Zachman 1987

Prob desc 6.3: The CATMOS technique in the field of acquisition and it takes in potential solutions (what) and matches them to find potential acquisition plans. It is envisaged that CATMOS will be used in conjunction with a collaborative method like TRAiDE for handling the 'why' to 'how' part of the problem. CATMOS was originally motivated due to problems found in the collaborative method of TRAiDE in that it was unknown how to go from capabilities (how) to acquisition programmes in the DLoD (what) when using the method. Through the 'how' to the 'what' is a lot more tame than the 'why' to the 'how' part of the acquisition problem it still has a slightly wicked nature. The CATMOS techniques answer to this is to provide justified goal models for the acquisition plans that explain how CATMOS goes from the 'how' to the 'what', so the acquisition plans can be checked by human experts for issues. CATMOS is intended to act as a decision support tool by providing the decision maker with different possible acquisition plans rather than a decision making tool.

6.5 Current naive weighted sum based approaches

Before introducing the CATMOS technique, the problems with the existing techniques will be explored. For this, we are going to demonstrate with a simple tea making example that the CATMOS technique will be applied to in the next chapter in section 7.3. The tea making example revolves around making a good cup of tea by acquiring a source of water, a source of tea leaves and a container to hold the tea in. In this example, the acquisition decision maker needs to decide what to acquire to maximise the goodness of the tea that can be produced whilst considering trade-offs against the cost.

This example will now be shown using a naive weighted sum based approach (similar to work done by Wyer & Long [25]). It should be noted that these weighted sum based methods are usually very simplistic, for example, considering each system contributing to the capability having an equal value [18].

Before applying a weighted sum based approach, a couple of problems need to be addressed. The first problem is that weighted sum based methods only consider one capability at a time [18, 25] whereas the tea making case study shown in figure 7.2 has four wanted capabilities in a tree structure.

There are two possible approaches to solving this with weighted sum. The first is to consider the bottom three leaf capabilities, 'Tea', 'Water' and 'Container' separately and the second is to consider them together using the top capability 'Good Tea' as an aggregate.

The next problem is that there is no support for concepts such as 'criticalValues' or 'benchmarkValues' in weighted sum based methods however weighted sum based methods get around this by pre-normalising the values. That is translating the measurements value so the 'criticalValue' is at 0 and the 'benchmarkValue' is at 1 as done in work by Wyer & Long [25].

The remaining problem is that weighted sum based approaches do not support dependencies between systems [25], which is because they are mostly underdeveloped. The work by Wyer & Long [25] does however support to some degree upgrading existing systems by writing a new set of MoP for the upgraded system and this mechanism can be used to implement a basic form of dependencies between systems. When a system upgrades a different system new Measure of Performances (MoP) values must be written out by hand for the new combined system [25].

When considering all the combinations of possible systems and how they interact together, this can result in an exponential amount of manual work making the technique impractical as is shown in the following sub-sections.

6.5.1 Considering the capabilities separately

For the first option of considering each capability separately, a table of the MoP values along with the costs for each applicable possible combination of systems for each capability needs to be created by hand. The individual capabilities satisfaction can then be added together using the formula: $GoodTea = 0.6 * TeaTemperature + 0.2 * TeaFlavour + 0.2 * InsulationQuality$
The weights are the same as used previously in section 7.6.

Combination of systems	Measurement of Performance	Cost
Kettle	0.0	10.0
Kettle, Hot Water Tap	1.0	10.0
Kettle, Cold Water Tap	1.0	10.0
Hot Water Tap	0.66	0.0
Cold Water Tap	0.13	0.0
Tea Maker	0.0	35.0
Tea Maker, Hot Water Tap	1.0	35.0
Tea Maker, Cold Water Tap	1.0	35.0
Nothing	0.0	0.0

Table 6.1: Tea Temperature Capability

The optimal choices for fulfilling the Tea Temperature capability are the Kettle at cost £10 and the Hot Water tap at cost £0. The rest of the choices are sub-optimal including the Tea Maker that provides water at the same temperature MoP at the Kettle but at a much greater cost. The fact that the Tea Maker also provides another capability Tea Flavour cannot be considered using this approach.

Combination of systems	Measurement of Performance	Cost
Tea Bags	0.1	4.0
Tea Maker	0.0	35.0
Tea Maker, Hot Water Tap	1.0	35.0
Tea Maker, Cold Water Tap	1.0	35.0
Tea Maker, Kettle	0.0	45.0
Tea Maker, Hot Water Tap, Kettle	1.0	45.0
Tea Maker, Cold Water Tap, Kettle	1.0	45.0
Tea Maker, Tea Bags	0.0	39.0
Tea Maker, Tea Bags, Hot Water Tap	1.0	39.0
Tea Maker, Tea Bags, Cold Water Tap	1.0	39.0
Tea Maker, Tea Bags, Kettle	0.0	49.0
Tea Maker, Tea Bags, Hot Water Tap, Kettle	1.0	49.0
Tea Maker, Tea Bags, Cold Water Tap, Kettle	1.0	49.0
Nothing	0.0	0.0

Table 6.2: Tea Flavour Capability

Due to the Tea Maker having dependencies, the number of combinations of systems that need to be considered has grown significantly larger for what is a small problem. There are likely to be serious scalability issues when considering actual problems that arise with this technique. The optimal choices for fulfilling the Tea Flavour Capability are the Tea Bags at cost £4, the Tea Maker with Tea Bags and (Hot or Cold) Water Tap at cost £39 and nothing at cost £0.

Combination of systems	Measurement of Performance	Cost
Mug	1.0	4.0
Plastic Cup	0.1	0.2
Nothing	0.0	0.0

Table 6.3: Insulation Quality Capability

For the insulation quality capability, all three of the options shown are optimal. The option that should provide the best capability possible at the lowest cost given by this approach is the Kettle, (Hot or Cold) Water Tap, Tea Maker, Tea Bags and Mug at cost £53, however this is incorrect. The best solution is just (Hot or Cold) Water Tap, Tea Maker, Tea Bags and Mug at cost £43 with the Kettle. This solution is missed by this approach since the capabilities are being considered independently of each other. While, it might be obvious in this trivial case, when the problems become non-trivial this type of problem will become much harder to spot. In general, any system that provides more than one capability is subject to errors using this approach.

Using nothing at all for the container is also considered to be a valid solution under this technique since while the Insulation Quality term of the weighted sum equation will be 0, the total satisfaction of the Good Tea capability can be higher than 0 due to the Tea temperature and Tea flavour capabilities being satisfied. Obviously, in practice you cannot have Tea without a container and these types of issues would need to be manually addressed in this approach.

There is also the issue that when the number of possible dependencies between systems grows, the number of combinations of possible systems that will need to be manually entered into Measures of Performance tables will increase exponentially making this approach infeasible for non-trivial problems.

6.5.2 Considering the capabilities together

Whilst considering the capabilities together will avoid some of the problems with considering them separately, the approach would need the MoPs for each combination of systems to be provided manually. This is 2^n entries in the MoP table where n is the number of possible components. In this example, n would be equal to 7 and there would be 128 different combinations of systems to be manually provided. This approach is effectively infeasible for any non-trivial problem due to the significant amount of manual effort required.

6.5.3 Comparison with naive weighted sum based approaches summary

In summary, there are three main problems with naive weighted sum based approaches as follow:

1. The introduction of dependencies causes an exponential amount of work for the decision maker to handle making the technique impractical for non-trivial problems.
2. Weighted sum based approaches cannot handle relationships for which any of the capabilities being unfulfilled leads to no overall capability being produced. This is just one of the relationships that the weighted sum based approaches cannot handle (see discussion in section 4.4 for more information).
3. The capabilities in question can either be considered separately leading to incorrect answers in the case where a system satisfies more than one capability, or together that leads to the approach being infeasible due to an exponential amount of manual work being required by the decision maker.

The CATMOS techniques tackles the three main problems as follows:

1. The technique avoids the problem of exponential work from dependencies by allowing the decision maker to define parts of possible solutions using the concept of *components*, which describe the individual systems and how they interact with each other.
2. The technique implicitly supports the relationship where no capability is produced if there are missing dependencies and the CATMOS technique is capable of supporting any form of more complex relationship using an external programming language; in the prototype tool case, Lua [128, 129].
3. The technique avoids the third problem by using a conceptual model based on goal modelling that implicitly supports the many-to-many relationship between capabilities and the Defence Lines of Development.

6.6 Summary

In this chapter, we have briefly cover the parts of the problem identified in chapter 1 that the CATMOS technique will address in the following three chapters. This has being followed by explaining how the CATMOS technique will be situated in the larger picture of research on acquisition problems and this has being followed up by looking at the problems with the current work towards relating the Defence Lines of Development to capabilities. In the next chapter, we start to introduce the CATMOS technique.

Chapter 7

Bridging the gap between the DLoD and capabilities

7.1 Research Overview

In this chapter and the next two chapters the research results is presented focusing on the three research gaps identified in section 1.3. The first research gap focuses on providing an objective method for transitioning from the desired capabilities to the Defence Lines of Development (DLoD). In this chapter, the research that addresses this gap is explained, and this is followed by a demonstration on the simple example of acquisition trade-offs in purchasing equipment for making tea.

In the next chapter, the second research gap of finding and presenting trade-offs to the acquisition decision makers is described. This is applied to a common problem found in the search based software engineering community of the Multi-objective Next Release Problem (MONRP) [130]. The novelties of using the CATMOS approach over existing approaches to the MONRP are then discussed.

Finally, in chapter 9, the research towards addressing the third research gap, managing the through life issues of capability based acquisition, is then covered. This is demonstrated using a realistic military case study.

Some of the research ideas that contribute to this chapter have already been published by the author in [1, 3].

7.2 Introduction

There is currently no working objective method for bridging the gap between artefacts acquired in acquisition programmes (categorised by the DLoD) and produced capabilities. This is because of the conceptual difficulty of mapping highly concrete acquisition programmes to the very abstract notions of military capability.

The current state of the art objective methods rely on using weighted sum [18, 25]. These methods tend be rather naive and can be as simplistic as giving each of the eight Defence Lines of Development for a capability an equal weight and adding them together when they come into service. The results of these methods do not survive trivial inspection. These methods are widely known in the defence acquisition community to be incorrect [12, 18] as discussed in section 4.4. A small example of the application of a weighted sum based method is shown in section 6.5.

The relationship between the DLoD and the produced capabilities is more complicated than weighted sum, and appears to be domain specific, depending on the exact systems and the desired capabilities. Additionally, according to Yue & Henshaw the relationship between the acquisition programmes, categorised by the DLoD, and the desired capabilities is many-to-many [12], which makes the relationship even more complicated. To say that the relationship is many-to-many means that each component can contribute to fulfilling more than one capability and each capability can be satisfied by multiple different components meaning that neither the components nor capabilities can be considered in isolation.

The question thus emerges of how to manage this relationship. According to Zachman [67] acquisition can be categorised using the terms of 'why', 'how' and 'what'. The move to capability-based acquisition is effectively a transition from specifying acquisition problems in terms of 'what' (physical things) to specifying them in terms of 'how' (desired abilities). Capability-based acquisition arises solely from the military domain; hence there is the question of whether an equivalent technique to capability-based acquisition exists.

From the research field of early requirement engineering there is already an existing technique called goal modelling that sits in the place of the 'how'. A contribution of this thesis is the recognition that goal modelling and capability-based acquisition both sit in the 'how' category from Zachman's framework [67] and though they have been developed independently, have substantial similarities.

The two major goal modelling techniques are KAOS [21] and i* [22]. The closest of the two techniques to capability-based acquisition is KAOS [21], as it considers goals belonging globally to the whole system rather than i* that considers goals belonging to individual agents in the system [22].

A capability (from capability-based acquisition) is conceptually equivalent to a goal in goal modelling. Both capabilities and goals can be decomposed into sub-capabilities and sub-goals. Both capabilities and goals can have real world measurements attached to them for evaluating how well the capability or goal is met [27, 127].

Capabilities are eventually met by the acquisition programmes, which are categorised by the Defence Lines of Development [12]. From now on, for clarity of writing we will define the term *Component* to refer to 'something acquired from an acquisition programme categorised by the Defence Lines of Development'. Goals are eventually met by an agent: a system, person or process [21]. A component from capability-based acquisition and an agent from goal modelling are broadly speaking equivalent concepts. Both represent either acquirable or acquired things that can satisfy the goal tree or capability requirements. The main difference is that the types of things that can be acquired have been categorised differently, i.e. equipment and infrastructure from the DLoDs are clearly systems whereas doctrine and concepts and training from the DLoDs are clearly processes. Effectively, capability-based acquisition and goal modelling are equivalent.

A difference that we are imposing in CATMOS on components (to set them apart from agents) is that components can have their own dependencies. This is due to a difference of scope between the two techniques. KAOS goal modelling focuses on the acquisition of a single system where dependencies between systems tend to be rare. In capability-based acquisition, when considering acquisition of system of systems, system dependencies tend to be quite common. Since system dependencies can be satisfied by multiple different systems each providing similar services, we choose to model these dependencies as capabilities that need to be satisfied before the component itself can be used.

A high level overview of the process behind goal modelling based on work by Lamsweerde et al [21] and a high level overview of the process behind the CATMOS technique are given below for a more detailed comparison.

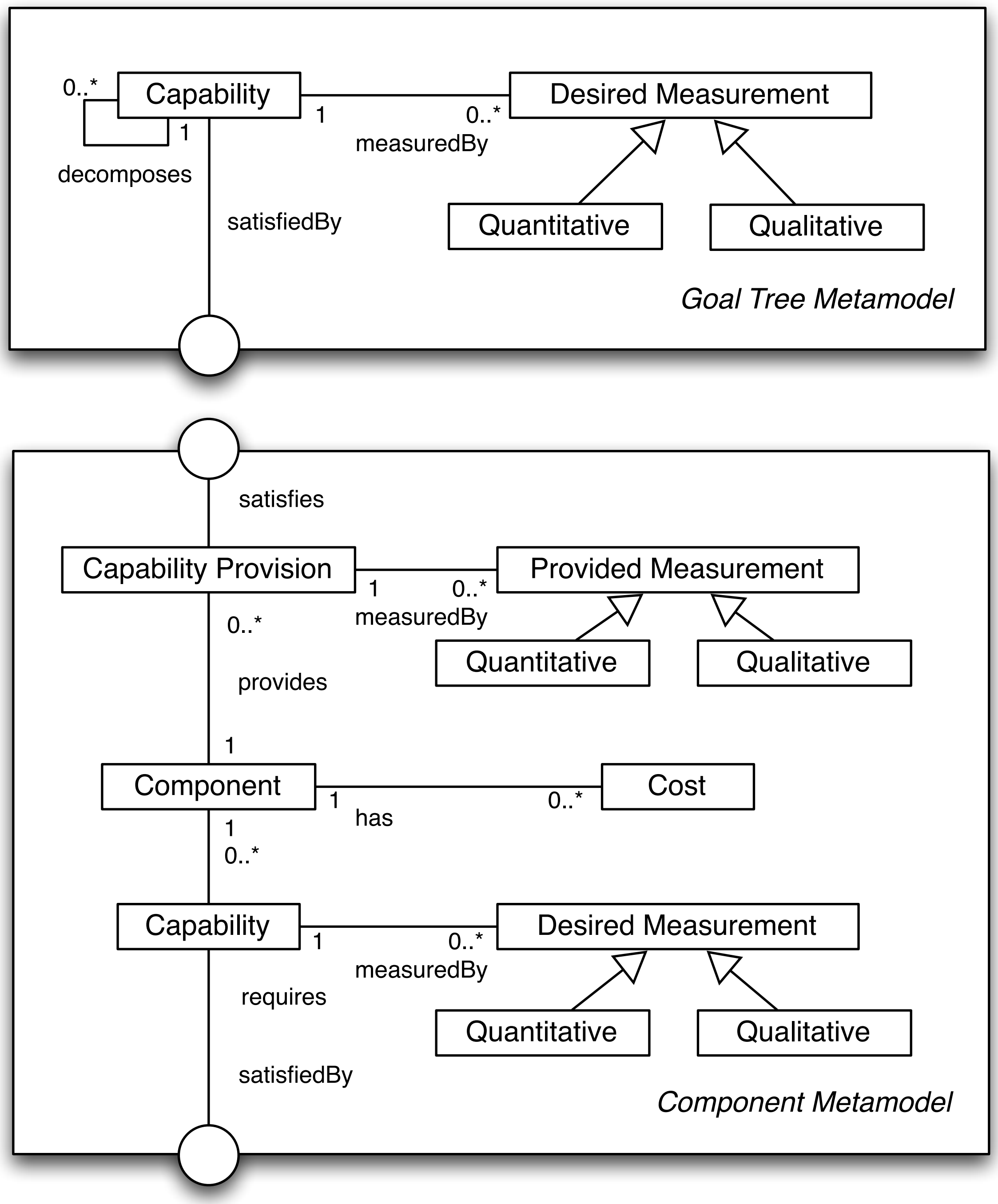
Goal Modelling Process High-Level Overview [21]	**CATMOS Technique Process High-Level Overview:**
Identify top-level goals by asking 'How' questions.	Identify top-level capabilities by asking 'How' questions.
Decompose top-level goals into smaller more manageable sub-goals.	Decompose top-level capabilities into smaller more manageable sub-capabilities.
Discharge responsibility for satisfying these sub-goals to agents (systems, people and processes).	Identify existing and acquirable components.
Once all sub-goals are satisfied the goal model is completed.	Identify the components' abilities to satisfy capabilities.
	Identify the needed capabilities for the components to function.
	Identify costs for acquirable components.
	Use tool support to automatically find the trade-off space of possible completed goal models.

Table 7.1: Comparison between Goal modelling and the CATMOS technique.

In a goal modelling technique such as KAOS [21], the acquisition decision maker begins by identifying the top-level organisation goals and then repeatedly decomposes these organisational goals into smaller sub-goals until there are agents (systems, people and processes) available that can directly satisfy these sub-goals. The acquisition decision maker then assigns the responsibility to satisfy these sub-goals to the relevant agents. Once there are no unsatisfied leaf sub-goals, the goal model is considered to be complete. The main steps of the goal modelling process are shown in the goal modelling process high-level overview in table 7.1.

The CATMOS technique has some notable differences. Firstly, we use terminology from capability-based acquisition, hence the term capabilities instead of goals and components instead of agents. In CATMOS, the acquisition decision maker begins by identifying the top-level organisation capabilities (goals) and then repeatedly decomposes these organisational capabilities (goals) into smaller sub-capabilities (sub-goals) until there are components (agents) available that can directly satisfy them.

Unlike goal modelling, the next step is not to directly assign the leaf sub-capabilities (sub-



Capabilities are connected to CapabilityProvisions via the satisfiedBy/satisfies relationship. Completed Goal Models are composed from a single Goal Tree model and multiple Component models connected together through this relationship. The Component models can be either connected directly to the Goal Tree model or as dependencies to other Component models through the satisfiedBy/satisfies relationship.

Figure 7.1: CATMOS Conceptual Metamodel

goals) to the relevant components (agents). Instead, the next step in CATMOS is to define all the existing and acquirable components (equivalent to systems & agents) individually. This is because the assignment of sub-capabilities to components defines a single solution. Whereas we intend to generate multiple solutions so we leave this undefined and allow the tool support to automatically define the assignments for us when it generates each of the multiple solutions. Each component is defined to have capability provisions that describe the capabilities (goals) that the component can satisfy.

This forms an additional layer of abstraction compared to normal goal modelling. This allows multiple components to be defined that all provide the same capabilities, meaning that multiple components can be considered for satisfying the same capabilities in the technique. In goal modelling once a sub-goal is satisfied by a system, person or process, the derivation of that part of the goal tree stops.

The modularisation of the goal model into separate parts using an abstraction layer is a novel contribution of the work. Since components are defined separately from the goal tree they can have associated acquisition costs that are incurred when they are included within a particular acquisition solution. They also have capability dependencies that represent needs that must be satisfied by other components, before the component itself can be used in a solution. These two additions are novel contributions of the work enabled by the modularisation of the goal model.

Once the acquisition decision maker has specified both the *Goal Tree* model, containing the capability decomposition, and multiple *Component* models, it is then the job of tool support to calculate completed *Goal Models* from these partial descriptions. In CATMOS, a *Goal Model* is a *Goal Tree* with attached *Components* that satisfy the goal tree. The *Goal Tree* metamodel is shown in figure 7.1. The tool support automatically generates multiple completed *Goal Models* that correspond to the 'optimal' trade-offs that can be made by the acquisition decision maker. This will be described in depth in Section 8. An overview of the main steps of the CATMOS process is shown in table 7.1.

CATMOS has the concepts of both capabilities and capability provisions. A capability represents either a stakeholder objective or a component's need that needs to be fulfilled by another component before it can be used. A capability provision represents the ability of a component to satisfy a capability.

Returning to the *Component* metamodel in figure 7.1, capabilities are annotated with measurements. Capabilities in the *Goal Tree* are annotated with both the minimal necessary measurements for the capability to be considered partially satisfied by a component and the benchmark measurements that are the ideal wanted measurements from a component satisfying the capability.

CapabilityProvisions in the component models are annotated with the *Measurements* provided by the *Component*. Needed *Capabilities* in the component models are the same as *Capabilities* in the *Goal Tree* with minimal and benchmark measurements.

During evaluation desired *Capabilities* either directly in the *Goal Tree* or as dependencies to *Components* can be in one of four states:

- Fully satisfied
- Partially satisfied

- Partially satisfied but with missing dependencies
- Not satisfied

Capabilities are evaluated by propagating the provided values up from the satisfying *Component's CapabilityProvision's Measurements* and comparing them to the *Capabilities'* critical and benchmark values.

A *Capability* is fully satisfied if for all the *Measurements* on it, the provided value is greater than the benchmark value meaning that all the wanted benchmarks have been met. A *Capability* is not satisfied if for any of the *Measurements* on it, the provided value is less than the critical value meaning that one of the critical measurements has not been met rendering the capability unusable. A *Capability* is partially satisfied if all of its provided values are greater than the critical values but one or more of the values is less than the benchmark value. A *Capability* is partially satisfied but with missing dependencies if it would normally be partially or fully satisfied but one of its needed *Capabilities* is either not satisfied or is partially satisfied but with missing dependencies itself. This means that the not satisfied state propagates up the goal model reducing the states of everything above it to at most partially satisfied, but with missing dependencies (not satisfied capabilities simply remain not satisfied).

If the critical value on a *Measurement* is greater than the benchmark value, the *Measurement* is assumed to be wanted to be as low as possible instead of as high as possible and the previous criteria for the four states are changed with the greater than conditions becoming less than conditions and vice versa.

In MODAF [23], only benchmark values and provided values are considered. The CATMOS technique extends this with critical values to allow the partial satisfaction of capabilities to be considered properly. MODAF also allows the definition of required ranges as upper and lower bounds (e.g. a band of radio frequencies), this can be handled in the CATMOS technique by using two measurements, one for the upper bound and one for the lower bound. So CATMOS can handle capability measurements as defined by MODAF.

The satisfaction of *capabilities* is ordered. A fully satisfied capability is always more satisfied than a partially satisfied capability and a partially satisfied capability is always more satisfied than a partially satisfied capability with missing dependencies and a partially satisfies capability with missing dependencies is always more satisfied than a not satisfied capability. Partially satisfied capabilities and partially satisfied capabilities with missing dependencies are considered to be satisfied between 0 & 1 by performing a linear evaluation. The linear evaluation is performed using the formula:

$$\text{satisfaction} = \frac{\text{provided measurement from the system} - \text{minimal measurement acceptable}}{\text{desired measurement} - \text{minimal measurement acceptable}}$$

A small example of evaluating the satisfaction is if 500 watts are wanted from a power generator, 250 watts are needed and only 450 watts are provided then the satisfaction for the power capability is 80% $(0.8 = (450.0 - 250.0)/(500.0 - 250.0))$. Values above 100% are clipped to 100%. In the case of multiple measurements being used to measure the satisfaction of a capability it takes the average of all the measurements.

The alternative formula for when measurements are desired to be low rather than high is:

$$\text{satisfaction} = 1 - \frac{\text{provided measurement from the system} - \text{desired measurement}}{\text{maximal measurement acceptable} - \text{desired measurement}}$$

The usage of the linear evaluation on partially satisfied capabilities and partially satisfied capabilities with missing dependencies has two uses. The first is that it allows partial satisfaction of a capability to different degrees, using the example above a solution that provides 450 watts is almost always better than a solution that only provides 300 watts and whilst it may be desired to fully satisfy all capabilities benchmarks due to the limited resources this is very unlikely to be possible.

In the simple case of a single measurement on a capability the formula is just a linear scaling between two bounds chosen by the acquisition decision makers and so is always adequate. In the less simple case of multiple measurements on a capability where the results of the linear scaling are averaged together it is not guaranteed to be adequate. In this case where it is not adequate, the solution is to create a parent capability with a single measurement containing a script that aggregates the values of the measurements together in the domain specific manner. Then instead of marking the capability as a search objective, mark the parent capability as a search objective meaning that the parent capability with the domain specific formula will be used during the multi-objective search.

The approach taken in CATMOS is that there is a default behaviour for evaluating goals that the acquisition decision makers can overwrite when is not appropriate to their needs. Similar work by Letier & Lamsweerde [27] that annotates goals in a similar way to how this work annotates capabilities avoids this problem entirely by having no default evaluation on goals. In their work the acquisition decision maker needs to specific a formula for the evaluation of each goal that needs to be evaluated.

The second usage for the linear evaluation is in supporting the multi-objective search technique that will be introduced in the next chapter. Using a linear evaluation provides a smooth search space for the search to work upon and this aids in finding better results. This is why the linear evaluation is still performed on partially satisfied but with missing dependencies capabilities even through the result is likely to be of less use to the acquisition decision makers. In the prototype implementation, capabilities are given a value 0 for not satisfied, 0 - 1 for partially satisfied but with missing dependency, 1 - 2 for partially satisfied and 2 for fully satisfied as explained in section 10.4.3.

The *Goal Tree* metamodel is similar to MODAF StV-2 [127] in that it defines capabilities in a nearly identical way. MODAF StV-2 being a strategic viewpoint in MODAF that provides a taxonomy of the capabilities being used [127]. It is intentional that they are similar because MODAF is an existing technique used for capability-based acquisition within the UK military acquisition community. The definition of capability used by CATMOS includes qualitative measurements from MODAF that are category-based with some qualitative measurements having values that are considered to be greater or lesser than each other. The *Component* metamodel is unique to CATMOS and has no direct equivalent in MODAF.

7.2.1 CATMOS Metamodel

CATMOS is underpinned by a metamodel that defines both the *Goal Tree* and *Component* models and how they can be joined together. The use of a metamodel allows the application of several MDE tools. A GUI interface has been semi-automatically generated from the metamodel using EuGENia [131] and GMF [132]. Additionally, a textual interface to the tool has been semi-automatically created using Xtext [133] and this is shown in section 7.5.

7.2.2 Aggregating Sub-goals

The top-level capabilities in the *Goal Tree* that decompose into sub-capabilities are not directly satisfied by components. Instead they derive their value from their sub-capabilities. These aggregations are largely domain-specific and so the CATMOS technique needs to allow almost any formula to be used. Allowing the user to specify the aggregation using a Turing complete language does this. Earlier versions of CATMOS used Epsilon Object Language [101] for this; however the current version uses Lua [128, 129]. Any standard Turing complete language is acceptable. Non-Turing complete languages may also be usable on a case-by-case basis. These formulas between a capability and a sub-capability are generally conversions between Measures of Performance to Measures of Effectiveness. Good examples of these types of formulas can be found in work by Urwin et al [134] and Venters et al [83].

Lua [128, 129] is a Turing complete language that was designed for embedding in programs to allow the end-user to extend the programs behaviour [128, 129]. The language is widely used and has extensive library support [128, 129]. The CATMOS technique has, as a requirement, a need to allow the end-user to specify complex relationships between parts of the goal tree and within the individual components between their dependencies and provisions. It also has a requirement to allow the technique to interface with external tools to be able to reuse information from existing simulations, real-world data sets and mathematical models. Lua is able to meet these needs.

7.3 Tea Making - Example

For explanation purposes, we will now go through the CATMOS technique on an illustrative example. The example that has been chosen is acquiring equipment to make ‘good’ tea. The example has been chosen for explanatory purposes since it is a domain that almost everyone can understand. The goal tree model and components model for tea making are shown in figure 7.2. Starting with the goal tree model, the main wanted capability is *Good Tea* that is being assessed by a *Tea Rating* measurement. The *Good Tea* capability decomposes into three sub-capabilities; the *Tea Leaves* that are being used and their *Flavour*; the *Water* and its *Temperature*; and the *Container* and its *Insulation*.

There are 7 components that exist or can be acquired in this example: the *Tea Bags* that provide tea and cost £4; the *Hot Water Tap* that provides hot water; the *Mug* that is a insulated container and costs £4; the *Cold Water Tap* that provides cold water; the *Kettle* that provides *Water* at 100 °C if it is provided with *Water* from another source. The Kettle costs £10. Plastic cups that are an uninsulated container and are cheap at 20p each. The *Tea Maker* that provides *Water* at 75 °C and

Tea Leaves with excellent flavour if it is provided with *Water* and *Tea Leaves* itself with the tea having at least flavour '*Good*'. This means that the *Tea Maker* takes in *Tea Leaves* and improves their flavour before passing them out into the Tea. The *Tea Maker* costs £35. The *Tea Maker* is a good example of a system that has dependencies that need to be fulfilled before it can make available its provisions. For example, the *Tea Maker* cannot provide hot water without first being provided with water.

The leaf capabilities in this example all contain Measures of Performance (MoP) that are directly satisfied by a corresponding component model. The only thing that is left unspecified in figure 7.2 is where does the value of *Tea Rating* come from. This is example of the difference between Measures of Performance (MoP) (section 2.4) and Measures of Effectiveness (MoE) (section 2.5). The *Tea Rating* is a MoE and its value needs to be derived from the MoP of *Flavour*, *Temperature* and *Insulation*. The CATMOS technique allows the specification of arbitrary formulas for deriving the values of measurements. The prototype tool uses Lua [128, 129] as described in section 7.2.2 for this purpose. The formula itself can be obtained either from domain expert knowledge, simulation or actual testing. In more complicated scenarios obtaining these formula may be a difficult task. There is however work [83, 134] that does derivate these types of formulas for military scenarios. The function used in this example is shown in section 7.6. Earlier versions of this research [3] used domain specific concepts for implementing the conversion between MoP and MoE and this motivated the inclusion of a embedded programming language to allow the conversion of the MoP to MoE without relaying on different domain specific concepts for each problem.

The prototype tool provides both a GUI interface and a textual interface for specifying the models. For the prototype tool, the textual interface has received more development for inputting problems. The textual specification of the models shown in figure 7.2 is shown in section 7.5. The *Capability* and *Component* parts are just direct translations from the graphical format shown in figure 7.2. The *FindTradeOff* part contains the general settings for the prototype tool. Once the problem has been input, the prototype tool then generates multiple completed goal models that represent possible solutions.

An example of a completed goal model that has been automatically generated by the technique is shown in figure 7.3. The completed goal model contains the initial goal tree model and several of the component models that have been included to make up the solution. In this case the *Tea Maker*, the *Plastic Cup*, the *Tea Bags* and the *Hot Water Tap* have being included along with the initial goal tree.

Additionally, links have being created between *CapabilityProvisions* and *Capabilities* between the *Goal Tree* and the *Components* and between the *Components* themselves showing how each capability is being fulfilled in the solution. Links are only created when the *CapabilityProvision* and the *Capability* have the same name. In this case, the *Tea Leaves* and *Water* from the *Tea Maker* has been connected to the *Tea Leaves* and *Water* in the initial goal tree, *Container* from *Plastic Cups* has been connected to *Container* in the initial goal tree and *Tea Leaves* from the *Tea Bags* has been connected to the *Tea Leaves* dependency from the *Tea Maker* and the *Water* from the *Hot Water Tap* has been connected to the *Water* dependency from the *Tea Maker*.

The goal model is considered to be complete when there are no dangling capability dependencies. A capability dependency is dangling if there is no connected *CapabilityProvision*. When a *CapabilityProvision* is joined to a *Capability*, the measurements from the *CapabilityProvision* are passed up to the *Capability* (filling in the providedValue fields in the *Capabilities'* measurements). This is subject to a single caveat: that the *Components* dependencies are all at least partially satisfied first. To be at least partially satisfied the provided value must be greater than the minimal acceptable value, which is called the critical value. Using the critical value, the benchmark value from the *Capability* and the provided value from the *CapabilityProvision*, the *Capabilities* satisfaction level is evaluated. The evaluation is performed linearly between the *criticalValue* and the *benchmarkValue*. As an example for the *Tea Making* measurement, the calculation performed using the formula defined in section 7.2 is:

$$0.75 = \frac{4 - 1}{5 - 1}$$

$$\text{satisfaction} = \frac{\text{providedValue} - \text{criticalValue}}{\text{benchmarkValue} - \text{criticalValue}}$$

giving the *Good Tea* capability a satisfaction of 75%. The costs of all the included *Components* are summed, giving a total acquisition cost of £39.20 for the solution.

Using the same problem description, other completed goal models can be created. An example of another two completed goal models for the same problem description are shown in figure 7.4. These have different levels of satisfaction and different costs from the first example. It should be noted that the top example in figure 7.4 has the same overall satisfaction as the first example but with a greatly reduced cost making the first example an inferior solution to it. How to decide which solutions are considered to be good or bad is discussed in the next chapter. What's important for now is that multiple different solutions can be derived from the same problem.

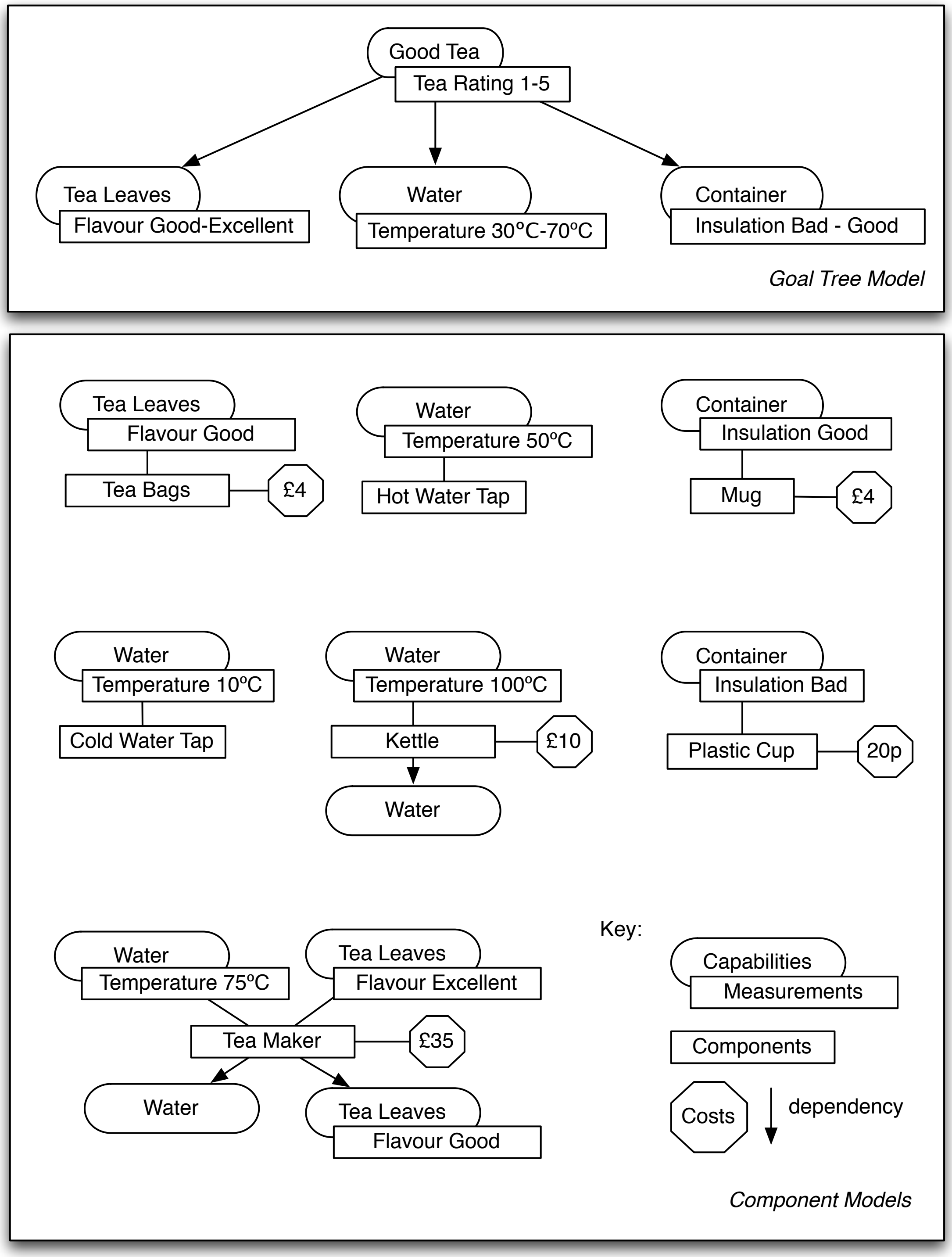


Figure 7.2: Tea Making Example, Goal tree and Component models

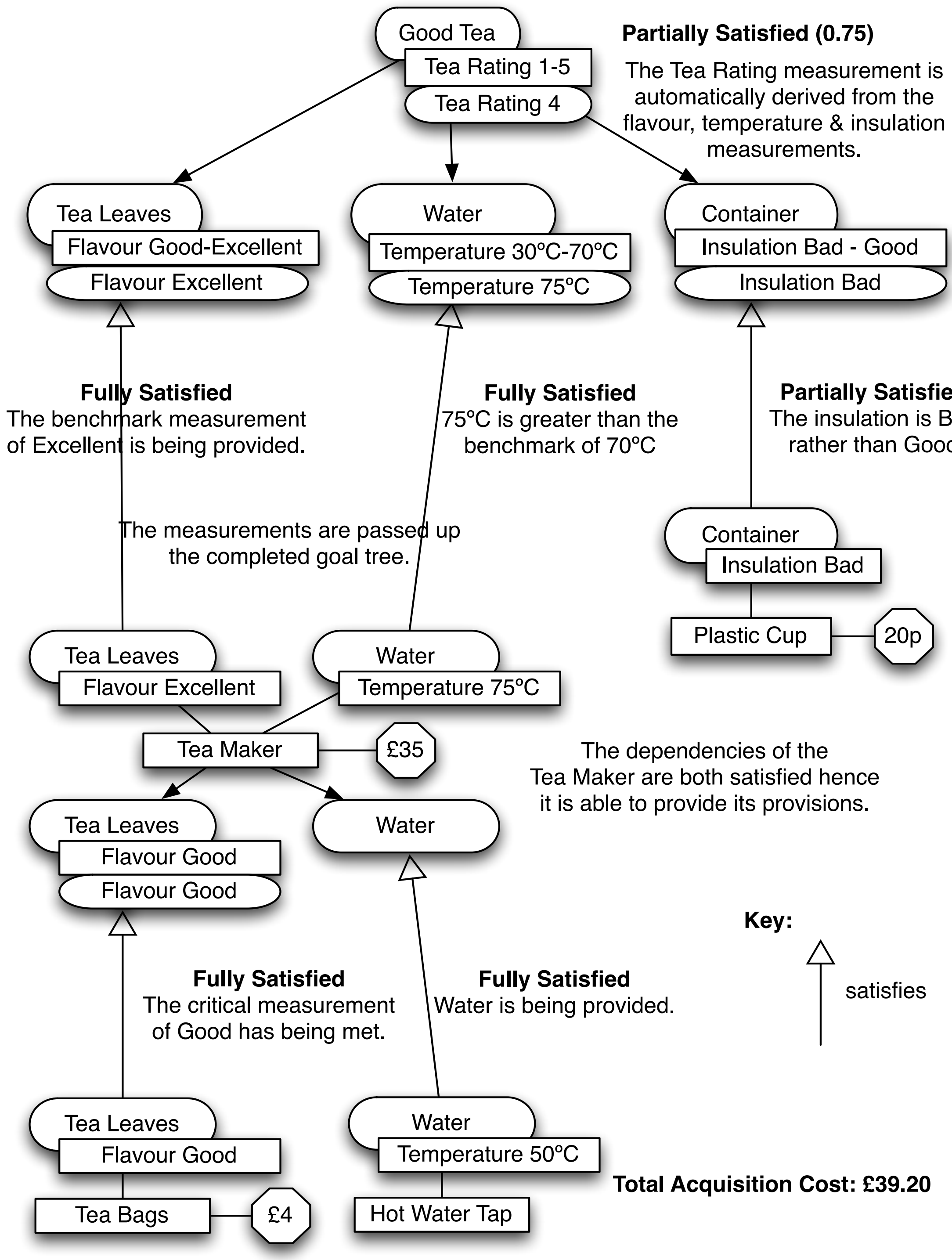

Good Tea
Tea Rating 1-5
Tea Rating 4
Partially Satisfied (0.75)
The Tea Rating measurement is automatically derived from the flavour, temperature & insulation measurements.
Tea Leaves
Flavour Good-Excellent
Flavour Excellent
Water
Temperature 30°C-70°C
Temperature 75°C
Container
Insulation Bad - Good
Insulation Bad
Fully Satisfied
The benchmark measurement of Excellent is being provided.
Fully Satisfied
75°C is greater than the benchmark of 70°C
Partially Satisfied
The insulation is Bad rather than Good
The measurements are passed up the completed goal tree.
Container
Insulation Bad
Plastic Cup
20p
Tea Leaves
Flavour Excellent
Water
Temperature 75°C
Tea Maker
£35
The dependencies of the Tea Maker are both satisfied hence it is able to provide its provisions.
Tea Leaves
Flavour Good
Flavour Good
Water
Key:
satisfies
Fully Satisfied
The critical measurement of Good has being met.
Fully Satisfied
Water is being provided.
Tea Leaves
Flavour Good
Tea Bags
£4
Water
Temperature 50°C
Hot Water Tap
Total Acquisition Cost: £39.20

Figure 7.3: Tea Making Example, Completed Goal Model

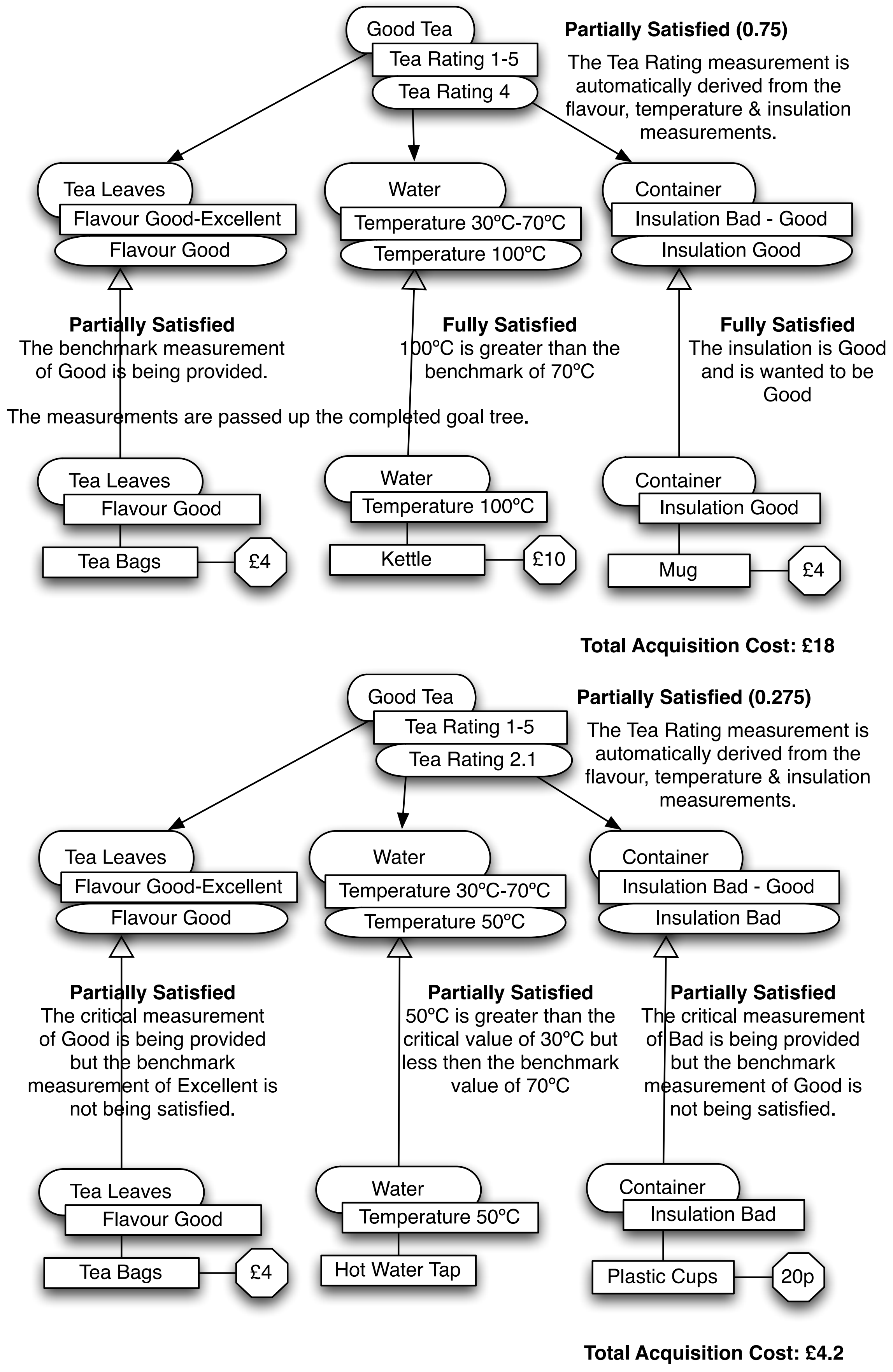

Good Tea
Tea Rating 1-5
Tea Rating 4
Partially Satisfied (0.75)
The Tea Rating measurement is automatically derived from the flavour, temperature & insulation measurements.
Tea Leaves
Flavour Good-Excellent
Flavour Good
Water
Temperature 30°C-70°C
Temperature 100°C
Container
Insulation Bad - Good
Insulation Good
Partially Satisfied
The benchmark measurement of Good is being provided.
Fully Satisfied
100°C is greater than the benchmark of 70°C
Fully Satisfied
The insulation is Good and is wanted to be Good
The measurements are passed up the completed goal tree.
Tea Leaves
Flavour Good
Tea Bags
£4
Water
Temperature 100°C
Kettle
£10
Container
Insulation Good
Mug
£4
Total Acquisition Cost: £18
Good Tea
Tea Rating 1-5
Tea Rating 2.1
Partially Satisfied (0.275)
The Tea Rating measurement is automatically derived from the flavour, temperature & insulation measurements.
Tea Leaves
Flavour Good-Excellent
Flavour Good
Water
Temperature 30°C-70°C
Temperature 50°C
Container
Insulation Bad - Good
Insulation Bad
Partially Satisfied
The critical measurement of Good is being provided but the benchmark measurement of Excellent is not being satisfied.
Partially Satisfied
50°C is greater than the critical value of 30°C but less then the benchmark value of 70°C
Partially Satisfied
The critical measurement of Bad is being provided but the benchmark measurement of Good is not being satisfied.
Tea Leaves
Flavour Good
Tea Bags
£4
Water
Temperature 50°C
Hot Water Tap
Container
Insulation Bad
Plastic Cups
20p
Total Acquisition Cost: £4.2

Figure 7.4: Tea Making Example, Additional solutions generated from the same problem description

7.4 Tea Making Textual DSL – Grammar

The CATMOS technique requires the user to provide information about the top-level capabilities and how they decompose along with information about each of the existing components and acquirable components that can be used in any potential solution. The information can be provided in more than one way to the tooling but for explanatory purposes we will now show how the information can be provided in a textual format. In the following section we explain both the grammar for the textual format and in the next section we show the Tea making example rewritten using the textual format. The full formal grammar definition for the CATMOS textual syntax is given in appendix A.

The keywords used in the grammar given below are shown in bold. Places where the user can enter data with a description of the wanted data is shown with < and > brackets. Normal sized ( ) brackets are used in conjunction with ? for marking an optional section and with + for marking a section that can appear 1 or more times and with * for marking a section that can appear 0 or more times in the grammar. Larger ( ) brackets have being used in the few places that ( ) brackets occur in the grammar to differentiate them from the section markers.

7.4.1 Problem Overview Information

Each problem definition in the grammar requires either a single *FindTradeOffs* block, which will be explained here, or alternatively a single *ThroughLifePlanning block*, which will be explained in section 9.4. The block is responsible for providing the tooling with basic information about the type of search to perform, which components to use and how many and whether the various costs should be minimised or maximised.

```
FindTradeOffs <scenarioName> {
    popSize <searchPopulationCount> genCount <searchGenerationCount>
    (ExistingComponent <componentName> <quantity>)*
    (AcquirableComponent <componentName> <quantity>)*
    (DesireLow <costType>)*
    (DesireHigh <costType>)*
}
```

Figure 7.5: FindTradeOffs Block Grammar

The usage of the *FindTradeOffs* blocks tells the tooling to generate a Pareto front of results of the designated search objective capabilities against the designated costs.

The *searchPopulationCount* and *searchGenerationCount* values are passed directly to the multi-objective search algorithm and control the amount of computation effort that is to be placed into the search.

The *ExistingComponent* line contains the name of a *Component*, which will be defined later on in the problem definition, and how many of that *Component* already exists and can therefore be used in an acquisition solution at no cost. The *Cost* information on a *Component* included this way is ignored. Any number of *ExistingComponents* can be defined.

The *AcquirableComponent* line contains the name of a *Component*, which will be defined later on in the problem definition, and how many of that *Component* can be acquired in an acquisition solution. Unlike the *ExistingComponents* line *Components* included this way will have their costs added to the total costs of the acquisition solution. Any number of AcquirableComponents can be defined.

The *DesireLow* and *DesireHigh* lines specify that a *Cost* is to be either minimised or maximised during the multi-objective search. An example of this is you may wish to minimise the amount of money used during an acquisition but maximise the amount of billable man hours used during the same acquisition. The default setting for a *Cost* is for it to be ignored during the multi-objective search. It is an error for a *Cost* to be both desired high and low.

7.4.2 Representing the top-level capabilities and their decomposition

The top-level capabilities that describe the problem are defined using the grammar block shown below with the grammar block being included once for each wanted capability.

```
(standAlone)?
(searchObjective)?
Capability <capabilityName> {
        (Measurement <measurementName> {
                (criticalValue <realNumber>
                benchmarkValue <realNumber>)
                |
                (criticalValues (<stringValue>(,<stringValue>)*)
                benchmarkValues (<stringValue>(,<stringValue>)*) )
                (script <script>)?
                }
        )*
        (decomposes (<capabilityName>(,<capabilityName>*)))?
}
```

Figure 7.6: Capability Block Grammar

Capabilities are defined with the *Capability* keyword and require a *capabilityName*. A *Capability* can be preceded by the *standAlone* and *searchObjective* modifiers. The *searchObjective* modifier designates the capability as a search objective to become one of the objectives for the multi-objective search.

The *standAlone* modifier states that the *Capability* is not satisfied by any of the *Components* directly. This is useful when used in conjunction with scripts for evaluating properties that belong to the overall acquisition solution rather than just a single *Component*. Consider an aircraft made out of individual components and each of the components has a weight. A *Capability* designated as *standAlone* can be used in conjunction with a script to collect together the weights of every *Component* in an acquisition solution to allow the overall weight of the airplane to be evaluated. The *standAlone Capability* will still be evaluated even through it is not being directly satisfied by any *Component's CapabilityProvision*.

Each *Capability* can have any number of *Measurements*. Each *Measurement* has a critical value and a benchmark value. The critical value is the minimum required for that *Measurement* to be considered partially satisfied and the benchmark value is the minimal required for the *Measurement* to be considered fully satisfied.

Each *Measurement* can optionally have a script attached. The script is evaluated when the *Measurement's Capability* is evaluated and is used to change the satisfaction level of the *Measurement*. Lua is used in the prototype tooling however any Turing complete scripting language is appropriate.

Measurements can be either quantitative where they deal with real number values or qualitative where they deal with qualitative statements but each individual *Measurement* can only be one of the two types. Qualitative statements can be set to be either higher or lower values than other qualitative statements as shown later on. Note the larger curved brackets used in the qualitative statement lines (8 & 9) are part of the grammar itself.

A *Capability* can be set to decompose into other sub-capabilities. This relationship between *Capabilities* will be shown when the results are displayed graphically. Mechanically, it behaves the same as the *standAlone* modifier, as it is assumed that the user will create *Measurements* on the *Capability* with scripts that aggregate the values of its sub-capabilities together.

7.4.3 Component descriptions

Each *Component* block describes a single *Component*. Any number of *Component* blocks can be defined. *Components* are only considered if a corresponding entry exists for them in the *FindTradeOffs* or *ThroughLifePlanning* block indicating that they are either an existing component, an acquirable component or both.

```
Component <componentName> {
    (CapabilityProvision <capabilityName> {
        (reuse <reuseTimes>)?
        (Measurement <measurementName> {
            ( providedValue <realNumber> )
            | ( providedValues (<stringValue>(,<stringValue>)*) )
            (script <script>)?
        })*
    })*

    (Capability <capabilityName> {
        (Measurement <measurementName> {
            ( criticalValue <realNumber>
            benchmarkValue <realNumber> ) |
            ( criticalValues (<stringValue>(,<stringValue>)*)
            benchmarkValues (<stringValue>(,<stringValue>)*) )
            (script <script>)?
        } )*
    } )*

    (Cost <costType> <costAmount>)*
}
```

Figure 7.7: Component Block Grammar

Each *Component* must have a componentName and can have any number of *CapabilityProvisions*, *Capabilities* and *Costs* in that order.

A *CapabilityProvision* has a name, which must be the same as the *Capability* it intends to provide for. Along with any number of *Measurements* that should match the *Measurements* on the *Capability* the *CapabilityProvision* intends to provide for. *Measurements* in a *CapabilityProvision* have a provided value, instead of a critical value and benchmark value, which represents the value provided by that *Component* for satisfying that *Capability*. As before in the top-level *Capabilities* the *Measurements* can either be quantitative or qualitative but not both at the same time. Note the larger curved brackets used in the qualitative statement lines (6, 15 & 16) are part of the grammar itself. A *CapabilityProvision* can optionally have a *reuse* line that states the maximum number of Capability instances that it can satisfy. No *reuse* line or a *reuse* line of 0 indicate that the *CapabilityProvision* can be used an unlimited amount of times.

The *Capabilities* on the *Component* are defined in the same way as they are defined as part of the top-level capabilities except they cannot be designated as *standAlone* or as a *searchObjectives* and they cannot decompose into other sub-capabilities.

A *Cost* is defined to be of a certain type (i.e. money, man-hours, weight) and have an amount, which is a real-numbered value.

7.4.4 Qualitative Values

There is a need when using qualitative values to establish a partial ordering on the qualitative values to state that the values are greater or smaller than each other. For example if a transport works in “Heavy Rain conditions” than it can be assumed to also work in Light Rain conditions. These relationships are described using the *Value* keyword.

```
Value <stringValue> (<|>) <stringValue>
```

Figure 7.8: Qualitative Value statement

The format is Value A >B or Value A <B to define A is greater than B or A is less than B respectively. The relationships are used when evaluating qualitative measurements satisfaction.

7.5 Tea Making Textual DSL - Example

The textual domain specific language (DSL) input is as follows:

```
FindTradeOffs TeaMaking { popSize 100 genCount 100
        ExistingComponent "Cold Water Tap" 1 ExistingComponent "Hot Water
            Tap" 1
        AcquirableComponent "Tea Bags" 1 AcquirableComponent "Mug" 1
        AcquirableComponent "Kettle" 1 AcquirableComponent "Plastic Cup" 1
        AcquirableComponent "Tea Maker" 1
        DesireLow "Money"
}

searchObjective
```

```
Capability "Good Tea" {
        Measurement "Tea Rating" { criticalValue 1.0 benchmarkValue 5.0 script "output = TeaRating()" }
        decomposes ("Tea", "Water", "Container")}

Capability "Tea" {
        Measurement "Flavour" { criticalValues ("Good") benchmarkValues ("Excellent") }}

Capability "Water" {
        Measurement "Temperature" { criticalValue 30.0 benchmarkValue 70.0 } }

Capability "Container" {
        Measurement "Insulation" { criticalValues ("Bad") benchmarkValues ("Good")}}

Component "Tea Bags" {
        CapabilityProvision "Tea" { Measurement "Flavour" {providedValues ("Good") }}
        Cost Money 4.0 }

Component "Hot Water Tap" {
        CapabilityProvision "Water" { Measurement "Temperature" { providedValue 50.0 }}}

Component "Mug" {
        CapabilityProvision "Container" { Measurement "Insulation" { providedValues ("Good")}}
        Cost Money 4.0 }

Component "Cold Water Tap" {
        CapabilityProvision "Water" {Measurement "Temperature" { providedValue 10.0}}
        }

Component "Kettle" {
        CapabilityProvision "Water" {Measurement "Temperature" { providedValue 100.0}}
        Capability "Water" {}
        Cost Money 10.0 }

Component "Plastic Cup" {
        CapabilityProvision "Container" { Measurement "Insulation" { providedValues ("Bad")}}
        Cost Money 0.2 }

Component "Tea Maker" {
        CapabilityProvision "Water" {Measurement "Temperature" { providedValue 75.0}}
        CapabilityProvision "Tea" { Measurement "Flavour" {providedValues ("Excellent") }}
        Capability "Water" {}
        Capability "Tea" {}
        Cost Money 35.0 }

Value "Excellent" > "Good" Value "Good" > "Bad" Value "Bad"
```

7.6 Tea Making - Scripts

As described previously in section 7.2.2, it must be possible to specify a script alongside the textural domain specific language for specifying the complex relationships between the capabilities in the top-level decomposition.

In this case, the script has been specified using Lua. In the Lua script the function TeaRating() is defined that specifies how a rating for tea can be derived from the temperature, flavour and the insulation of the container of the tea. In this case, two hooks into the goal model are being used. *getScenarioMeasure* gets the value of a measurement attached to a capability in the goal model. *canUseCapability* returns whether the capability can be used i.e. the capability is satisfied and all of the capabilities on the satisfying component are also satisfied and so are all of the capabilities on the components that satisfy the satisfying capability and so on recursively. Other hooks are available and are defined in section 10.3.5.

```lua
--Tea Rating Function - Domain Specific Knowledge of "Good Tea"
function TeaRating()
        --Get Measurements From Completed Goal Model
        temperature = getScenarioMeasure("Temperature")
        flavour = getScenarioMeasure("Flavour")
        insulation = getScenarioMeasure("Insulation")

        score = 0.0

        --Temperature over 70 degrees doesn't help
        if (temperature > 70.0) then temperature = 70.0 end

        --Score based on Temperature
        score = temperature / 70.0 * 3.0

        --If Flavour Is Excellent Add 1 To Score
        for a,b in pairs(flavour) do
                if (b == "Excellent") then
                        score = score + 1.0
                end
        end

        --If in Insulated Cup Add 1 To Score
        for a,b in pairs(insulation) do
                if (b == "Good") then
                        score = score + 1.0
                end
        end

        --If any of the Tea, Water or Container is missing Score 0.
        if (canUseCapability("Tea") == 0 or canUseCapability("Water") == 0
                or canUseCapability("Container") == 0) then
                score = 0.0
        end

        return score
end
```

7.7 Tea Making Example - Pareto Front

Using techniques described in detail in the next chapter, CATMOS is able to generate a graph showing the highest satisfaction for the 'Good Tea' goal at different costs (figure 7.9). Each point on the graph refers to a different possible acquisition solution such as those shown in figures 7.3 & 7.4. All these different acquisition solutions have been generated from the same problem input.

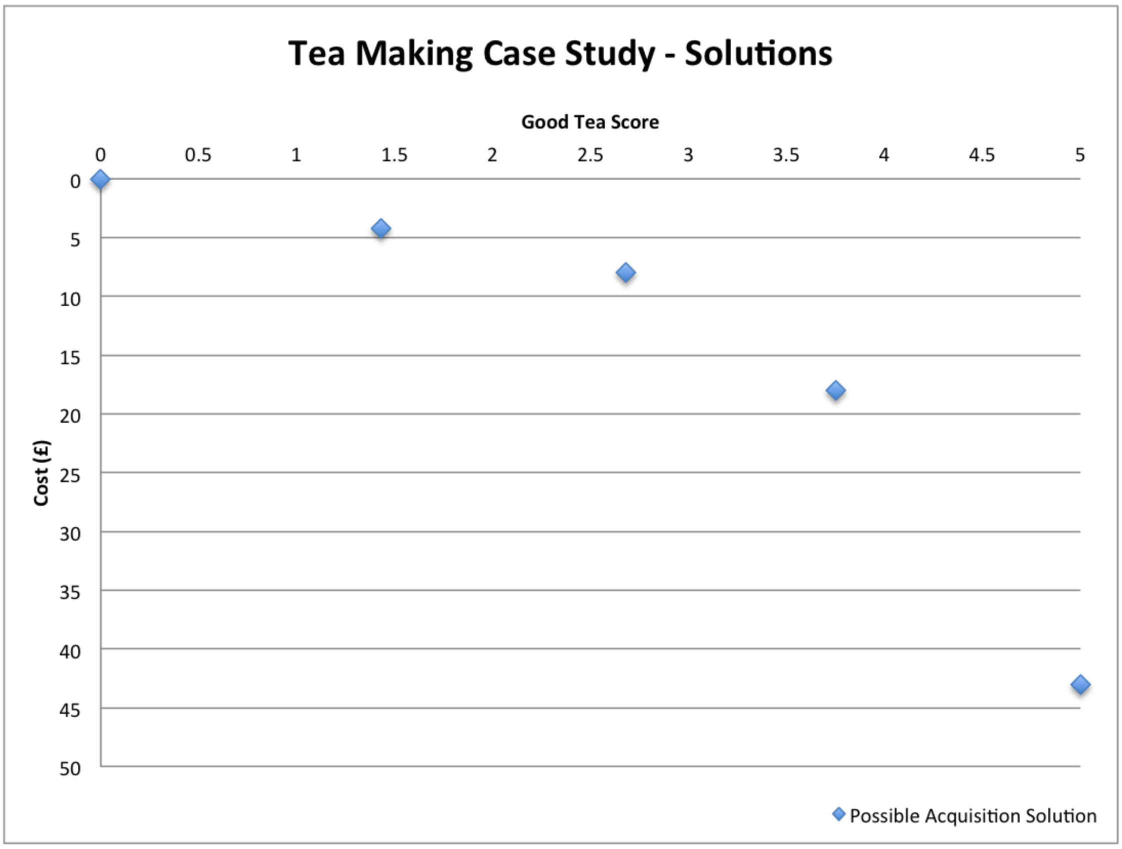


Figure 7.9: Tea Making Case Study - Pareto Front

The graph shows on the top axis the highest 'Good Tea Score' found for the costs shown on the left axis. There are five solutions shown and these are only the only generated solutions that should be considered by the decision maker because all the other solutions will either have a higher cost or a lower 'Good Tea Score' than the solutions shown on the graph.

7.8 Addressing Research Gap 1 - Technique Features

Whilst the basic features of CATMOS have been described already, we have avoided going into full detail for all of the features for clarity reasons. We now do so, before the next chapter moves on to discuss how multi-objective search is applied to the CATMOS technique.

7.8.1 Complex Goal-Tree Decompositions

The first feature to be discussed is handling of complex goal-tree decomposition. Capabilities in TLCM are considered to decompose both by function and by area [23]. A functional decomposition of a capability breaks it into smaller capabilities that represent smaller tasks. When put together, these will lead to the completion of the larger capability. An area decomposition of a capability breaks the capability into multiple sub-capabilities, one for each of the environments the capability operates in. For example 'Scouting' may be decomposed into 'Scouting At Sea' and 'Scouting On Land'. The leaf capabilities are given direct measurements by satisfying systems; however the capabilities that have been decomposed need to somehow aggregate their child capabilities' measurements. In practice these aggregations can be complicated, so simply providing a weighted sum or averaging feature wouldn't be sufficient.

To allow these aggregations to be performed, measurements are permitted to have attached annotations that allow them to define their value in relation to the values of other capabilities, measurements or external data sources. These annotations are currently being performed using Lua [128, 129] though other programming languages or formalisations could be used. The requirements on a formalism to be used for this depend on the application domain that the CATMOS technique is being applied to. The Lua annotations are provided with multiple functions that hook into the goal model allowing them to read values from the current goal model structure.

In capability based acquisitions, there are both Measures of Performance (section 2.4) and Measures of Effectiveness (section 2.5). There is a need to convert between the Measures of Performance, which measure the systems directly, and Measures of Effectiveness, which measure how well the system performs within a scenario. In the Tea making example, the Tea rating, which is a Measure of Effectiveness, is based on the tea's flavour, the tea's temperature and whether or not the container is insulated, which are all Measures of Performance. How to convert between the Measures of Performance, which will be found on the leaf capabilities, to the Measures of Effectiveness, which will be found on the higher level capabilities in the *Goal Tree*, in general is domain specific knowledge and requires expertise. This conversation may be done using simple formulas from domain experts (as is done in work done under the NECTISE project [83, 134]), existing datasets or full simulations. This means that the method provided must be flexible, which is why a full programming language is used.

This feature is required for helping to address research gap 1, in relating to the satisfaction of the higher level military capabilities to the acquisitions of the concrete things within the Defence Lines of Development (DLoD). The feature is used by the Tea making case study, the Multi-objective Next Release Problem case study in the next chapter and the realistic military case study used in chapter 9.

7.8.2 Partial dependency satisfaction

The next feature to be discussed is handling of partial dependency satisfaction. This is where a component's dependencies are only partially met. Consider a fire hose that relies on a water hydrant to provide water pressure to allow it put out fires. The ability of the fire hose to put out fires depends heavily on the water pressure being provided by the water hydrant. There maybe existing simulations or formulas that can be used to establish how well a fire hose can put out a fire at different water pressures.

The measurement annotations discussed above can also be used to allow a component's capability provisions to be defined in terms of the satisfaction of the component's dependencies. Additionally, the annotations can also be used to interface with external simulations or pre-generated datasets for supporting more complicated relationships.

This feature is required to allow the effects of only partially fulfilling the dependencies of a component within a system of systems has on the component's performance. In a system of systems, the systems involved will inevitability depend on each other to operate correctly. This feature helps address research gap 1 by allowing the effects of the dependencies between the various components, which are categorised by the DLoD, to be modelled so the effect it has on the produced capabilities can be handled by the technique.

7.8.3 System of systems properties

Another feature of interest is system of systems properties. A system of systems property is a property that is the result of multiple systems being used together. A basic example of this is the total power usage of the system of systems is the sum of all the systems power usage or alternatively the weight of an aircraft is the sum of the weight of its parts.

There needs to be a way in the CATMOS technique to support trade offs against these system of systems properties. This is supported by allowing capabilities to be marked as *standAlone*, meaning that it can be satisfied without any joining capability provisions. This is used in conjunction with a script on a measurement in the capability to determine its level of satisfaction. In the two examples just explained of power usage and aircraft weight a script can use the getAllMeasures() hook to find out the total power usage or weight of the systems. The power usage and weight of individual systems can just be specified as measurements attached to one of the component's capability provisions.

Whilst the concept of system of systems may refer to entire armed forces or parts thereof, it can also refer to systems such as Aircraft, which are composed out of a large number of separate systems being used together. In this case, the concept of properties belonging to the entire system of systems is useful. There are also scenarios where things such as the total response time of multiple systems may wish to be measured. Measuring things like the total response time can already be done without this feature using the existing capability decompositions however this feature is still useful syntactic sugar in this case. This feature helps address research gap 1, bridging the gap between the DLoD and capabilities, in some edge cases and otherwise is useful syntactic sugar for the acquisition decision makers to use.

7.8.4 Capability Upgrades

Capability upgrades are supported by the CATMOS technique to cover situations where one system is acquired to modify the capabilities of another system. For example, a sniper's rifle could be fitted with a laser sight to improve its accuracy. A component can contain capabilities upgrades that describe the changes that component does to other components in the system of systems. A capability upgrade targets a specific component and can add new capabilities, modify the measurements on existing capabilities or delete existing capabilities from the specified component.

In military acquisition, the acquisition of modifications to existing systems to improve capabilities on systems is commonplace and upgrading the capabilities provided by a system is supported even by existing weighted sum based approaches [25]. Since the acquisition of modifications to systems to improve capabilities is commonplace, the CATMOS technique also needs to support this for helping to address research gap 1.

7.8.5 Capability Accumulations

Capability accumulations handle the situation when multiple providing systems add together to form the same capability. This is best explained with an example. Consider a 'Fire extinguishing' capability belonging to a 'Fire truck'. The 'Fire extinguishing' capability increases depending on the number of 'Fire trucks' at the scene of the fire. The more 'Fire trucks' the larger the fire that can be put out. This is dealt with in CATMOS by using the accumulation feature. A measurement on the capability is set to be the accumulation measurement. Once this is done, any number of capability provisions can be used in an acquisition solution to satisfy the same capability. The accumulation measurement for example 'Gallons of water per second' takes the sum of all the satisfying capability provisions measurements. Capability accumulations cannot be supported by just annotating measurements in the goal model because it affects the structure of the produced goal models in allow the same capability to be satisfied by multiple different solutions simultaneously.

When dealing with real world scenarios such as demonstrated by the realistic military case study in section 9.5, there is more to consider than just the question of whether or not a capability can be achieved such as 'Performing an artillery strike at range' but also how much of this capability is available for usage. Capability accumulation allows the concept of how much of a capability is available to be considered and whilst it does not see usage in the Tea making example or the Multi-Objective Next Release Problem it is made extensive use of in the realistic military case study. This feature helps address research gap 1 by allowing the amount of a capability that is produced by the components categorised by the DLoD to be considered.

7.9 CATMOS DSL Additional Notations excluding Through Life

Following on from the explanation given in section 7.4, we will now explain the remaining grammar and syntax for the CATMOS DSL excluding the through life extensions covered in the next chapter. The full formal grammar definition for the CATMOS textual syntax is given in appendix A and is defined using Xtext [133].

The keywords used in the grammar given below are shown in bold. Places where the user can enter data with a description of the wanted data are shown with < and > brackets. Normal sized ( ) brackets are used in conjunction with ? for marking an optional section and with + for marking a section that can appear 1 or more times and with * for marking a section that can appear 0 or more times in the grammar. Larger ( ) brackets have being used in the few places that ( ) brackets occur in the grammar to differentiate them from the section markers.

7.9.1 Scripting behaviour

Consider the following definition of a *Capability*:

```
standAlone
searchObjective
Capability capabilityName {
    Measurement myQuantitativeMeasurement {
        criticalValue 50.0
        benchmarkValue 350.0
        script "output = 250"
    }
}
```

The *Capability* is considered to be *standAlone* so nothing will directly satisfy it. The *Capability* is also considered to be a *searchObjective* so its value will be maximised during the multi-objective search. The *Capability* in this example gains its satisfaction level from the included *Measurement*, which because it will never have a *providedValue* given to it from a *CapabilityProvision*, it needs to set this value so it can be evaluated. This is done using the script in the *Measurement*. In this simple case, the script sets the *Measurement's* effective *providedValue* to 250.0 by setting the output variable. This is then assessed between the critical value of 50.0 and the benchmark value of 350.0 to give an overall satisfaction of 67% ((250.0-50.0)/(350.0-50.0)). Scripts placed on *Measurements* in *CapabilityProvisions* that change the output value override the existing *providedValue* on the *Measurement* before it is passed up through the satisfies relationship to a *Capability*.

7.9.2 Capability Upgrades

The grammar and syntax for the capability upgrades has not yet been defined. This is an extension to the *Component* grammar block and the new extended *Component* grammar block is shown in figure 7.10.

In the new extended grammar, *CapabilityUpgrades* are placed within *Components*. They have a name for the upgrade and a target component that they are going to change in some manner. They also have a capability change type, which can be set to add, del or mod, for adding new *CapabilityProvisions*, deleting existing *CapabilityProvisions* or modifying existing *CapabilityProvisions* on the target component. The *CapabilityChange* contains a set of *CapabilityProvisions*, which will be used by the desired change. When adding the *CapabilityProvisions* are added whole to the target component. When deleting only the *CapabilityProvisions* name is used for selecting the *CapabilityProvision* to delete. When modifying the *CapabilityProvisions* name is used for selecting

```
Component <componentName> {
    (CapabilityProvision <capabilityName> {
        (reuse <reuseTimes>)?
        (Measurement <measurementName> {
            ( providedValue <realNumber> )
            | ( providedValues (<stringValue>(,<stringValue>)*) )
            (script <script>)?
        })*
    })*

    (CapabilityUpgrade <upgradeName> {
        targetComponent <targetComponentName>
        (CapabilityChange <changeType> {
            CapabilityProvision <changedCapabilityProvisionName> {
            (reuse <reuseTimes>)?
            (Measurement <measurementName> {
            ( providedValue <realNumber> )
                | ( providedValues (<stringValue>(,<stringValue>)*) )
                (script <script>)?
            })*
        })*
    })*

    (Capability <capabilityName> {
        (Measurement <measurementName> {
            ( criticalValue <realNumber>
            benchmarkValue <realNumber> ) |
            ( criticalValues (<stringValue>(,<stringValue>)*)
            benchmarkValues (<stringValue>(,<stringValue>)*) )
            (script <script>)?
        } )*
    } )*

    (Cost <costType> <costAmount>)*
}
```

Figure 7.10: Component Block Grammar

the corresponding *CapabilityProvision* on the target component and the all of its *Measurements* values are updated to the new values where applicable.

7.9.3 Capability Accumulations

The grammar and syntax for capability accumulations, introduced in section 7.8.5, have not yet been introduced. Both capabilities in the *Goal Tree* and on *Components* as dependencies can be turned into a capability accumulation by adding the single optional *accumulation* line. An example is:

```
Capability "Fire extinguishing" {
    accumulation "Water Gallons per second"
    Measurement "Water Gallons per second" {
        criticalValue 5.0
        benchmarkValue 50.0
    }
}
```

The use of the *accumulation* line allows the *Capability* to be satisfied any number of times by *CapabilityProvisions*. Capability accumulations allow the consideration of quantities of acquired *Components* to be considered. For example 5 'Fire Trucks' providing 5 water gallons per second will provide 25 water gallons per second towards the *Fire extinguishing* capability shown above. A *Measurement* needs to be specified on the *accumulation* line and this specified *Measurement* will be evaluated using the summation of all the *providedValues* on all the *CapabilityProvisions* attached to it. *CapabilityProvisions* reuse values are taken into account hence if a *CapabilityProvision* can be used twice it is able to provide double the value of its *Measurement* value (the named measurement in the accumulation) to the *Capability*.

7.10 Summary

The research shown so far addresses research gap 1. The desired capabilities can be defined in the *Goal Tree* model and the acquisition programmes in the Defence Lines of Development can be defined as *Component* models. Then by using *Capability* as a common joining concept the individual models can be brought together in different ways to create different acquisition plans. These acquisition plans can then be evaluated by evaluating the contained measurements on the *Capabilities* and *Capability Provisions* and checking the dependency satisfaction of the *Components*. Different acquisition plans can be created from the same models by including different models in the final solution and changing the structure of how the models connect together.

This chapter also showed that by adding capability provisions, adding component dependencies, attaching costs to components, modularisation of components, etc. an effective technique can be created for handling alternative goal tree derivations. This is demonstrated with multiple goal models for the tea making example given in figures 7.3 & 7.4. Alternative goal tree derivations is a recognised research area in goal modelling by Lamsweerde [24] but currently little to no work has been done in exploring this research area.

In the next chapter, how to automatically generate large numbers of these *Goal models* that represent different trade-offs for the stakeholders will be covered. The technique presented so far enables the process of mapping from a set of desired capabilities to a set of acquisition programmes, which are categorised by the DLoD.

Chapter 8

Multi-objective Acquisition Trade-offs

In the previous chapter, we described a technique that relates a set of acquisition programmes to a set of desired capabilities. The technique operates by taking a *Goal Tree* model and multiple *Component* models and combines them together to create an evaluable acquisition plan. What was not described is how to combine the models to produce ‘good’ solutions or even what a ‘good’ solution was.

What constitutes a ‘good’ solution? For the purposes of Through Life Capability Management, a ‘good’ solution is one that maximises the satisfaction of the system of systems objectives, whilst also minimising the costs. Increasing the satisfaction of the desired objectives generally involves acquiring more systems and incurring more costs putting the two objectives of increasing satisfaction and reducing costs against each other meaning that a trade-off needs to be found between these two objectives.

It is however possible to keep the same level of satisfaction whilst reducing costs or keep the same costs whilst increasing the level of satisfaction by finding more efficiently composed solutions. At maximum efficiency these solutions are called Pareto optimal solutions.

Some of the research ideas that contribute to this chapter have already been published by the author in [3], which covers applying the CATMOS technique to the Multi-objective Next Release Problem (MONRP).

8.1 Pareto optimality

To precisely define a ‘good’ solution we need to introduce a well-established concept from the field of economics: Pareto optimality. Pareto optimality can be applied to any situation where there are multiple competing objectives; in this case, maximising all the capabilities whilst minimising all the costs. A solution to a multi-objective problem is considered to be Pareto optimal when there are no other solutions that dominate it. A solution is considered to dominate another solution when it has a higher value for one of the objectives and for all other objectives it does not have a lower value than the solution it’s dominating.

An example of Pareto optimality on two objectives is shown in figure 8.1. All of the Pareto optimal points are considered to make up the Pareto front. For each of the Pareto optimal solutions, there is no other solution that is better than them for both of the objectives. The objective for each solution is to simultaneously maximise the values for both objective A and objective B. For the non

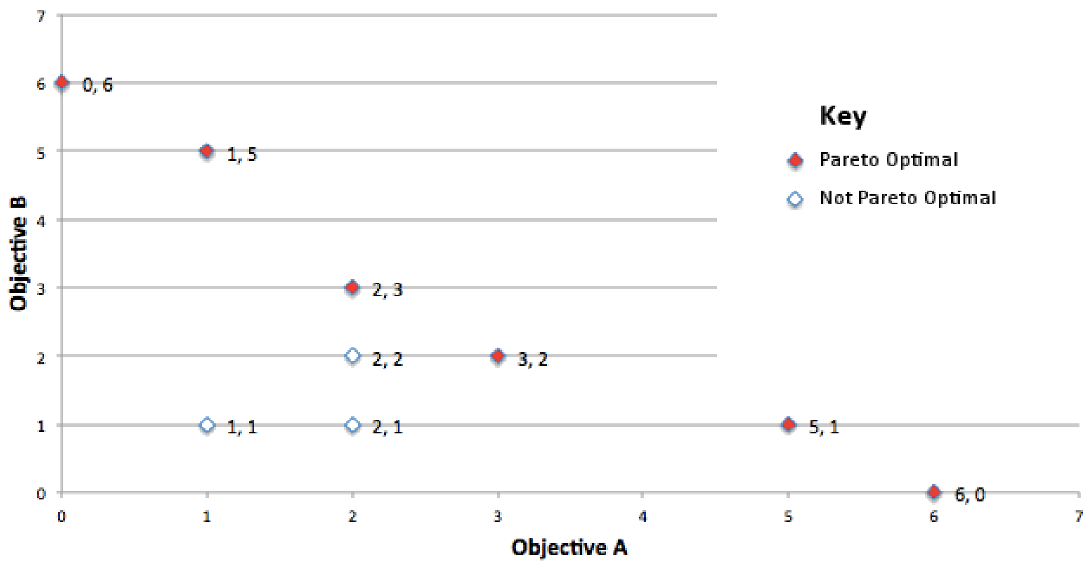


Figure 8.1: Pareto Optimality Example

Pareto optimal solutions there is another solution that is better than them for both of the objectives. For the non Pareto optimal solutions: (2,2) is dominated by (2,3) and (3,2). (2,1) is dominated by (2,2), (2,3), (3,2) and (5,1). (1,1) is dominated by (1,5), (2,1), (2,3), (2,2), (2,1), (3,2) and (5,1).

The concept of Pareto optimality is useful because if we take all the possible acquisition solutions created from all combinations of possible *components*, we can discard any solution that is not Pareto optimal: there will be a solution that is Pareto optimal that is strictly better than it present in the solution set.

Research gap 2 addresses the multi-objective nature present in an acquisition problem, and on giving acquisition decision makers the ability to make trade-offs between the various organisational goals and costs to the organisation. To address this problem, we can present the acquisition decision makers the Pareto optimal solutions, which are all 'optimal' against the various objectives; they can then choose the exact trade-offs they want to make.

The first issue is how to calculate these Pareto optimal solutions. Even on a moderate size problem the number of combinations of *Components* and their inter-connections is sufficiently large to be computationally infeasible to compute. For the realistic military case study, introduced in section 8.3, leaving aside the inter-connections between the *Components*, just determining how many of each *Component* should be acquired forms an initial lower bound for the search space of 3.2 million possible acquisition plans. This is without considering the different possible inter-connections between the *Components* that will further enlarge the search space greatly. For the simpler acquisition problem of the Multi-objective Next Release Problem (discussed in section 8.3) it is considered to be computationally infeasible to compute all the solutions [130].

A common solution to addressing problems where finding all answers is computationally infeasible is to use metaheuristic techniques that aim to find good solutions instead of the exact perfect solution [111]. Multi-objective search techniques extends metaheuristic techniques to handle cases where there are multiple competing objectives [135].

8.2 NSGA-II

Multi-objective search techniques (section 5.5.1) are designed to addresses combinational problems with multiple objectives. In this case the multiple competing objectives are stakeholder objectives and the overall costs and the combinational part is the chosen acquisition solutions categorised by the DLoD and how they are linked together to produce a working solution. The widely used algorithm NSGA-II [119] is being used in this work as it specialises in generating Pareto fronts. The algorithm uses a novel selection function on top of a normal genetic algorithm (section 5.5) to cause the genetic algorithm population to evolve into a Pareto front of solutions instead of a single good solution. Multi-objective search algorithms are a type of meta-heuristic technique, which is a general strategy for solving combinational problems. It does not solve the problem in its own right. For each specific problem, multiple algorithms need to be defined to fill in missing parts of the overall strategy. [117]

As a brief outline, using the NSGA-II algorithm [119] requires the definition of:

- Genotype - A representation of the solutions amenable to search operators. In CATMOS this is a vector of triples representing the joins between the goal tree model and the component models.
- Phenotype - A representation of the solutions that can be evaluated. In CATMOS this is a completed goal model.
- Phenotype to Genotype mapping - A mapping that relates solutions in the form of a genotype into a corresponding solution using the form of a phenotype. In natural biology, this is the transformation between DNA (the genotype) and the animal or planet of interest (the phenotype) [117].
- Breeding operators - Taking the genotype of several ‘parent’ solutions, the breeding operators create new ‘child’ solutions based on their parents. In CATMOS this is a double point crossover operator, destructive mutation step and a repair step. Double point crossover splits the two parents genotype into three segments and creates a new genotype by combing the middle segment of one of the parents with the two side segment of the other [117]. The destructive mutation step being used and the repair step being used are CATMOS specific and will be explained in detail later on. However in general, the mutation step causes a small amount of random change to the child genotype to replicate the effects of random copying errors in DNA and the repair step fixes any errors caused by the crossover operator to make the child genotype represent a valid solution [117].
- Evaluation function - Accepts a phenotype as input and returns the degree to which the wanted objectives are met. This is evaluated in CATMOS using the method described in section 7.3.

A flowchart of the overall CATMOS technique including the NSGA-II search operators is given in figure 8.2. NSGA-II [119] is a genetic algorithm, inspired by biological evolution. The first thing that needs to be defined with such algorithms is the genotype for the problem. In nature

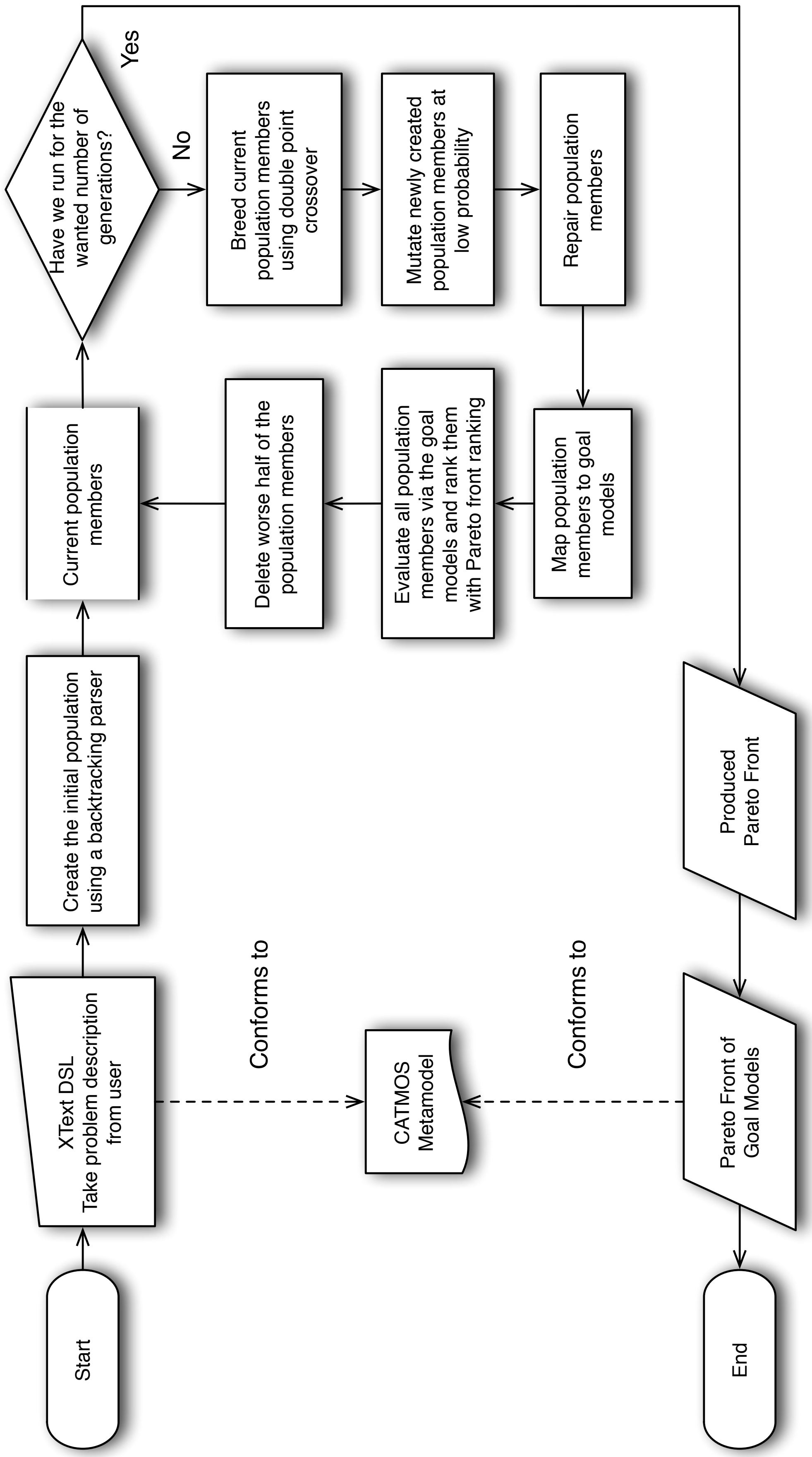

Start
XText DSL
Take problem description from user
Create the initial population using a backtracking parser
Current population members
Have we run for the wanted number of generations?
Yes
No
Breed current population members using double point crossover
Mutate newly created population members at low probability
Repair population members
Map population members to goal models
Evaluate all population members via the goal models and rank them with Pareto front ranking
Delete worse half of the population members
Conforms to
CATMOS Metamodel
Conforms to
Produced Pareto Front
Pareto Front of Goal Models
End

Figure 8.2: Flowchart for the CATMOS technique

the genotype is the DNA of an organism and the organism itself is called the phenotype. The computational equivalent of the genotype is usually a vector of values that can be in some way mapped to the phenotype, the actual artefact of interest. The criteria for the design of the genotype is that it is amenable to performing search operators. The most common breeding operator is crossover with two different genotypes: parts from both are taken and a new child genotype is formed, which is an easy operation to perform with vectors of triples. This is usually followed by mutation where at low probability a value in the genotype is randomly changed. Mutation occurs in actual biological DNA because the process by which the DNA strands are copied can make errors. [117]

NSGA-II [119] is an adaption of normal genetic algorithms designed for supporting the generation of Pareto fronts of solutions rather than just a single best solution. Genetic algorithms work by having a population of solutions and selecting the best solutions in the population via an evaluation function and breeding them together to produce the next generation of solutions. NSGA-II changes the evaluation function to use non-dominated ranking. Normally, each solution would be assigned a score directly from the evaluation function however in NSGA-II all the solution scores are first calculated then each solution is assigned a non-domination score. The non-domination score is how many other solutions dominate this solution per the description of domination above (lower scores being better). The selection step uses the non-domination scores to guide the genetic algorithm towards producing a Pareto front of solutions. NSGA-II contains a relatively fast algorithm for computing the non-domination scores. The other innovation in NSGA-II is the use of crowding distance, which is a method for determining how close each solution is on the Pareto front to other solutions on the Pareto front and penalising closeness of solutions during selection, to encourage solutions to spread out further along the Pareto front rather than all end up close together on a small part of it. [117, 119]

The reason why we want the solutions to be spread out on the Pareto front is this allows the acquisition decision makers to see a wide range of possible solutions to their problem allowing them to make a more informed choice. This also prevents the technique from converging to a single solution with the other solutions being only minor or trivial variants of each other that provide minimal extra information to the acquisition decision maker.

The genotype for this problem is being defined as a vector of joins. More specifically, the part of the problem we are interested in encoding is how the various *Components* are connected to the *Goal Tree* and the other *Components*.

The genotype is therefore defined as a vector of triples:

> <(sourceComponent, capability, targetComponent), (sourceComponent, capability, targetComponent), etc. >

The triple (sourceComponent, capability, targetComponent) indicates that the sourceComponent provides the stated capability to the targetComponent. This means that the sourceComponent has the capability as a *CapabilityProvision* and the targetComponent has the capability as a *Capability* and the satisfied-by relationship between the *Capability* and the *CapabilityProvision* is to be connected.

The genotype contains sufficient information to form the phenotype. When forming the phenotype, the entire *Goal Tree* is automatically included in the solution and every *Component* mentioned in the sourceComponent or the targetComponent of the triples is included into the solution (included in this sense means that the model is fully copied). The satisfied-by relationship between *CapabilityProvisions* and *Capabilities* in the *Goal Tree* is also made according to the triples. The keyword 'scenario' is used as the targetComponent to indicate that the connection is to the *Goal Tree* rather than another *Component*. The including into the solution of the *Goal Tree* and the individual *Component* and the creation of the satisfied-by relationship between these parts is demonstrated in the Tea making example in chapter 7 figures 7.2 & 7.3.

The genotype also contains an additional feature: *Components* that upgrade other *Components'* capabilities. This is supported by including triples (sourceComponent, upgradeName, targetComponent) in the vector along with the normal joining triples. The upgrade triples are evaluated first when forming the phenotype from the genotype as they alter the available *CapabilityProvisions*. The upgrades specified in the upgrade triples are applied from the sourceComponent to the targetComponent. This also creates a dependency between the two automatically. Upgrades can either add new capabilities on the target component, modify the measurements of existing capabilities on the target component or delete capabilities on the target component. The actions performed by an upgrade are contained within the *Component* model for the component providing the upgrade. There is no difference in the genotype relating to what actions the upgrade performs. It only contains the source and target of the upgrade.

An example of an application of an upgrade with the relevant components models, upgrade triples and the resulting model fragment for the produced *Goal Model* is shown in figure 8.2. In the example application, a *Car* is upgraded with *Air Conditioning* and a *Diesel engine*. Originally, the *Car* only provides *Transport* with the fuel type of petrol. The *Air Conditioning* upgrade adds the *Air Conditioning* capability to the *Car* and creates a dependency between the *Car* and it. The *Diesel Engine* modifies the existing *Transport* capability and changes its fuel type to diesel. This also adds a dependency between the *Car* and the *Diesel Engine*. The dependency is added to represent that without the *Diesel Engine* the *Car* is now non-functional. This handles cases such as if the *Diesel Engine* had as a dependency *Diesel Fuel*, the *Diesel Fuel* is now needed to move the *Car*.

The two breeding operators that are being used are double point crossover and mutation. Double point crossover is where two points along the two parent vectors are chosen, splitting them into three parts [136]. A child genotype is created from the two parent vectors by appending the left and right sections from one to the middle of the other [136]. Normally the vectors of elements used in crossover are of the same length, however in this case the vectors can and are likely to be different lengths depending on the number of connections used in the solution. This can have undesired side effects as when crossover is repeatedly applied the same triple will appear multiple times in the same solution. To counteract this, in CATMOS every unique triple of values is given a unique identifier to indicate its positions along a virtual chromosome. These unique identifiers are generated after the all the initial population members are created and are ordered (using whole numbers). The double point crossover takes place on this virtual chromosome. Since the triples effectively do not change position between crossovers on this virtual chromosome, the triples can not be included twice within a child solution regardless of the number of times the crossover operator is applied. An example of the double point crossover using the virtual chromosome is shown

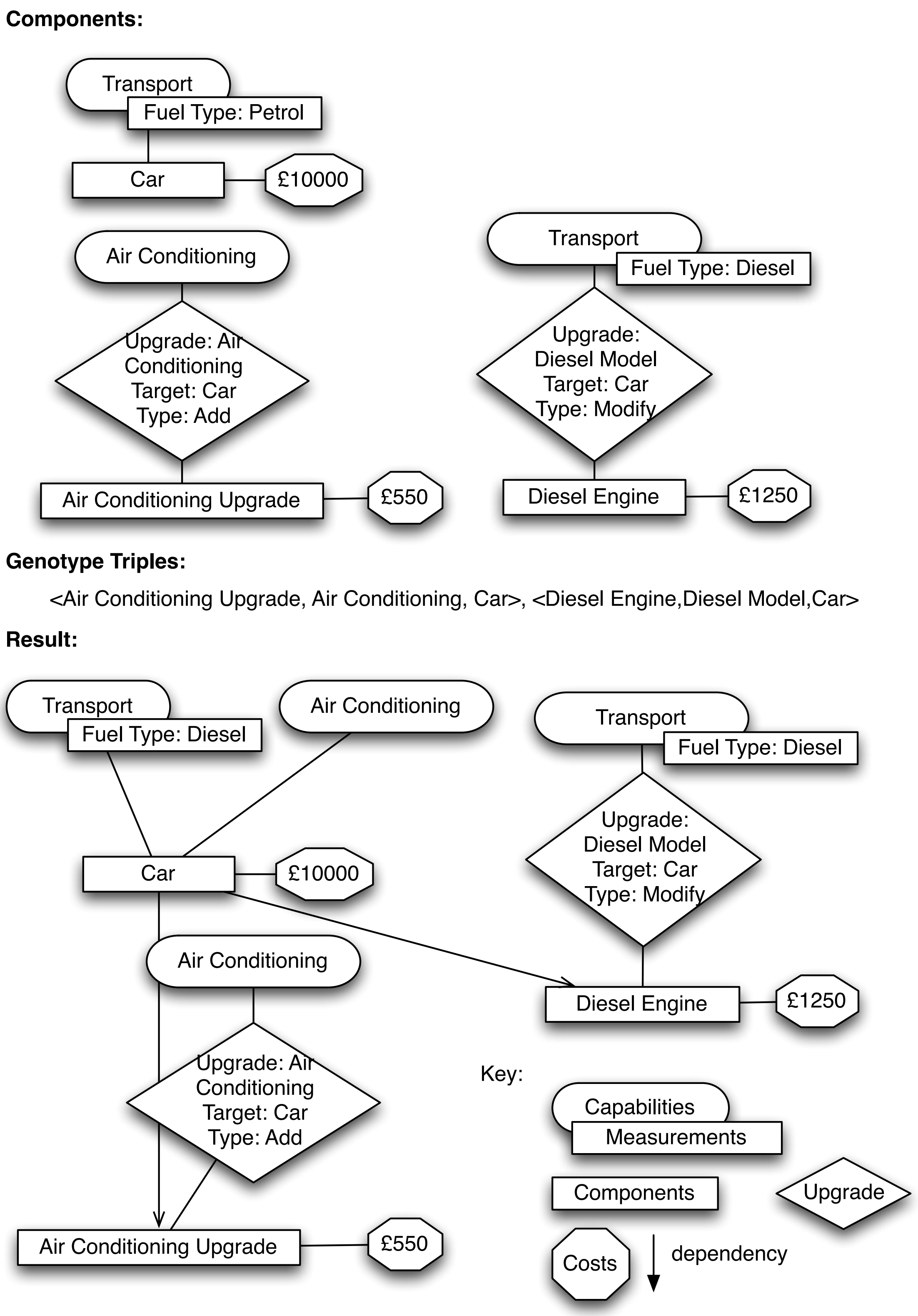


Figure 8.3: The CATMOS Technique - Applying Upgrades

```
Double Point Crossover Example:

Parent A chromosome: <tripleA, tripleB, tripleC>
Virtual positions: 1, 2, 3

ParentB chromosome: <tripleB, tripleD, tripleE>
Virtual positions: 2, 4, 5

Choose two random positions on the chromosome.
Crossover just before 2 and just after 3.

Left side of child from parent A, selecting entries <2:
<tripleA>

Middle of child from parent B, selecting entries >=2 & <=3:
<tripleB>

Right of child from parent A, selecting entries >3:
<>

Resulting child chromosome:
<tripleA, tripleB>
```

Figure 8.4: Double Point Crossover - Example

in figure 8.4.

After crossover is applied, a repair step is then applied to the genotype. Since during crossover, parts of different solutions are combined, the different parts may not connect to each other even when it is possible for them to connect. To solve this and produce better solutions, for every unsatisfied capability in the solution, if there is a corresponding capability provision not currently connected to anything, then it is connected to the unsatisfied capability. In the case of multiple different options for making connections i.e. multiple capabilities and multiple capability provisions with the same name, the connections are made randomly. The repair step in addition to providing better solutions acts as an additive mutation.

The mutation step itself is purely destructive. Random triples are deleted from the child vector at low probability. The advantage of deleting triples is that the costs of the solution are reduced allowing smaller cheaper solutions to be found, possibly with reduced capability satisfaction. The repair step already effectively acts as the additive mutation, so no further additive mutation is required.

The phenotype of the goal model is created by taking the loaded model fragments and connecting *CapabilityProvisions* from one component to another *Component's Capabilities* as specified in the connection triples. The upgrade triples are slightly more complicated in that the upgraded *Component* is temporary modified to include the new *CapabilityProvisions* and a temporary capability dependency is created between the upgrading *Component* and the upgraded *Component* to say that the upgraded *Component* cannot be satisfied until the upgrading *Component* is satisfied.

An evaluation function is needed to allow the meta-heuristic search algorithm to assess how good each solution is. The evaluation function is demonstrated in the previous section (7.2) and the exact algorithm used is given in section 10.4.3. It is applied to the corresponding phenotype, which is constructed as described previously, for the genotype. After all the solutions in a population are evaluated NSGA-II performs non-domination ranking on the values and these non-domination rankings are used in the selection step [119].

An initial population of solutions is also needed to initialise the genetic algorithm. A naive method to generate such a population is to create a vector populated purely by random valid triples. This would be ineffective since most of the solutions generated in this manner will evaluate to having no capability satisfaction at all, due to missing dependencies.

A better method, and the one used in CATMOS, is to use a custom algorithm to create the initial population set. The use of the custom algorithm allows the initial population to be created only with completed solutions where all the dependencies are fulfilled. This means that instead of the genetic algorithm beginning with solutions that will have little to no satisfaction of their objectives due to missing dependencies, the genetic algorithm will start with fully working solutions and will be able to make incrementally better solutions as the search progresses. When the initial population is created with invalid solutions, the genetic algorithm can stall in its search progression as even though it breeds solutions to create new solutions, the new solution and the old solutions both have no satisfaction of the objectives, making it impossible to tell which of the solutions is better meaning the population does not converge to better solutions until it manages to find working solutions by random chance.

The custom algorithm begins by adding all of the leaf capabilities from the *Goal Tree* to a list of unsatisfied capabilities. Each component is treated as having the ability to satisfy a set of capabilities (its capability provisions) and having a set of dependencies that are needed to be able to use it.

Until the unsatisfied capability list is empty:

- Choose a random capability from the unsatisfied capability list.
- Choose a random component with the ability to satisfy that capability.
- Record the choice made as a genotype triple <sourceComponent, capability, (targetComponent or 'scenario' keyword) >.
- Remove the capability from the unsatisfied capability list.
- Add all of the components dependencies to the unsatisfied capability list.

When the unsatisfied capability list is empty, the genotype for a completed goal model has been derived.

There is the minor issue that it is possible for the algorithm to become stuck with an unsatisfied capability that cannot be satisfied due to earlier choices. This is avoided by having the algorithm backtrack; when it reaches an unsatisfiable capability, it reverses its earlier choices and then proceeds to make new random choices preventing it from becoming stuck.

Since the custom algorithm makes random choices in creating the genotype for the completed goal model, a population of random genotypes can be formed by simply repeatedly running the algorithm. If a problem is completely unsolvable the backtracking parser will fail to find a solution.

By using the customisations shown previously, the NSGA-II algorithm [119] can search for a Pareto front of solutions to present to the acquisition decision makers. The solutions provided by the NSGA-II algorithm [119] are approximately rather than exactly on the Pareto front, since it would be computationally impractical to find the exact solutions.

8.3 Multi-objective Next Release Problem - Case Study

To demonstrate CATMOS capabilities with regards to performing multi-objective trade-offs, we are going to use the Multi-objective Next Release Problem (MONRP) as a case study. The MONRP has become well established in the search based software engineering community as a standard problem [28, 130, 137] and hence it makes sense to apply our technique and prototype tool to it to establish a baseline against related work. MONRP is a suitable case study as it focuses on performing a single acquisition whilst considering stakeholder trade-offs. The TLCM problem is more complicated than the MONRP problem but this extra complexity will be covered by the next case study in section 9.5. A preliminary version of the MONRP case study has been published in [3]. Substantial improvements to the case study have been made since the original publication and these are presented in this chapter.

The Next Release Problem (NRP) was originally defined by Bagnall et al [138] and has since been revised by multiple authors. The problem centres on a software company that is planning the next release of their software product [138]. They have multiple customers, each of who have certain requirements they want to be implemented in the next release of the software. However the software company is constrained by limited resources for implementing software requirements before the next release. In the version of the problem we will be using [130], the objective is to determine which requirements will be implemented for the next software release and to find the best requirements to implement at each cost level to the software company. Each customer is considered to have a weighted importance to the software engineering company and each requirement a weighted importance to the customer. The aim of MONRP is to find the solutions on the Pareto front of customer satisfaction against costs [130, 138].

To get a clear idea of what the MONRP is and how we can solve it, we are first going to look at a small instance of the problem. A high street shop is looking to replace its existing stock management system and has contacted a software developer who has quoted them different prices for implementing different pieces of the stock management software. There are effectively two customers of interest: the first is the Shop Manager who owns the store and has the most say in any system to be implemented. The second is the Shop Clerk who has been hired by the Shop Manager to run the store's day-to-day operations. The software developer assigns weights of importance to the two customers, the Shop Manager and the Shop Clerk and those in turn assign weights of importance on the individual requirements.

The Shop Manager with weighting 0.6 has three requirements:

Requirement	Weighting
Monthly Reports	0.4
Email Notifications	0.2
Automatically Generating Orders	0.4

The Shop Clerk with weighted 0.4 has three requirements:

Requirement	Weighting
Easier Stock Handling	0.5
Better User Interface	0.4
Automatically Generating Orders	0.1

Normally, in the NRP and MONRP, all requirements are considered to be implemented as a singular addition to a software package with a cost to implement. CATMOS can support more fine-grained problem descriptions; it can capture software features being implemented independent of the requirements they fulfil. This allows the consideration of alternative solutions to meeting requirements, which is needed for supporting acquisitions trade-offs or evaluating alternative system architectures against each other. This is done by modelling software features as components that provide capabilities separately from modelling software requirements as capabilities in a goal tree.

Returning to the example, there are multiple software features that the software developer is willing to provide at a price to the high street shop. These are:

- *Reusable stock management system base code.* This is reusable code from a previous job by the software developer and costs £400 to purchase and is necessary for any of the other features.

- *Stock-reordering algorithm.* The stock-reordering algorithm calculates when stock is going to run out based on current demand and change in stock levels. This software feature only depends on the stock management system itself and costs £400. This software feature satisfies the requirement for automatically generating orders by the Shop Manager and the Shop Clerk.

- *Email notifications.* This feature generates an automatic notification to the Shop Manager when stock is running low. This solves the Shop Managers requirement and only costs £300. It depends on the stock-reordering algorithm being implemented first.

- *Monthly reports.* This feature provides detailed reports of stock flow each month to the Shop Manager it costs £250 and depends on just the Stock Management System.

- *Barcode scanning system.* This system replaces the existing manual input system for tracking the stock performed by the Shop Clerk and provides the requirement of better stock handling. It costs £450 due to equipment costs.

- *Custom written GUI.* This software feature is a new custom written GUI for the Shop Clerk and fully meets the requirement for a better user interface by the Shop Clerk. This feature costs £250 and depends solely on the reused stock management code.

- *Pre-written GUI.* This software feature is a pre-written GUI used in a previous job by the software developer. It partially meets the requirement for a better user interface but will only cost £50.

The pre-written GUI software feature demonstrates two benefits of using CATMOS not present in other techniques used on the Next Release Problem. The first benefit is that two independent software features can explicitly satisfy the same requirement. For example, two different software features can satisfy the better user interface requirement. The second benefit is being able to capture partial satisfaction of requirements. The first benefit corresponds to trade-offs in the software architecture [3] whereas the second benefit is the support of continuous variable requirements, which was stated as an open research question by Zhang et al [28].

8.3.1 MONRP To CATMOS Overview

When converting problems from the MONRP to CATMOS, the customers and their individual requirements are changed into capabilities with the customers decomposing into their individual requirements. A top capability can be added that decomposes into all the customers to measure the weighted sum of their satisfaction. CATMOS also has a requirement / solution split, in not only do we need to say we want the requirement we also need to say what provides the requirement. In the MONRP case, this means that a component needs to be created for each requirement that provides the satisfaction of that requirement as a capability provision. The requirement's costs and dependencies are added to that component.

8.3.2 CATMOS DSL Explanation - MONRP Case Study Problem

This problem has been expressed using the CATMOS DSL shown in section 8.4. Firstly, there is an 'overallSatisfaction' capability that is used to represent the total satisfaction of all the customers subject to the software developer's weights. The 'overallSatisfaction' capability is decomposed into each of the customers represented as capabilities. In this case the 'Shop Manager' and the 'Shop Clerk'. Each customer then decomposes into their individual requirements (e.g. Monthly reports, Email notifications, etc.) also represented as capabilities. This forms the overall *Goal Tree* for the problem.

The 'overallSatisfaction' capability is annotated by a measurement with a critical value of 0.0 and a benchmark value of 1.0. An in-line script is used to aggregate the satisfaction levels of all the customers and apply the software developer's weights. The customers are likewise annotated by a single measurement each with critical value of 0.0, benchmark value of 1.0 and an in-line script that aggregates the satisfaction of all their requirements subject to the customer's weights. Most of the requirements can either be fully fulfilled or fully unsatisfied and in which case they need not be annotated with a measurement. Requirements that can be partially fulfilled need to be annotated like the 'Better Userinterface' with a measurement with a critical value of 0.0 and a benchmark value of 1.0. This allows the use of continuous variable requirements in satisfying the requirements.

The software features are captured as components. Each component lists the capabilities it provides (which in this case are the requirements it satisfies) as capability provisions. Dependencies are handled by listing required capabilities on the component with the providing component listing the relevant capability provision. An example of this is ‘Stock Management System’ that provides the capability ‘Stock Management System’ that is then required by other components. The cost to acquire each component is listed as well on the component.

The top part of the DSL code is the technique’s settings. This includes the population size and generation count that affect the accuracy of the search results. The components that can be acquired during the search and how many of each is available. In the MONRP this is always one of each component. The last setting is that the cost ‘Money’ is desired to be as low as possible. This forms one of the objectives in the search. The other objective is the maximisation of the overallSatisfaction capability set by using the keyword ‘searchObjective’ in front of it.

8.3.3 CATMOS Runtime

As an example of the algorithm’s runtime using population size 200 and generation count 200 for the algorithm an average run (averaged over 5 runs) takes 93.4 seconds (range 71 - 127 seconds) running on a single core on a MacBook Air Mid 2011 (Intel Core i5 1.7Ghz, 4GB 1333 MHz DDR3). A generated Pareto front from the runs is shown in figure 8.5. It should be noted that because of the stochastic nature of the algorithm, the Pareto front slightly varies between runs. This means that slightly different trade-off points are presented to the decision maker between runs; however this is not normally a problem because the differences are small. It should also be noted that the produced Pareto front is an approximation. This can be seen by looking at the left most point with no produced customer satisfaction but some cost. The run included one of the components in the result where the optimal result for cost equals 0 is to purchase nothing at all. All of the points are related to corresponding goal models allowing the decision makers to examine the solutions for themselves.

8.3.4 Example Solution

An example solution is shown in figure 8.6. The production of the corresponding goal models allows the decision maker to check the produced acquisition solutions and see not only included components but also how they work together to achieve the desired result. The graphical notation currently produced by the prototype tool is rather coarse. This would be a target for improvement in commercialisation of the work. Capabilities are represented with boxes containing a name and a satisfaction level of ‘Green’, ‘Yellow’ & ‘Red’. Green means fully satisfied, yellow means partially satisfied above the critical measurement levels and red means not satisfied. Components are also represented with boxes but contain a name with a number instead of a satisfaction level. The number identifies which component is being referred to in the case of multiple components with the same name been acquired. Capabilities only contain their measurements. Component boxes are split into three sections and contain capability provisions in their top box, capability dependencies in their second box and in their third box contain costs. Capability decompositions and capabilities being satisfied are shown using simple arrows.

Another advantage of our approach over existing techniques for the NRP is that our tool is more generic and supports the generation of datasets for more than 2 objectives on the Pareto front. A 3 dimensional Pareto front for the problem is shown in figure 8.7. The 3D Pareto front

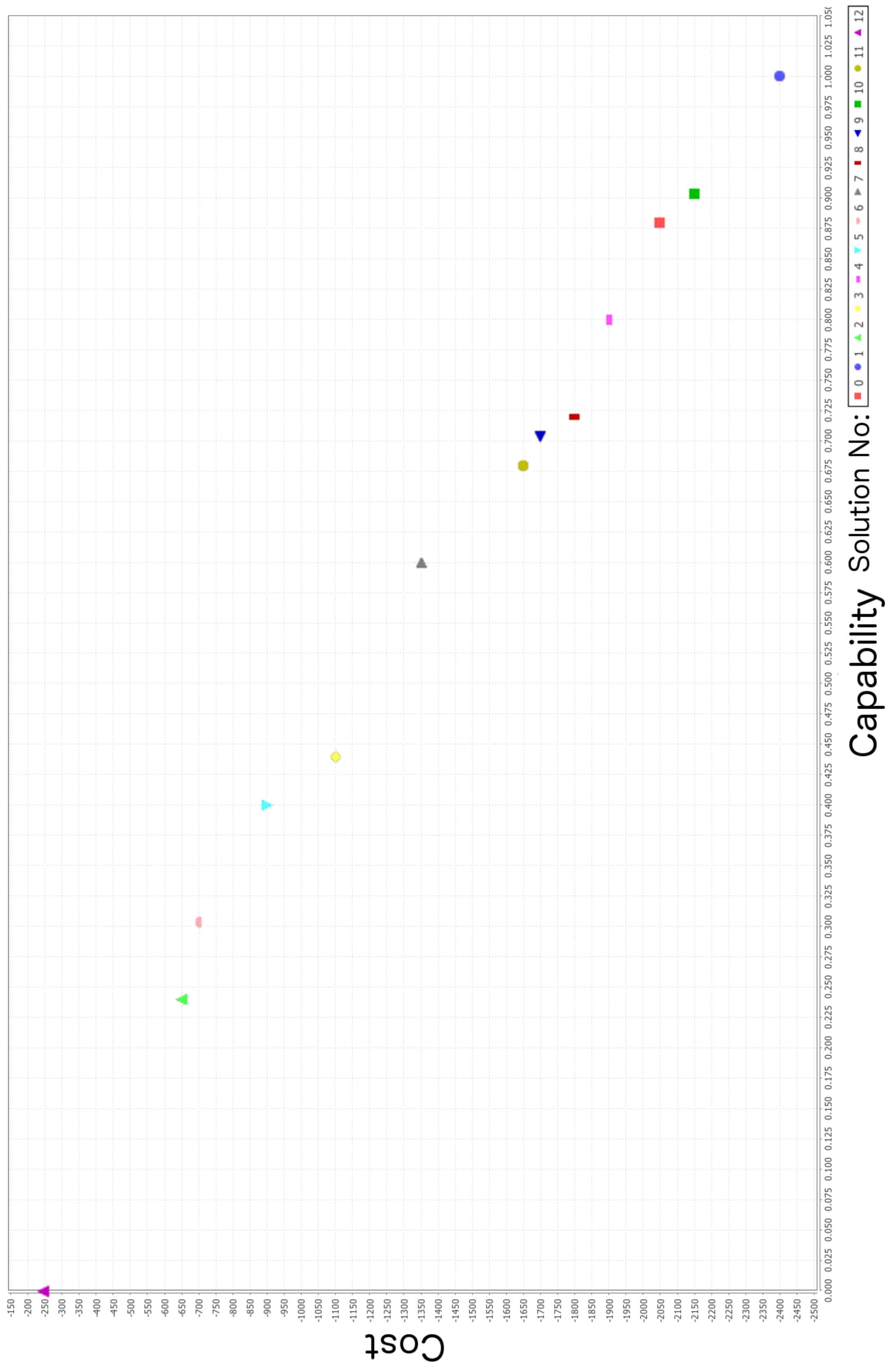
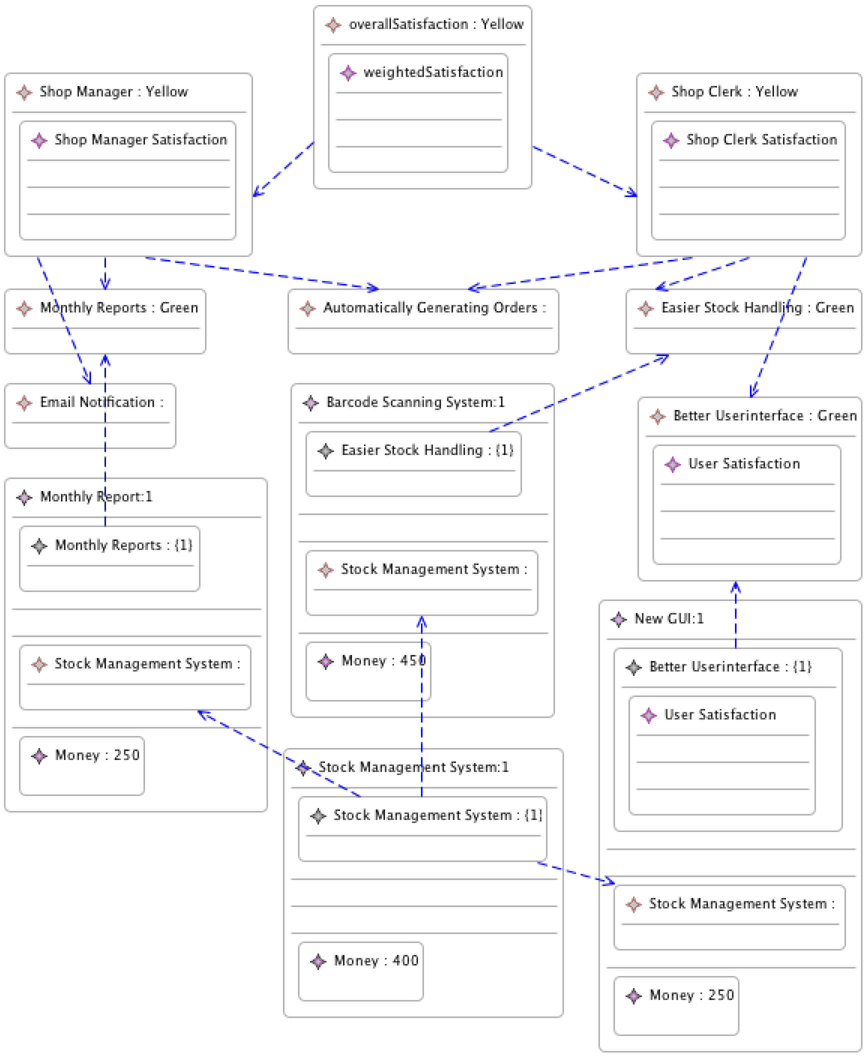


Figure 8.5: Shop Keeper Example - Pareto Front of Satisfaction vs. Cost



overallSatisfaction : Yellow
weightedSatisfaction
Shop Manager : Yellow
Shop Manager Satisfaction
Shop Clerk : Yellow
Shop Clerk Satisfaction
Monthly Reports : Green
Automatically Generating Orders :
Easier Stock Handling : Green
Email Notification :
Barcode Scanning System:1
Easier Stock Handling : {1}
Better Userinterface : Green
User Satisfaction
Monthly Report:1
Monthly Reports : {1}
Stock Management System :
Money : 450
New GUI:1
Better Userinterface : {1}
User Satisfaction
Stock Management System :
Money : 250
Stock Management System:1
Stock Management System : {1}
Stock Management System :
Money : 400
Money : 250

Figure 8.6: Example Goal Model Solution

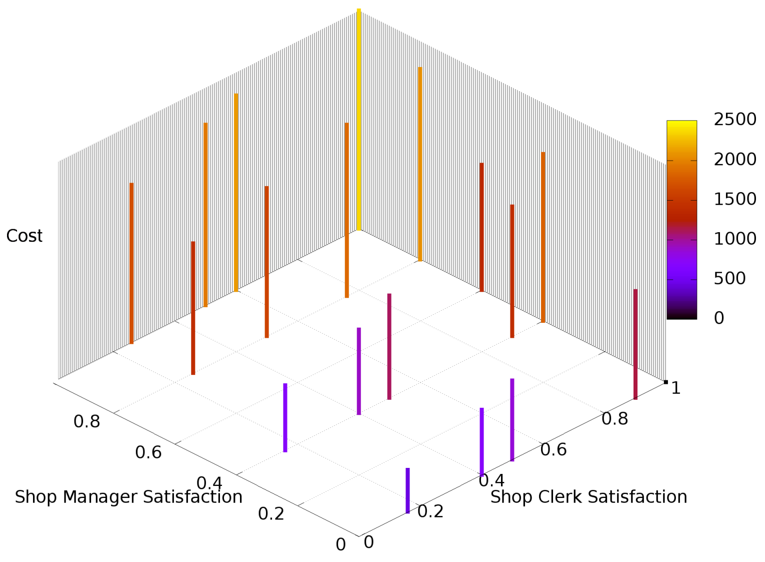


Figure 8.7: Shop Keeper Example - 3D Pareto Front

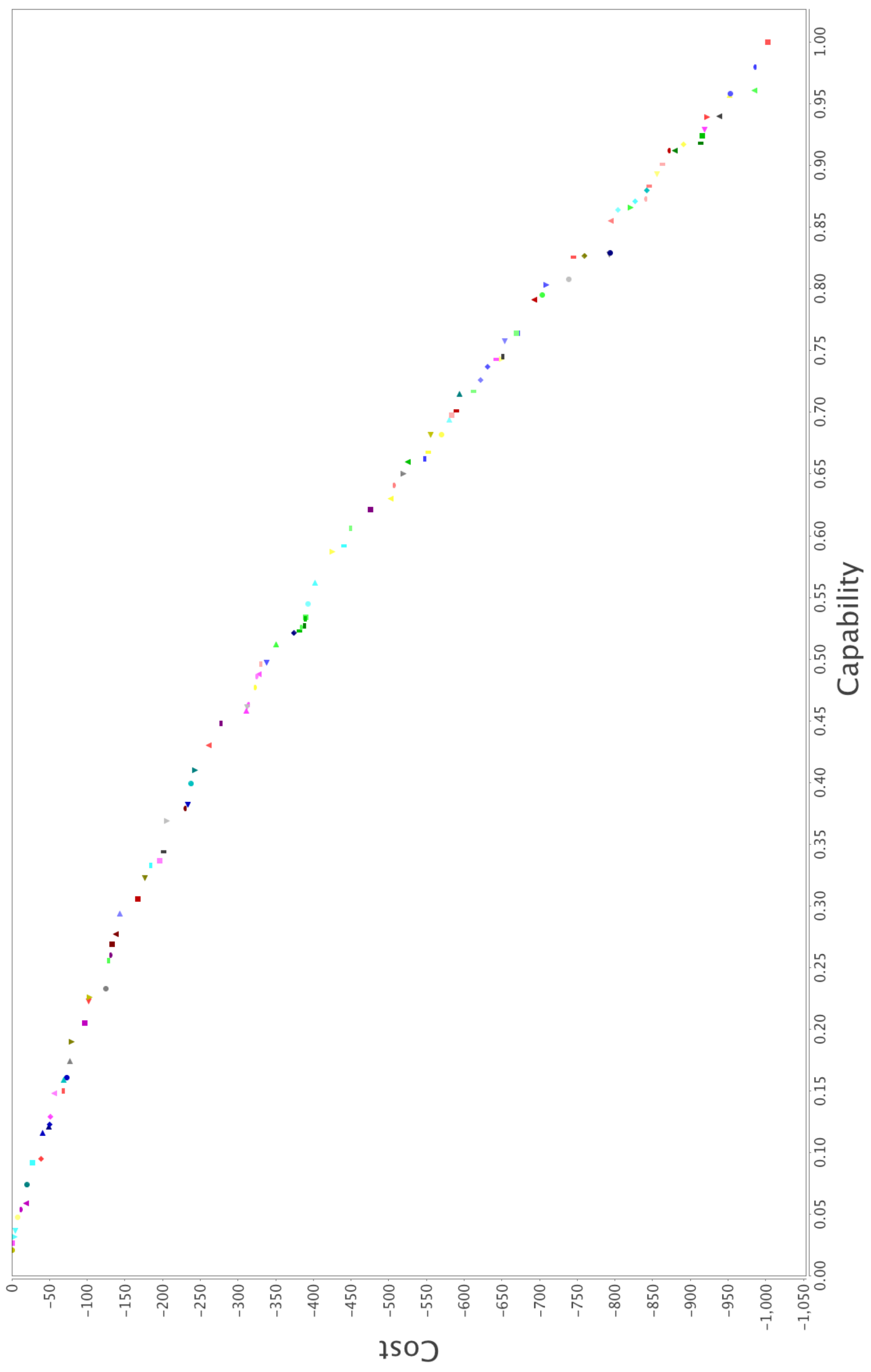


Figure 8.8: Pareto Front For MONRP 100 Customers 200 Requirements

has the Shop Manager satisfaction on one axis, the Shop Clerk satisfaction on the other axis and the line height on the plot between the two axes is the cost. An example of the tool being run on a larger problem with 100 customers and 200 requirements is shown in figure 8.8. The algorithm complexity is typically O(n^2) (see section 11.2).

8.4 Shop Keeper Textual DSL Input

The following is a textual specification of the shop keeper case study problem. The grammar and syntax are as described previously in section 7.4.

```
FindTradeOffs ShopCaseStudyNRP { popSize 200 genCount 200
AcquirableComponent "Email Notifications" 1
AcquirableComponent "Stock Reordering Algorithm" 1
AcquirableComponent "Invoice Generator" 1
AcquirableComponent "Stock Management System" 1
AcquirableComponent "Monthly Report" 1
AcquirableComponent "Barcode Scanning System" 1
AcquirableComponent "New GUI" 1
AcquirableComponent "Pre written GUI" 1
DesireLow "Money" }
```

The top part defines the problem's name, the population count for the multi-objective search algorithm, the generation count for the multi-objective search algorithm, each of the acquirable components to include in the search and that there is one of each of them and that the cost 'Money' should be minimised.

```
Capability overallSatisfaction {
        Measurement weightedSatisfaction {
                criticalValue 0.0 benchmarkValue 1.0
                script "output = 0.6 * getCapability(\"Shop Manager\") +
                    0.4 * getCapability(\"Shop Clerk\")" }
        decomposes ("Shop Manager", "Shop Clerk") }
```

The overall satisfaction capability represents the overall weighted customer satisfaction. The small in-line script performs weighted sum on the two customer satisfactions. The overall satisfaction capability is considered to decompose into the customers (Shop Manager & Shop Clerk).

```
searchObjective
Capability "Shop Manager" {
        Measurement "Shop Manager Satisfaction" {
                criticalValue 0.0 benchmarkValue 1.0
                script "output = 0.4 * getCapability(\"Monthly Reports\") +
                     0.2 * getCapability(\"Email Notification\") + 0.4 *
                    getCapability(\"Automatically Generating Orders\")"}
        decomposes ("Monthly Reports", "Email Notification", "Automatically
              Generating Orders") }

searchObjective
Capability "Shop Clerk" {
        Measurement "Shop Clerk Satisfaction" {
                criticalValue 0.0 benchmarkValue 1.0
                script "output = 0.5 * getCapability(\"Easier Stock
                    Handling\") + 0.4 * getCapability(\"Better
                    Userinterface\") + 0.1 * getCapability(\"Automatically
                    Generating Orders\")" }
        decomposes ("Easier Stock Handling", "Better Userinterface", "
            Automatically Generating Orders") }
```

The two customers are defined to have a satisfaction that depends on the weighted sum of the satisfaction of the requirements they desire. This is again done using a small in-line script. They are also considered to be the search objectives for the multi-objective search via the ‘searchObjective’ keyword. The customers are both considered to decompose into the requirements they desire.

```
Capability "Monthly Reports" {}    Capability "Email Notification" {}
Capability "Automatically Generating Orders" {}    Capability "Easier Stock
    Handling" {}

Capability "Better Userinterface" { Measurement "User Satisfaction" {
    criticalValue 0.0 benchmarkValue 1.0 } }
```

All of the requirements are then defined as capabilities to indicate they are wanted things during the acquisition. The ‘Better Userinterface’ capability has a measurement to allow it to be partially satisfied between 0.0 and 1.0.

```
Component "Email Notifications" {
        CapabilityProvision "Email Notification" {} Capability "Stock
            Reordering Algorithm" {} Cost Money 300.0 }

Component "Stock Reordering Algorithm" {
        CapabilityProvision "Stock Reordering Algorithm" {} Capability "
            Stock Management System" {} Cost Money 400.0 }

Component "Invoice Generator" {
        CapabilityProvision "Automatically Generating Orders" {} Capability
            "Stock Reordering Algorithm" {} Cost Money 300.0 }

Component "Stock Management System" { CapabilityProvision "Stock Management
    System" {} Cost Money 400.0 }

Component "Monthly Report" {
        CapabilityProvision "Monthly Reports" {} Capability "Stock
            Management System" {} Cost Money 250.0 }

Component "Barcode Scanning System" {
        CapabilityProvision "Easier Stock Handling" {} Capability "Stock
            Management System" {} Cost Money 450.0 }

Component "New GUI" { CapabilityProvision "Better Userinterface" {
    Measurement "User Satisfaction" { providedValue 1.0 } }
```

```
        Capability "Stock Management System" {} Cost Money 250.0 }

Component "Pre written GUI" { CapabilityProvision "Better Userinterface" {
    Measurement "User Satisfaction" { providedValue 0.4 } }
        Capability "Stock Management System" {} Cost Money 50.0 }
```

The software features are defined as components that provide certain software requirements, have a cost and optionally require another software requirement to be implemented first. The 'Pre written GUI' component uses a measurement on its capability provision to provide a partial satisfaction of 0.4 to the requirement.

8.5 Contributions to the Multi-objective Next Release Problem

In published work [3] by the author, the CATMOS technique was shown to offer a number of specific novelties when applied to the Multi-objective Next Release Problem (MONRP). These novelties are due to the CATMOS technique being designed to handle the more complicated acquisition problem of Through Life Capability Management (TLCM).

- *Continuous variable requirements.* A stated research challenge for work on the Next Release Problem was the handling of continuous variable requirements [28]. These are requirements that can be satisfied over a real number. For example, a web server may have a minimal requirement to serve a webpage within 300ms and a desired requirement to serve a webpage within 100ms. Our approach allows the requirement to be treated as not satisfied, satisfied or partially satisfied to a degree between 0 & 1 on the real numbers. In previous work [130, 138–140], a requirement may only be satisfied or not satisfied. [3]

- *Visualisation of solutions.* An issue in the NRP is to not only find the 'best' solution but to also explain to the stakeholders why the solution is good [28]. Our approach partially supports this by using a graphical domain specific language to visualise solutions so that stakeholders can understand them. [3]

- *Continuous Release Support.* Our technique includes support for releasing over continuous time periods, not just a single release or a couple of release dates like previous existing techniques for solving the Multi-objective Next Release Problem. This feature is shown in the case study in chapter 9.

- *Technique Flexibility.* The technique being designed for a more encompassing problem than dedicated Next Release Problem techniques is able to support things such as generating Pareto fronts with more than 2 dimensions. An example of this is there being two customers and generating a 3 dimensional Pareto front with the two customer's satisfaction on axis and a cost on the third axis. Another example is software features that can alter the properties of other software features. [3]

8.6 Summary

In this chapter, we have covered the application of multi-objective search to the CATMOS technique to automatically produce a Pareto front of acquisition plans. So far we have covered what is required to perform capability management rather than Through Life Capability Management (TLCM). In the next chapter, we cover the through life extensions to the work presented in this chapter to allow the technique to address the TLCM problem as a whole. This is followed by performing the CATMOS technique on a realistic military case study.

Chapter 9

Scheduling Acquisitions Through Life

9.1 Introduction

In previous two chapters, we have introduced a technique for managing acquisitions via Capability Management. In this chapter, we extend that technique to deal with the through life aspects of Through Life Capability Management (TLCM) to address research gap 3 of the thesis. We begin by briefly discussing the through life aspect of TLCM and what the problems are in supporting it before moving on to provide details of our approach to it. Next, we provide new grammar and syntax extensions for supporting the through life aspect along with a new metamodel. Finally, we will perform a case study using CATMOS on a realistic military acquisition scenario. The application of the CATMOS technique to the realistic military case study has been published in [1].

9.2 Through Life Extension to CATMOS

In any System of Systems, not all the systems will be acquired at the same time; instead they are gradually acquired over time. Systems will also eventually retire. This can lead to capability gaps where systems go out of service before replacements are introduced [16]. Another major issue is that of budgetary constraints that prevent large numbers of system being acquired simultaneously. This means that acquisitions need to be scheduled over large periods of time. Since the acquisition of systems can depend on the acquisition of other systems, this leads to imposing a temporal ordering on the acquisition. For addressing this problem, we both need to be able to schedule the acquisitions over time with temporal constraints and need a way to identify capability gaps before they occur.

In the CATMOS technique to handle through life concerns, the user needs to provide the following information:

- When each capability is wanted, its start and end date.
- When existing components come into and leave service.
- How long it takes for an acquired component to come into service and if applicable when does it then leave service.
- The costs for an acquired component and when they need to be paid.

- Whether the acquired component needs another acquired component to come into service first before it can be acquired.

- The overall budget information of when resources are available for acquisition.

With this information the CATMOS technique can schedule the acquisition plans it makes and reevaluates their fitness in accordance with the temporal constraints.

The information for when capabilities are wanted is provided by the end user annotating the capabilities with a start and end date. Existing components are annotated with the date they came into service and if applicable when they will leave service. Acquirable components need to be annotated with how long it takes to acquire them and their lifespan if applicable. Additionally, any costs associated with acquirable components need to be annotated with when they will occur relative to the acquisition and whether there are any repeating costs such as wages or maintenance costs that need to be considered.

Previously, CATMOS was used to generate a Pareto front of results of capability against costs. When performing through life acquisitions with CATMOS, it can still do that but more likely the decision maker wants CATMOS to find solutions that fit within their budget rather than what they would gain or lose by adding more or less budget. For this, since the acquisitions take place over time, CATMOS needs to know both the resources available for performing the acquisitions and when they are available so CATMOS can schedule the acquisitions of components around these constraints.

9.3 CATMOS technique modifications

For the through life extension, the CATMOS technique needs to be modified in several ways. The first modification is to components and their dependencies. Components are unable to use their dependencies to satisfy them until the dependencies have come in to service. Additionally, capability dependencies of components can be marked as ‘sequential’ for supporting cases when the acquisition of a component cannot begin until another component is in service. This can occur when the acquisition of a component depends on design decisions made in a previous acquisition. The scheduler takes this information and ensures that the component will not be acquired until the component satisfying the capability dependency is already in-service.

The technique also needs to be given a designated time period during which it will consider scheduling its acquisitions between otherwise the technique could suggest starting to acquire a component yesterday.

When using the through life extension to the CATMOS technique, components cannot be acquired until there are sufficient resources available. Scheduling is carried out as a separate step in the genetic algorithm after the phenotype has been formed but before it is evaluated. The scheduling algorithm is used in a meta-heuristic search algorithm meaning that in a typical run it will be called at least 10,000 times and therefore the scheduling algorithm needs to be fast. The number of calls is based on the typical run sizes used for the simpler MONRP problem [130]. This is achieved by making the scheduler make only one pass of the solution in chronological order. The scheduler takes the locally greedy options at each point meaning while it is likely to produce good schedules the produced schedules are not guaranteed to be optimal. This is a performance vs. accuracy trade-off. The vast majority of scheduling problems should be simple enough that taking the locally greedy options will result in the optimal schedule. In the other cases, the genetic algorithm has the ability to alter the connections between the parts of the goal model that can significantly affect the way the solution is scheduled. The genetic algorithm will select for solutions that can be better scheduled due to selective pressures.

The scheduler starts by looking at the earliest needed capability and looking for components that satisfy it. From these components it selects the component that can satisfy the capability first. In the case of multiple components it looks for the component that can satisfy it the longest. Something to note is that the same capability can be satisfied by multiple components in through life mode, allowing multiple different components to satisfy a capability throughout its lifespan. The component that satisfies it for the longest is then scheduled. If the capability is not satisfied for its entire lifespan, then the next component that can satisfy it for the next longest period of time after the first component is selected and so on. Once the capability is satisfied or there are no remaining components to schedule for it, the process moves on to the next earliest capability to be scheduled and so on.

There is additional complexity in calculating when components can satisfy a capability. It is not sufficient to simply check if there is sufficient budget to cover the component's cost. The costs of all of the component's dependencies and when they can be scheduled needs to be considered first. This is done by scheduling each of the component's dependencies in order to a temporary plan. When each of the component's dependencies is added to this temporary plan, the resources available to add the next dependency to the temporary plan are reduced. This leads to component's being scheduled later on when resources become available to acquire them. If the component is selected to be scheduled then the temporary plan is added to the scheduling plan otherwise it is deleted.

The scheduling of the solutions is performed just before the evaluation step in the genetic algorithm. The scheduling is done here rather than trying to include scheduling information into the genotype because using a genetic algorithm is typically a lot more computationally expensive than simply using a dedicated algorithm for the same problem. A flowchart for the CATMOS technique with the scheduling and through life parts of the technique added is given in figure 9.1.

For the through life extension to CATMOS, the evaluation step has been modified to take account of the scheduling information. The evaluation step evaluates how well a solution meets the desired objectives. The evaluation needs to be modified to take account of how the available capability from a solution changes over time. Re-running the evaluation for each time a capability starts or stops being required or a component enters or leaves service does this. When the evaluation is run at a time it only considers components that are in-service to be part of the solution and only evaluates with capabilities wanted at that time. Each of the capabilities' satisfaction scores are evaluated using their average satisfaction level during the time period they are desired.

As an example, if a capability is wanted for 10 days and it is satisfied for 5 days completely and the other 5 days only half then the capability is evaluated as follows: $(0.5 * 1.0) + (0.5 * 0.5) = 75\%$ satisfaction level.

When the CATMOS prototype tool was applied to a problem previously in chapters 7 or 8, it

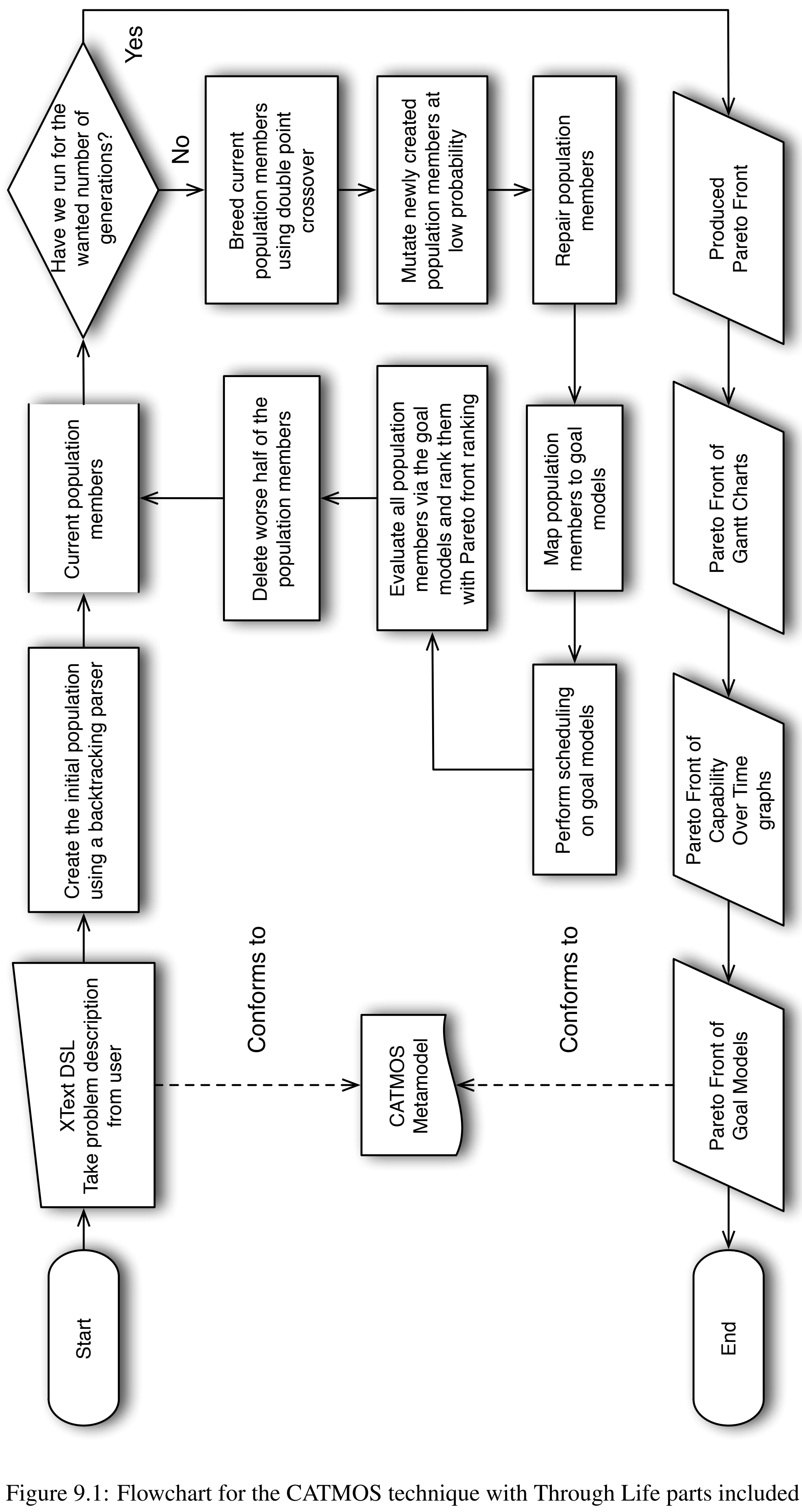

Start
XText DSL
Take problem description from user
Create the initial population using a backtracking parser
Current population members
Have we run for the wanted number of generations?
Yes
No
Breed current population members using double point crossover
Mutate newly created population members at low probability
Repair population members
Map population members to goal models
Perform scheduling on goal models
Evaluate all population members via the goal models and rank them with Pareto front ranking
Delete worse half of the population members
Conforms to
CATMOS Metamodel
Conforms to
Produced Pareto Front
Pareto Front of Gantt Charts
Pareto Front of Capability Over Time graphs
Pareto Front of Goal Models
End

Figure 9.1: Flowchart for the CATMOS technique with Through Life parts included

produced a Pareto front with corresponding goal models for each solution. When using the through life extension it produces these charts and additionally produces for each solution a capability-over-time chart of how the satisfaction of the capabilities vary over time and a Gantt chart of when the acquisitions of the individual systems should be scheduled.

An example of these additional charts for a modified Multi-objective Next Release Problem (MONRP) is shown in figure 9.2 and figure 9.3. The modifications made to the MONRP are to introduce an acquisition time for each component, marking some of the dependencies between components as sequential i.e. you cannot implement a requirement until another requirement has been implemented and the inclusion of a scheduled budget rather than a single lump sum of resources.

9.4 Through Life Extensions to the CATMOS Domain Specific Language

To support the through life extension to CATMOS, the technique needs to take in extra information about the acquisition scenario including information about when capabilities are desired, the timings of the budget for the acquisition project and the scheduling timings for the various acquirable and existing components.

To include this information in the proof of concept tool support requires the definition of a single *ThroughLifePlanning* block rather than a single *FindTradeOffs* block. The full underpinning metamodel for the CATMOS technique is shown in figures 9.7 and 9.8 and this metamodel is both used for generating the textual DSL interface, which will be described below and graphical interface. In this section, we will discuss the through life extensions to the grammar and syntax. The basic grammar and syntax is discussed in sections 7.4 and 7.9.

```
ThroughLifePlanning <scenarioName> {
    popSize <searchPopulationCount> genCount <searchGenerationCount>
    startDate <dateString> endDate <dateString>
    (ExistingComponent <componentName> <quantity> (startDate <dateString>
        endDate <dateString>)? )*
    (AcquirableComponent <componentName> <quantity> acquisitionTime <
        durationString> lifeSpan <timeString>)*
    (DesireLow <costType>)*
    (DesireHigh <costType>)*
    (Budget <costType> {
        amount <realNumber>
        startDate <dateString>
        (repeatDuration <durationString>
        (endDate <dateString>)?)?
    })*
}
```

Figure 9.4: ThroughLifePlanning Block Grammar

A *ThroughLifePlanning* block is similar to the *FindTradeOff* block discussed earlier in section

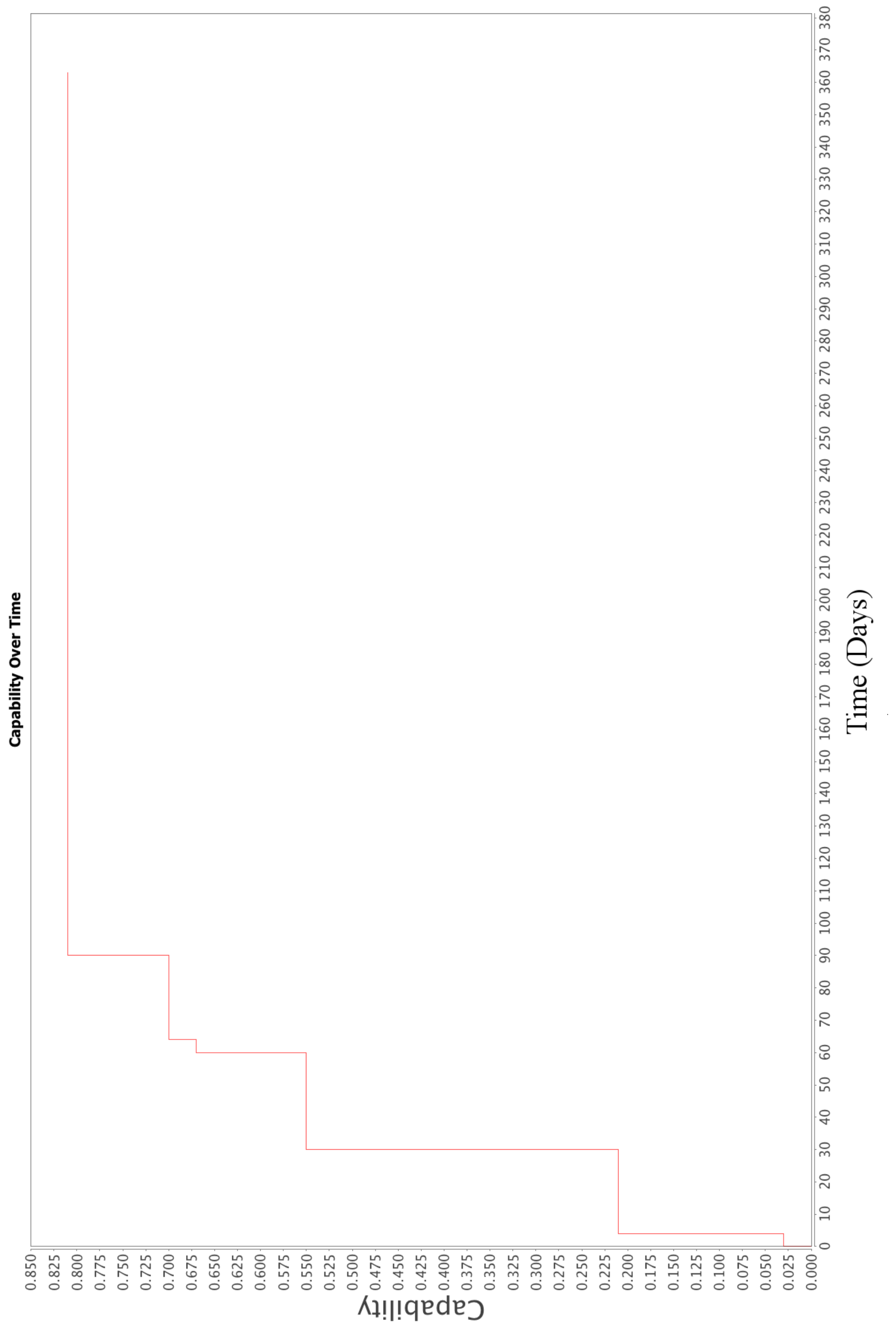


Figure 9.2: Through Life MONRP - Example Generated Capability Over Time Graph

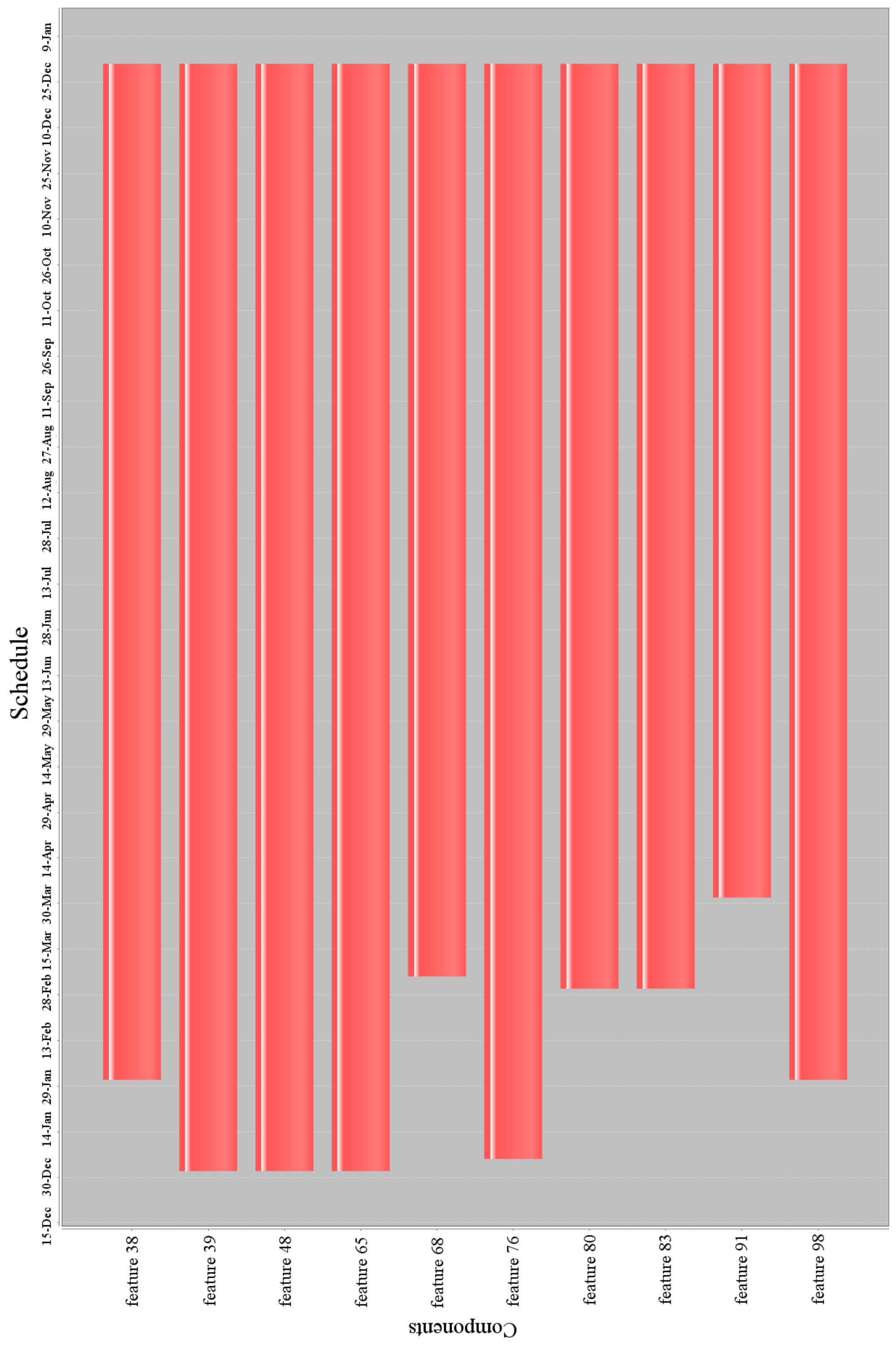


Figure 9.3: Through Life MONRP - Example Generated Gantt Chart

7.4.1. The additional features are the usage of the *startDate* and *endDate* line. This defines the time period for which the problem will be evaluated. No acquisitions can be scheduled before the *startDate*. A <dateString> is a string formatted as “12/05/2014”.

ExistingComponents can optionally have a *startDate* when they came into service and an *endDate* when they will leave service. *AcquirableComponents* must have a *acquisitionTime*, which describes how long it takes from starting to acquire the component until it comes into service and a *lifeSpan* of how long the component stays in service for. A <durationString> can be a number value such as “3” to represent 3 days. Alternatively it can be a value such as “5 months” to represent 5 months. The month strings are internally converted to days assuming 30 days to the month. Lastly, the duration string can be ‘inf’ for an infinite duration.

Budget is a new feature in the through life extension and describes the resources available for an acquisition. The <costType> matches up with the *Costs* defined on *Components*. A *Component* cannot be acquired in through life mode if there is insufficient budget for it. *Budgets* have a real numbered amount, a *startDate* and optionally a *repeatDuration* for if they are reoccurring and optionally a *endDate* for if they stop reoccurring at some time.

```
(standAlone)?
(searchObjective)?
Capability <capabilityName> {
    (accumulation <measurementName>)?
    (startDate <dateString> endDate <dateString)?
        (Measurement <measurementName> {
                (criticalValue <realNumber>
                benchmarkValue <realNumber>)
                |
                (criticalValues (<stringValue>(,<stringValue>)*)
                benchmarkValues (<stringValue>(,<stringValue>)*) )
                (script <script>)?
        }
        )*
        (decomposes (<capabilityName>(,<capabilityName>*)))?
}
```

Figure 9.5: Capability Block Grammar - Through Life

Capabilities on the top level goal model have a very small modification of having a *startDate* and *endDate*, which defines the time period during which the satisfaction of the capability will be considered by the multi-objective search.

```
1  Component <componentName> {
2      (CapabilityProvision <capabilityName> {
3          (reuse <reuseTimes>)?
4          (Measurement <measurementName> {
5              ( providedValue <realNumber> )
6              | ( providedValues (<stringValue>(,<stringValue>)*) )
7              (script <script>)?
8          })*
9      })*
10
11     (CapabilityUpgrade <upgradeName> {
12         targetComponent <targetComponentName>
13         (CapabilityChange <changeType> {
14             CapabilityProvision <changedCapabilityProvisionName> {
15             (reuse <reuseTimes>)?
16             (Measurement <measurementName> {
17             ( providedValue <realNumber> )
18                 | ( providedValues (<stringValue>(,<stringValue>)*) )
19                 (script <script>)?
20             })*
21         })*
22     })*
23
24     (sequentialScheduling)?
25     (Capability <capabilityName> {
26         (accumulation <measurementName>)?
27         (Measurement <measurementName> {
28             ( criticalValue <realNumber>
29             benchmarkValue <realNumber> ) |
30             ( criticalValues (<stringValue>(,<stringValue>)*)
31             benchmarkValues (<stringValue>(,<stringValue>)*) )
32             (script <script>)?
33         } )*
34     } )*
35
36     (Cost <costType> <costAmount>
37     ({
38         ('startAfter' <durationString>
39         ('repeatDuration' <durationString>
40         ('stopAfter' <durationString>)?)?)?
41     })?
42     )*
43 }
```

Figure 9.6: Component Block Grammar - Through Life

Components have a few minor modifications in through life mode. The first is their capability dependencies can be marked as sequential (line 24) as discussed previously. The second is that costs can now have temporal information added. The *startAfter* allows the time of the cost to be delayed. The *repeatDuration* allows the *Cost* to be made repeating to capture costs such as

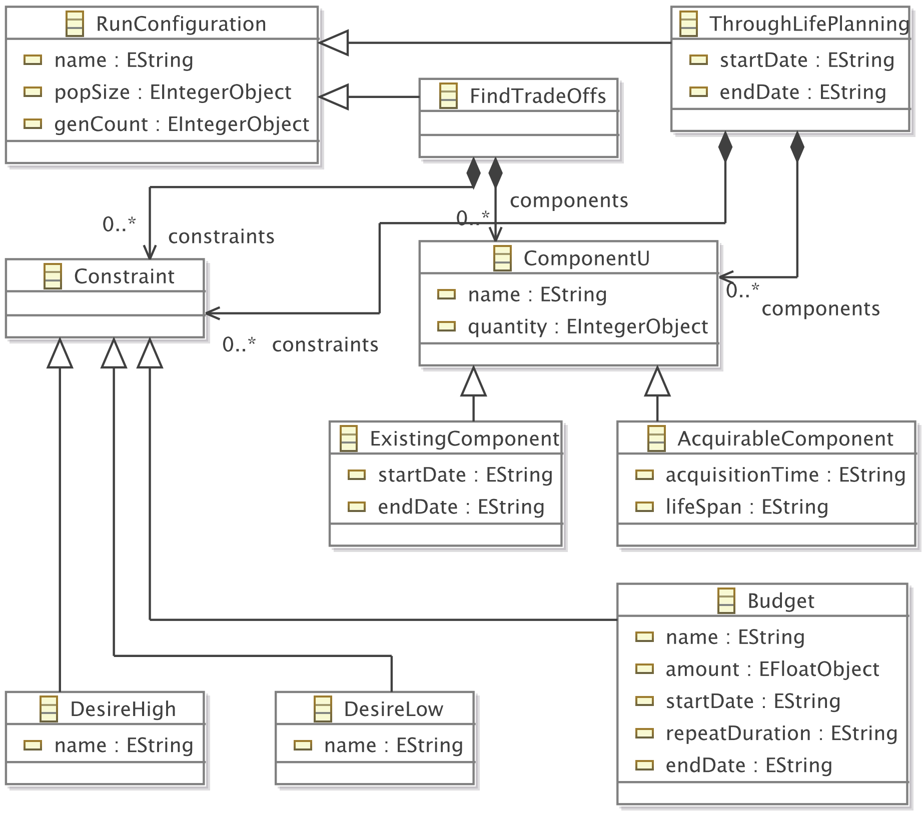


Figure 9.7: CATMOS Full Meta-Model - Acquisition Settings

maintenance costs. The *stopAfter* designates when the repeating should stop (if ever).

This concludes the new notation for the through life extension to CATMOS and the grammars and syntax shown here are the full grammars and syntax for the CATMOS technique.

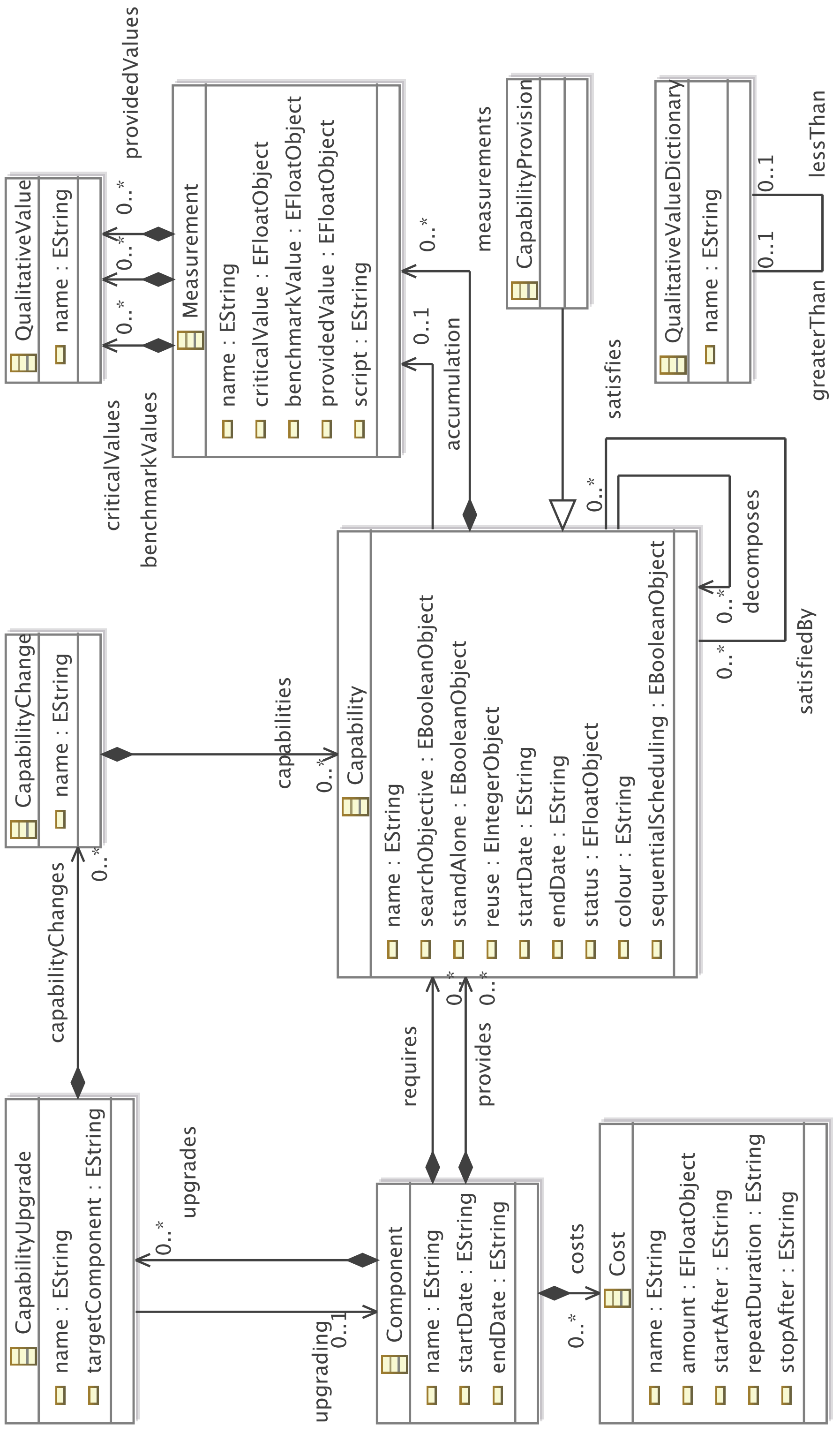


Figure 9.8: CATMOS Full Meta-Model - Capabilities and Components

9.5 Military Acquisition Scenario - Case Study

In this section, we will apply the CATMOS technique with its through life extension to a military acquisition scenario. The work shown here has already been published by the author in [1]. The scenario is based on a scenario provided by MooD International; the scenario is realistic but not actually real. The overall scenario objectives and the scope and complexity of the scenario are realistic and the vast majority of the systems and their interactions are real. The semantic annotations for the aggregations require detailed domain knowledge of the individual systems and are therefore mostly fictitious. The main two purposes of this case study are show that CATMOS is able to deal with the complexity of real acquisition problems and to demonstrate that CATMOS can handle the through life part of the Through Life Capability Management (TLCM) problem.

9.5.1 Scenario

The military acquisition scenario revolves around stopping an enemy transporting supplies across a river. There are three major objectives in the scenario. The first objective is clearing a route to a forward base that will be established near the river. The second objective is establishing and holding the forward base near the river. The third and final objective is preventing the enemy from crossing the river.

Each of these objectives have being further decomposed into sub-objectives. Clearing the route to the forward base is decomposed into having sufficient ground firepower to repel enemy forces, having sufficient hard target removal to clear enemy bunkers and fortifications and having the ability to disable enemy road side mines. Holding the forward base decomposes into establishing the forward base and keeping the forward base supplied with goods, water and fuel. Preventing enemy river crossings decomposed into detecting enemy crossings and stopping enemy river crossings. It's not possible to stop an enemy river crossing that you failed to detect. The full goal-tree decomposition is shown in figure 9.9.

The aim of the acquisition scenario is to perform a trade-off between satisfying the three objectives with the limited budget. There are also several concerns that need to be considered in the acquisition scenario. Currently, due to political pressures, it is not possible to obtain approval for the acquisition of more troops. In light of this, the head of the SAS training division has suggested rolling out SAS training more widely to make better use of the existing troops. Another major concern is that the contract for the maintenance of the L118 Light Guns is about to expire. To keep the existing L118 Light Guns in-service or to acquire more L118 Light Guns will require the contract to be renewed at considerable expense. Additionally, the manufacturer of the Mobile Artillery Battlefield Radar system, which so far has had limited roll out, has filed for bankruptcy meaning that the existing systems are unmaintainable. The systems could be replaced with an alternative system current in-service in a friendly allied nation however this is an expensive option. The UK MoD acquisition budget to be allocated to this acquisition scenario is £185 million being made as four payments over a three-year time period. Any potential solutions to the acquisition scenario need to deal with all of the above concerns simultaneously rather than trying to solve the concerns one at a time.

This type of acquisition scenario can be considered to be a typical example of a system of sys-

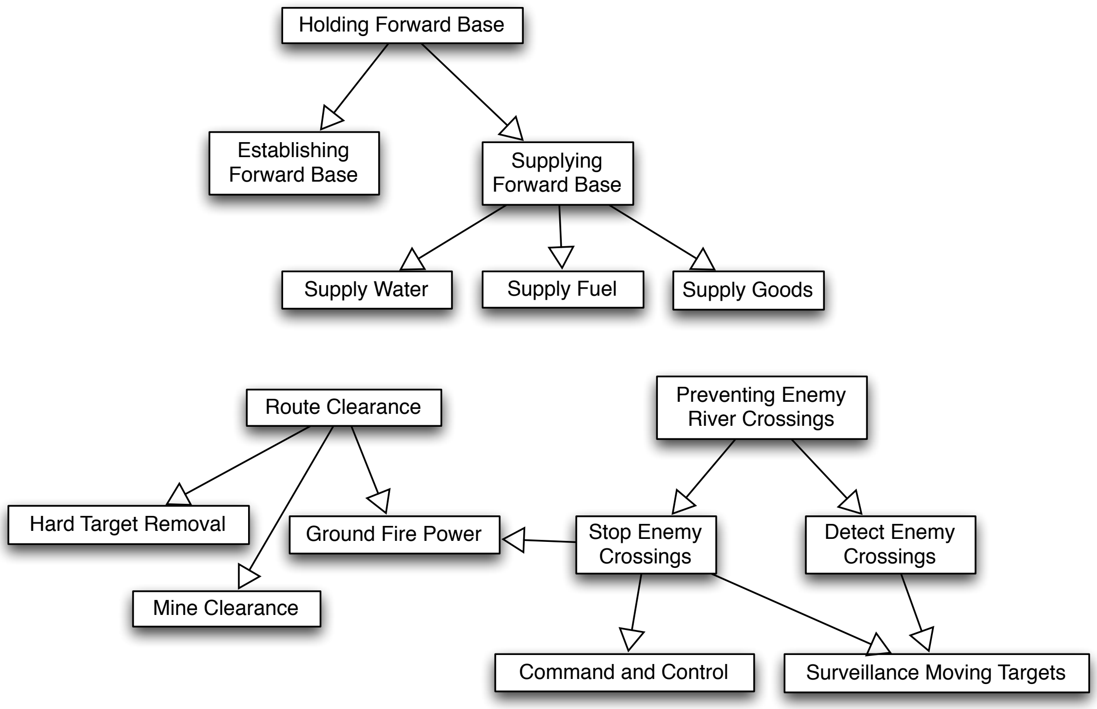


Figure 9.9: Military Acquisition Scenario Case Study Objectives

tem acquisition problem with a multitude of objectives, a large number of heterogeneous systems and numerous constraints on the possible solutions.

The scenario already contains an existing system of systems being used to address the problem. There are currently 14 systems making up the system of systems that are:

- *Mastiff Vehicles.* Mastiffs are 6 wheeled patrol vehicles that can carry up to 8 troops and two crewmembers. They can be fitted with either a mounted heavy machine gun or grenade launcher. [141]

- *Vector Vehicles.* Vectors are patrol vehicles that can carry up to 4 troops and two crewmembers. They can be fitted with two mounted machine guns. [142]

- *Troop Regiments.* A troop regiment represents a large number of trained soldiers with basic equipment and training.

- *L118 Light Gun.* The L118 Light Gun is an artillery piece that can fire a wide variety of ammunition. It needs to be transported by another vehicle. [143]

- *Land Rover 101 Forward Control Vehicles.* The Land Rover 101 Forward Control Vehicle is used by the UK MoD to tow the L118 Light Gun in position and to carry its ammunition [144].

- *Mobile Artillery Battlefield Radar System.* The mobile artillery battlefield radar system is a weapon fire location system. It is able to locate the source firing location of artillery fire, rocket fire and mortar fire. [145]

- *Buffalo Vehicles.* The Buffalo is an armoured mine disposal vehicle. It is heavily armoured and has a controllable arm for dealing with mines or improvised explosive devices. [146]
- *Engineering Team.* In the case study, the engineering team is considered to be the team responsible for setting up the new forward command base along with their supplies.
- *Oshkosh Wheeled Tanker Vehicles.* The Oshkosh wheeled tanker is a militarised delivery vehicle. It can be fitted to carry 20000 litres of fuel or 18000 litres of water. [147]
- *Leyland DROPS Vehicles.* The Leyland Demountable Rack Offload and Pickup System (DROPS) is one of the main logistics vehicles for the UK MoD. The main use for the vehicle is in carrying ammunition. [148]
- *Global Hawk UAVs.* The Global Hawk is an unmanned aerial vehicle (UAV) specialising in intelligence, surveillance and reconnaissance (ISR) [149].
- *Reacher Satellite Internet System.* The Reacher Satellite Internet System is a transportable system that provides Internet via satellite in remote locations. Typically, the Reacher is mounted on a Duro 6x6 vehicle. [150]
- *Mowag Duro III Vehicles.* The Mowag Duro III vehicles are used for carrying the Reacher Satellite Internet System [151].
- *Bowman Radio Communication System.* The Bowman is a digital radio communication system [151].

9.5.2 Baseline Assessment

Normally, in MoD military acquisition before looking at any possible solutions, the first thing to do is to look at the effect of acquiring nothing on a military scenario to decide whether any actions need to be taken at all before looking at acquiring new systems to address a problem [1]. For this case study, we will therefore also perform a baseline assessment of the existing in-place systems before looking at what we can acquire.

The 'Holding the forward base' objective is almost fully satisfied, at 89%. Looking at the corresponding goal model, the reason the satisfaction is not 100% is that there are lower than wanted levels of fuel and good being supplied to the forward base due to insufficient transport vehicles. There is more than the wanted level of water being delivered. These figures are based on the aggregations included within the goal model structure.

The 'Route Clearance' objective is being satisfied at 71% dropping to 69% when the 'Mobile Artillery Battlefield Radar' goes out of service and then dropping to 63% when the 'L118 Existing Service Contract' goes out of service. Insufficient ground forces and insufficient hard target removal are currently limiting the Route Clearance objective. Due to political pressures more troops cannot be acquired to make up for the limited ground forces. There are two options available for consideration. The first option is to spread out more widely some of the SAS training (as suggested by the head of the SAS training programme) to make better use of the troops already on the

[1]Private discussion at System Engineering 1 course, LSCITS Programme, 2012

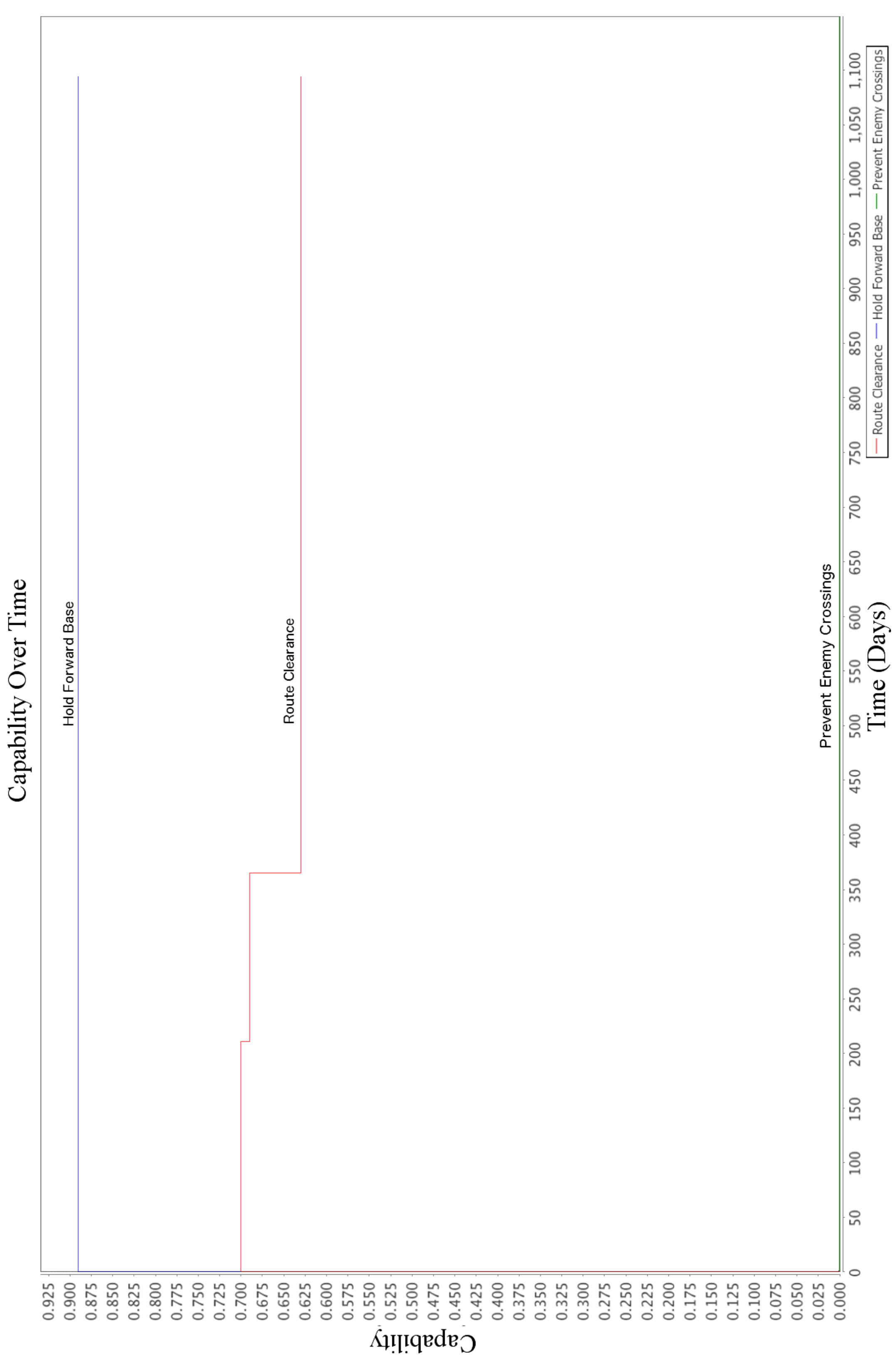


Figure 9.10: Military Scenario - Baseline Capability Over Time

ground. The second option is the acquisition of more vehicles with inbuilt weaponry to give the troops on the ground more firepower.

The 'Prevent Enemy Crossings' objectives are not being satisfied. Looking at the corresponding goal model, the reason it is not being satisfied is that only 40% of enemy crossings can currently be stopped when there is a requirement to stop 70% of enemy crossings for the operation to be considered successful. The limiting factor is not the lack of ability to stop enemy crossings there is sufficient forces available to do that but lack of ability to detect enemy crossings. Currently, only 40% of enemy crossings can be detected hence no more than 40% of enemy crossings can be stopped.

9.5.3 Acquirable Systems

As well as dealing with the acquisition problems listed at the start of the scenario, any potential solutions also need to help address the capability short falls of the existing system of systems.

The following systems can be acquired to help addressing the problems in the scenario:

- *Mastiff Vehicles.*
- *Vector Vehicles.*
- *L118 Light Guns.*
- *Land Rover 101 Forward Control Vehicles.*
- *Global Hawk UAVs.*
- *MQ-9 Reapers.* The MQ-9 Reaper is an unmanned air vehicle (UAV) that has surveillance abilities similar to the Global Hawk however additionally it has the ability to carry and deploy offensive payloads in the form of Hellfire missiles [152].
- *SAS Training Programme.*
- *Mobile Artillery Battlefield Radar Foreign.* This system is a replacement for the Mobile Artillery Battlefield Radar system that has been developed independently in an allied country.
- *L118 New Service Contracts.* This is a new replacement contract for the maintenance of the L118 Light Guns keeping them in service.

9.5.4 Military Acquisition Scenario - Textual DSL Input

The full textual DSL input for the case study is given in appendix B. The problem description begins with defining the meta-heuristic search settings, the time period the problem is to be evaluated over, the existing systems and acquirable systems that can be used in solutions and the budget available whilst scheduling solutions. The existing systems and acquirable systems are annotated with temporal information such as the date they came into service and date they will leave service in the existing system case and the time taken to acquire them and the lifespan in the acquirable system case. The problem description defines the goal tree structure for helping to evaluate any potential solutions. A graphical overview of the goal-tree decomposition is shown in figure 9.9. The problem description also defines details about the existing systems and the acquirable systems.

9.5.5 Runtime Information

The case study contains 25 types of different systems with multiple copies present for most of them. The case study problem is of a realistic size and takes approximately 70 minutes to compute on a MacBook Air Mid 2011 (Intel Core i5 1.7Ghz, 4GB 1333 MHz DDR3) with population size 200 and generation count 200. It is therefore unlikely that runtime will be a serious issue in most envisaged uses of CATMOS. Currently, a capability investigation using a workshop based process like TRAiDE [50] takes 6 months hence an extra 70 minutes spend on running the tool after the problem has been input is insignificant. Using better computing equipment or multiple computers would further reduce the runtime. The case study is based off a realistic military scenario provided by a domain expert and therefore should be of a realistic size.

9.5.6 Results

When the CATMOS prototype tool is run on the problem a Pareto front consisting of one point was found. In the solution the three objectives, route clearance, holding the forward base and preventing enemy crossings, were satisfied at 91.3%, 89.5% and 43.2% respectively. The small size of the Pareto front is to be expected because, unlike in the earlier Multi-objective Next Release Problem case study, the cost is a merely a constraint rather than an objective. There is a budget to help fulfil the objectives rather than we are trying to find the best option at each budget level so the Pareto front generated is just competition between the satisfaction of the three objectives. The reason why it is likely to have been reduced to a single result is because the three objectives have a great deal of crossover in their needs and so are partially fulfilled by the acquisition of the same components. Another run of the prototype tool found 2 results on the Pareto front but both are slightly worse than the result shown here. Solely for comparison with the earlier MONRP case study, a Pareto front with the three main objectives being summed together as one objective and minimising the overall costs as the other objective is shown in figure 9.14 and this Pareto front has 21 points.

A corresponding capability over time graph, Gantt chart and goal model is generated for all the solutions (in this case only one). The capability over time and Gantt chart are shown in figures 9.11 & 9.12. A simplified version of the goal model omitting the full details, due to size issues with fitting the full goal model on a single page, is shown in figure 9.13. Most of the useful information about the acquisition plan, such as when components are acquired and go out of service and how this affects the amount of capability over time, can be gathered from the capability over time chart and the Gantt chart.

For the simplified goal model, existing components with the same name and acquired components with the same name have being merged together. The numbers at the end of the component's title indicate the individual systems that have being merged together i.e. SAS Training : 1, 3, 4 means there were 3 separate SAS training components that were acquired. The numbers simply refer to the original labels of the copies of the component in the goal model and an E in front of a number means the components were not acquired but existed at the beginning. Cost information has also being omitted from the goal model. This has been done simply for presentational reasons. The Gantt chart in figure 9.12 also uses the same numbering conventions for presentational reasons.

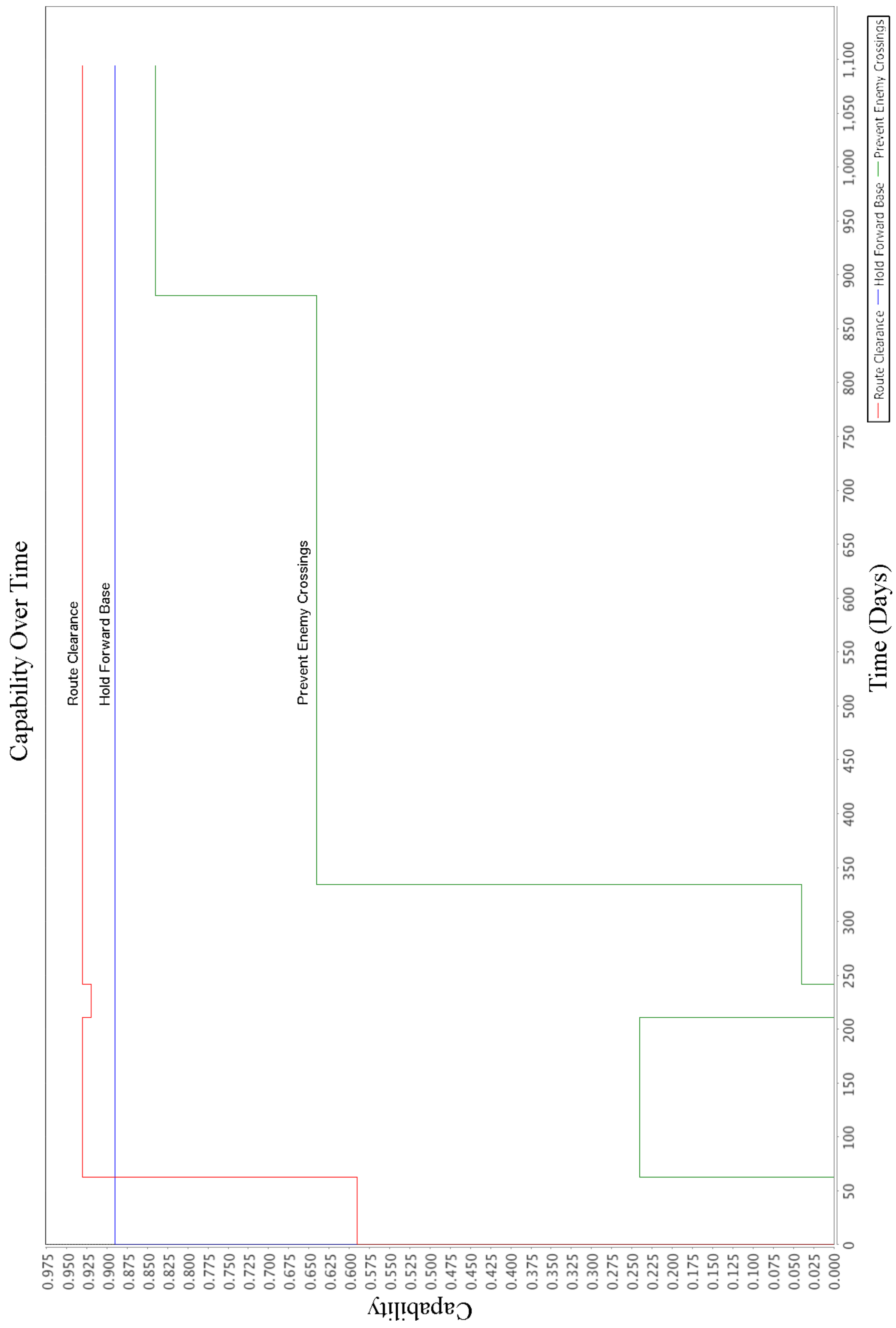


Figure 9.11: Military Scenario - Scenario Capability Over Time

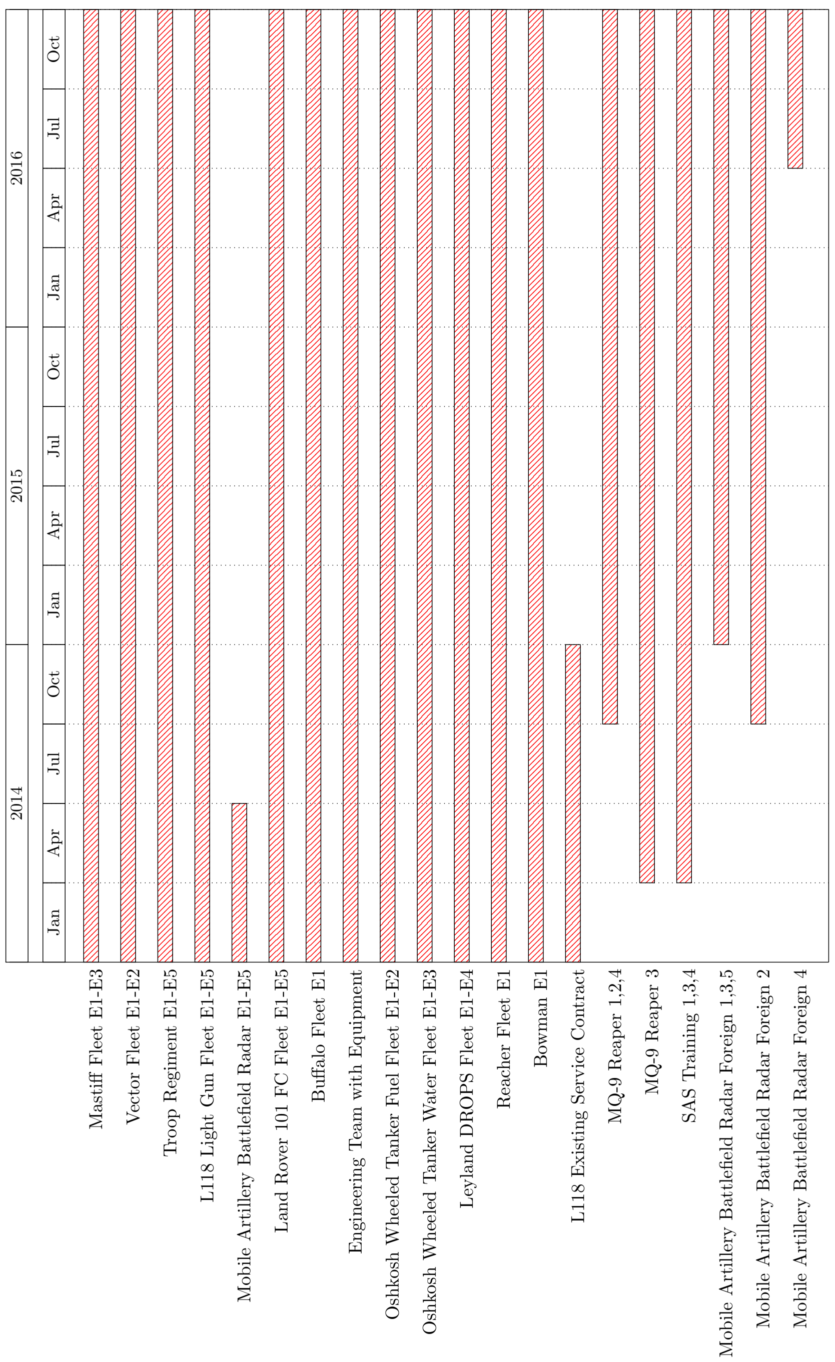
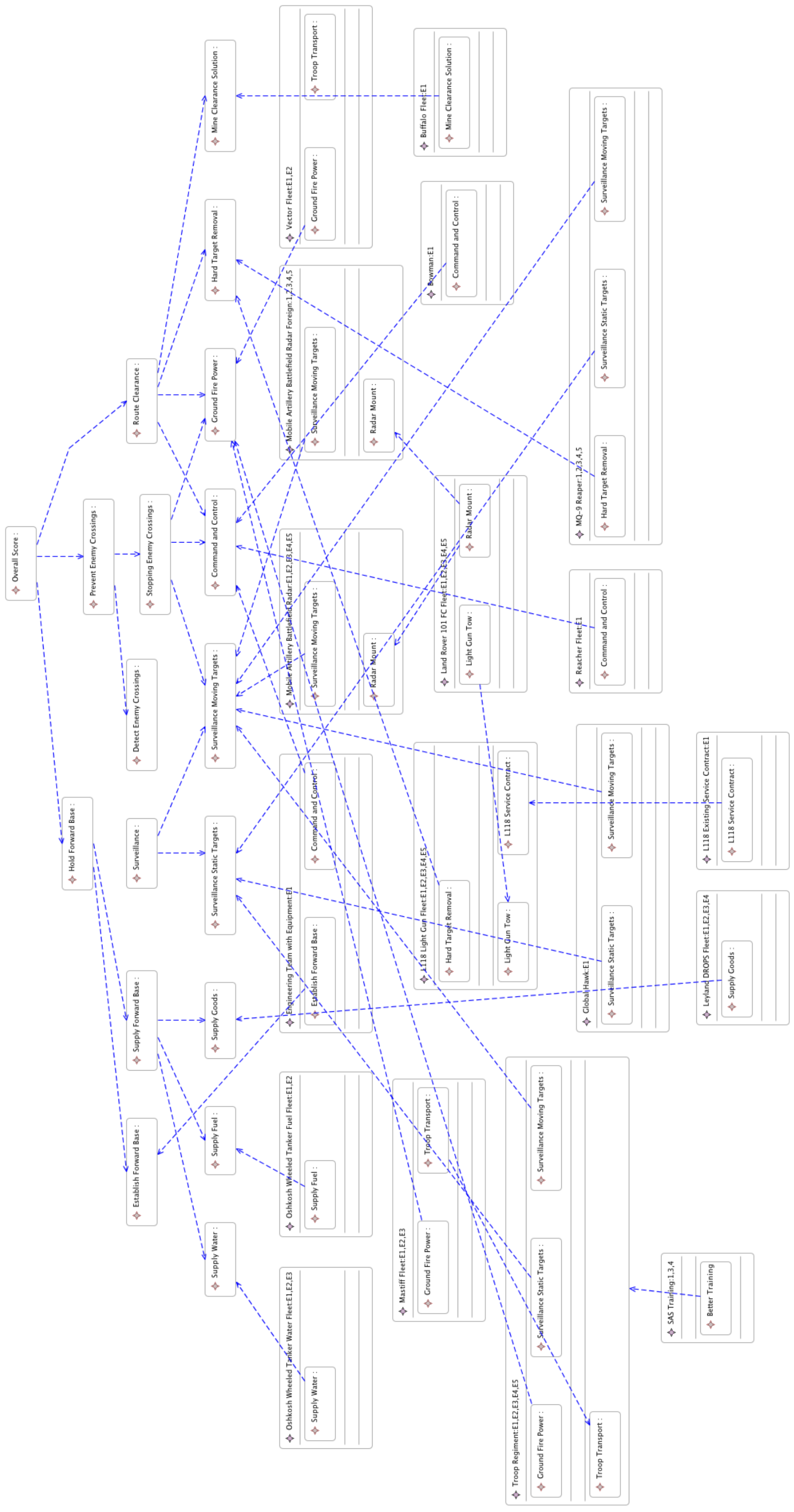


Figure 9.12: Military Scenario - Scenario Gantt Chart



Overall Score :
Prevent Enemy Crossings :
Route Clearance :
Stopping Enemy Crossings :
Detect Enemy Crossings :
Hold Forward Base :
Surveillance :
Supply Forward Base :
Establish Forward Base :
Mine Clearance Solution :
Hard Target Removal :
Ground Fire Power :
Command and Control :
Surveillance Moving Targets :
Surveillance Static Targets :
Supply Goods :
Supply Fuel :
Supply Water :
Vector Fleet:E1,E2
Troop Transport :
Buffalo Fleet:E1
Bowman:E1
Mobile Artillery Battlefield Radar Foreign:1,2,3,4,5
Radar Mount :
MQ-9 Reaper:1,2,3,4,5
Land Rover 101 FC Fleet:E1,E2,E3,E4,E5
Light Gun Tow :
Mobile Artillery Battlefield Radar:E1,E2,E3,E4,E5
Reacher Fleet:E1
Engineering Team with Equipment:E1
L118 Light Gun Fleet:E1,E2,E3,E4,E5
L118 Service Contract :
Global Hawk:E1
L118 Existing Service Contract:E1
L118 Service Contract :
Leyland DROPS Fleet:E1,E2,E3,E4
Oshkosh Wheeled Tanker Fuel Fleet:E1,E2
Mastiff Fleet:E1,E2,E3
SAS Training:1,3,4
Better Training
Oshkosh Wheeled Tanker Water Fleet:E1,E2,E3
Troop Regiment:E1,E2,E3,E4,E5

Figure 9.13: Military Scenario - Scenario Simplified Goal Model

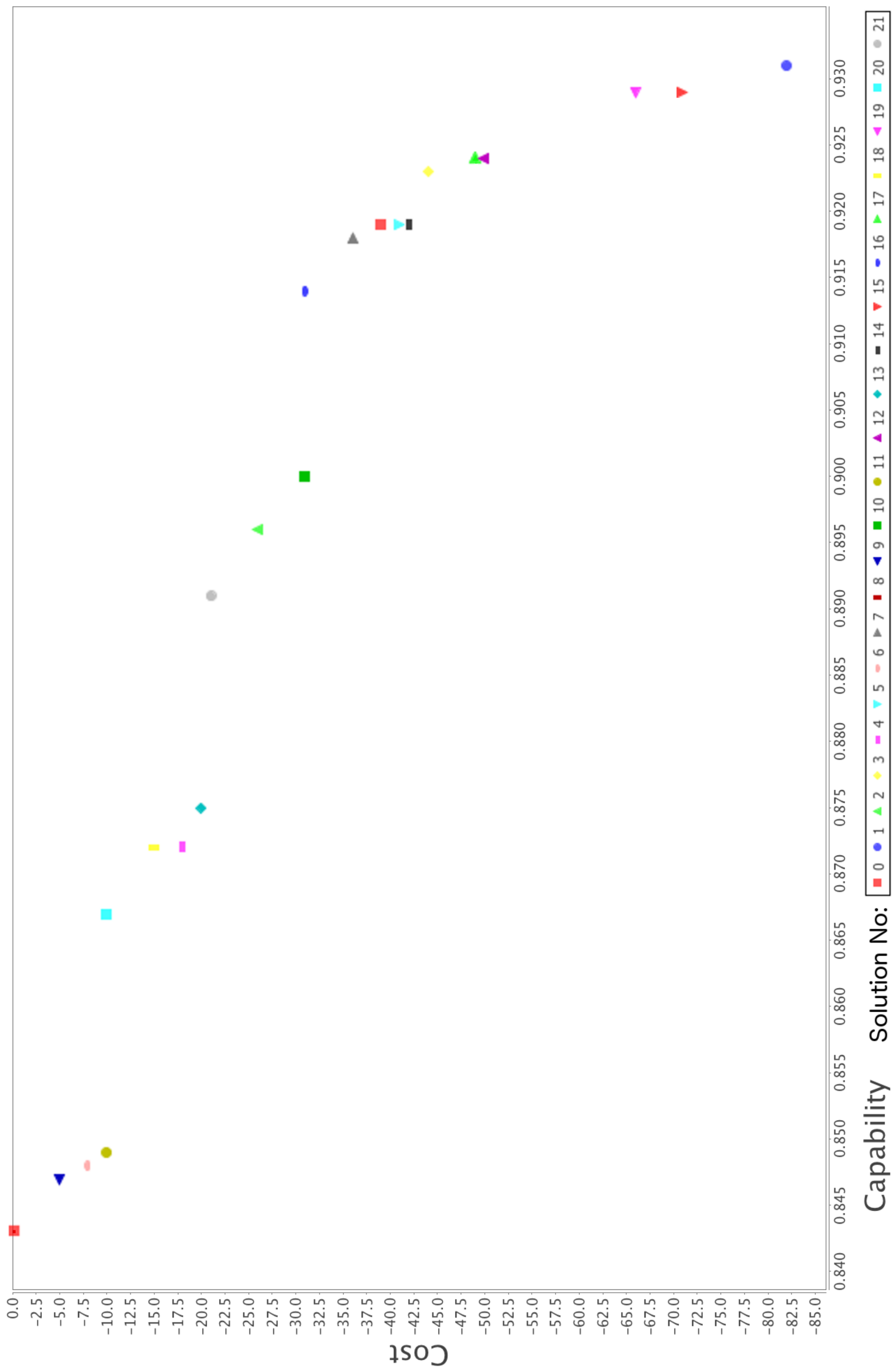


Figure 9.14: Military Scenario - Pareto Front of Overall Capability vs. Overall Costs

9.5.7 Case Study Conclusions

The CATMOS technique can handle highly complicated system of system acquisition problems. It is able to take a problem definition, system definitions, system dependency information and budgetary constraints and generate a Pareto front of near optimal solutions for the decision makers to choose from. Each answer has a corresponding capability over time graph, Gantt chart and goal model that can be examined by the decision maker. The goal models allow the decision makers to study the computer-generated solution and identify any possible problems in the solution.

Earlier work by Symes & Daw [50] on TRAiDE, described the concept of an Integrated Management Plan. An Integrated Management Plan is a Gantt chart of acquisition problems overlaid by capability over time graphs. A major problem with the Integrated Management Plan was there was no known working way to generate capability over time graphs, as there was no known objective method for mapping from the capabilities to DLoD or vice versa. The problems found in TRAiDE motivated this research and this research has now addressed this problem by being able to produce the wanted capability over time graphs.

The CATMOS technique has been shown on a realistic case study to be sophisticated enough to handle realistic military acquisition problems. There were no issues in applying the CATMOS technique itself to the case study problem. Though a couple of implementation specific bugs did need to be fixed in the prototype tool such as reimplementing the capability accumulation feature in a more computationally efficient manner.

Chapter 10

Implementation Details

10.1 Overview

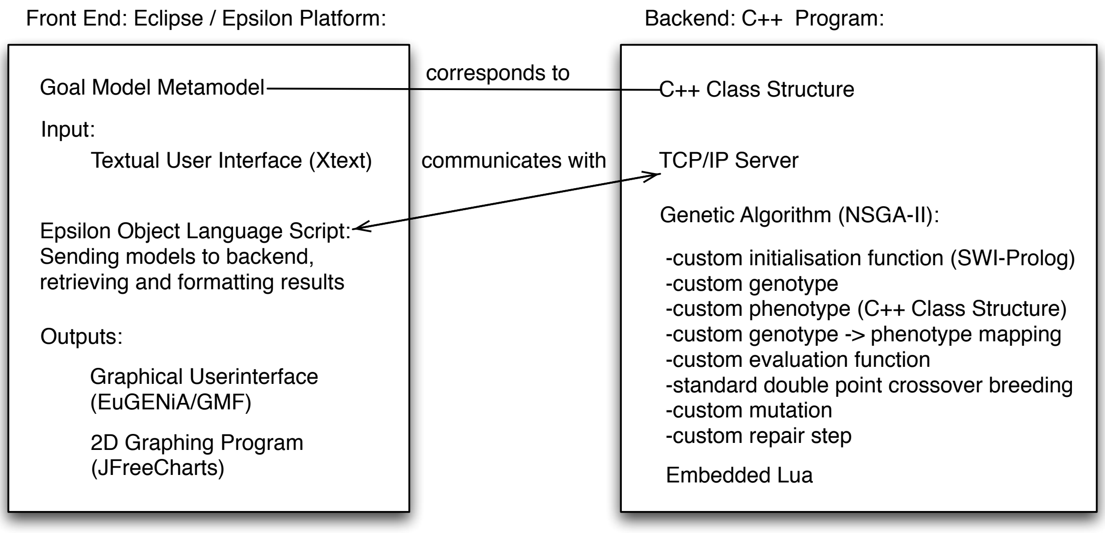
In this chapter, we will cover some of the implementation specific details for the prototype tool support for CATMOS, as some parts of it are novel in their own right. An architecture diagram for the CATMOS prototype tool is shown in figure 10.1. The current prototype is split into a front end and a back end. The front end contains the user interface, while the back end performs the vast majority of the problem solving. The front end was split from the back end as a deliberate engineering choice.

This split allows the user interface to be written in high-level languages for ease of implementation, whilst the back end can be written in low level programming languages for performance. The performance is important as the technique uses meta-heuristic search that evaluates thousands of candidate solutions; hence for the technique to be practical it needs good performance.

This also allows the front end graphical user interface to be easily upgraded or replaced if the work was to be commercialised whilst keeping the same back end logic intact.

10.2 Front end

The front end is written using Epsilon platform languages [101] such as Epsilon Object Language (EOL) [153], Flock [108] along with Xtext [133] and Java. The front end was written with these languages because they allow the easy manipulation and display of models and the CATMOS technique functions by creating goal models from model fragments. The tasks performed by the front end are taking in the problem description, performing basic error checking on it, passing it to the back end using a telnet-like protocol, retrieving the results from the back end once the search has completed and displaying the results to the user. When searching for trade-offs without considering the scheduling of the solutions, the results are a Pareto front, on which each point is labelled with the number for a corresponding goal model, and a corresponding goal model for each solution. When using the through life extension the results are the same plus additionally a capability over time chart and Gantt chart for each goal model providing the temporal information to the user. The prototype tool uses JFreeCharts [154] for generating its charts, though this could be easily replaced with a different library.



Front End: Eclipse / Epsilon Platform:
Backend: C++ Program:
Goal Model Metamodel
corresponds to
C++ Class Structure
Input:
Textual User Interface (Xtext)
communicates with
TCP/IP Server
Genetic Algorithm (NSGA-II):
Epsilon Object Language Script: Sending models to backend, retrieving and formatting results
-custom initialisation function (SWI-Prolog)
-custom genotype
-custom phenotype (C++ Class Structure)
-custom genotype -> phenotype mapping
-custom evaluation function
-standard double point crossover breeding
-custom mutation
-custom repair step
Outputs:
Graphical Userinterface (EuGENiA/GMF)
2D Graphing Program (JFreeCharts)
Embedded Lua

Figure 10.1: Prototype Tool Architecture Overview

In this thesis, MDE is not being used for general software development; instead we are exploiting the model manipulation abilities of the MDE techniques to allow us to manipulate the structures of goal models to automatically find acquisition trade-offs. To use MDE techniques a metamodel for the goal models we wish to manipulate must first be defined. The core metamodels used for this are shown in figures 10.2 and 10.3 and were implemented using Emfatic [155], which creates an Ecore metamodel [106].

The front end, which is built inside an Eclipse [105] workspace, currently has two user interfaces available. The first is the graphical user interface that was implemented semi-automatically by using EuGENia [101] and Eclipse Graphical Modelling Framework (GMF) [132]. EuGENia takes in an Ecore metamodel with some minor extra annotations and automatically generates a GMF editor, which is a graphical user interface. While quick to generate, the editor is quite hard to customise as it uses 4 different interlinked metamodels to specify the GUI layout and has little update support for when the original metamodel changes.

The second is a textual user interface that was implemented semi-automatically by using Xtext [133]. Xtext takes in an Ecore metamodel and automatically produces a corresponding working textual syntax. The developer can then easily customise this textual syntax. Xtext automatically provides an editor for the domain specific language including syntax highlighting.

These modelling tools were initially used because the core of the CATMOS technique relies on combining model fragments together. In the first versions of the prototype tool, the combining was implemented using modelling operations, such as model transformation. These tools were used because of their ability to quickly define domain specific languages in a form that end users are able to use. The modelling tools are still used in the display and presenting of the information to the acquisition decision makers however the operational part is now written in C++ for performance reasons.

Both user interfaces work with the same Ecore models and are hence interchangeable. At the moment, the Xtext textual user interface is the default for creating a problem and the GMF editor is the default for viewing the solutions.

Originally, most of the operational part of the prototype tool for the approach was also written using model-to-model transformation languages such as Epsilon Transformation Language [107] and Flock [108]. These were eventually phased out in place of a C++ back end that performs the main work of the prototype tool. This was done because the languages provided by Epsilon are interpreted rather than compiled and coupled with the use of meta-heuristic search this causes performance issues. The CATMOS prototype tool still uses Epsilon Object Language [153] for performing tasks such as checking the model and loading the model into the C++ back end and loading the results back from the C++ back end into the model format to allowing the results to be viewed via the graphical user interface.

Flock [108] is still used for helping in the formatting and displaying of the resulting models received from the back end. Flock is not a general-purpose model-to-model transformation language like ETL but instead is designed for the situation where metamodels are upgraded [108]. When a metamodel is upgraded, old models that used to confirm to the metamodel may no longer conform to the metamodel and be invalid [108]. Flock is a tool for migrating these now invalid models to the new metamodel [108]. From the author's own experience Flock is a lot quicker and far more concise to use than ETL for specifying simple transformations between similar metamodels as it assumes information is copied directly between all non changed parts of the metamodels between the source and target metamodel. Through it should be noted that Flock doesn't contain the same type of expressive power as ETL. Since most of the transformations currently used are for preparing the model information for display purposes and therefore move information between very similar metamodels, these transformations are written using Flock.

Another tool that was considered for visualising the goal models was Graphiti [156], which could have provided a GUI editor similar to EuGENia however at the time of use the tool was insufficiently mature to be of practical use.

10.3 Back end

The back end is responsible for combining the *Goal Tree* and *Components* together to produce *Goal Models* and performing the multi-objective search over the possible *Goal Model* structures and dealing with any through life constraints and scheduling of the solution if applicable. The back end listens on a TCP/IP port for connections and uses a simple telnet like protocol for accepting the problem specification generated by the front end. Once, the multi-objective search is finished the back end hands back the solutions on the Pareto front to the front end.

10.3.1 Back end Performance

The back end is written in C++ for performance reasons. Originally, the work performed by the back end was performed by the now current front end using languages from the Epsilon platform [101]; however, these languages are interpreted, and hence rather slow with some searches taking multiple days to complete. When the logic was rewritten in C++, an approximately 300x performance increase was obtained.

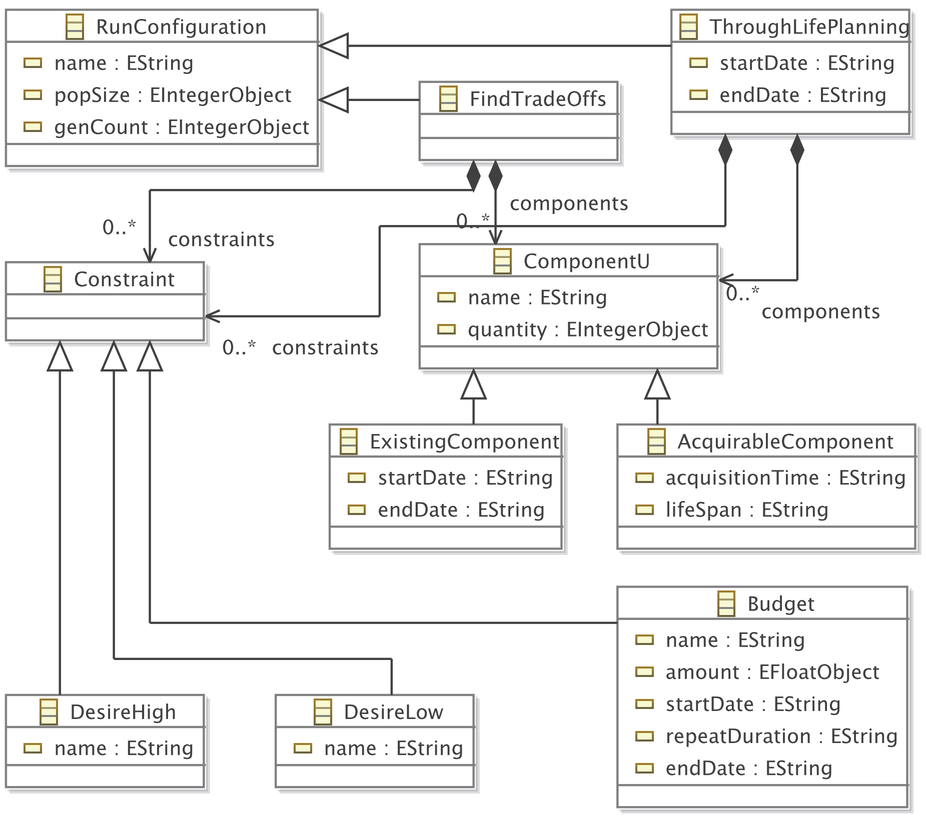


Figure 10.2: CATMOS Technique Full Meta-Model - Acquisition Settings (Repeated)

10.3.2 Metamodel in C++

The front end and the back end both use the same underlying metamodel. In the front end the metamodel is implemented using EMF [106] and in the back end the metamodel is implemented using an object oriented programming class structure.

10.3.3 NSGA-II

The prototype tool only partially implements the NSGA-II algorithm [119]. NSGA-II contains a crowding distance to spread out solutions over the Pareto front. Usually, NSGA-II is run on problems with continuous solutions spaces where there can an infinite number of different solutions close together on the Pareto front. In this work, where there are discrete components making up the solutions, the points on the Pareto front are not continuous hence the full crowding distance algorithm is not needed. Instead, a penalty is added for penalising exactly identical solutions on the Pareto front, which causes the solutions to spread out along the Pareto front, as the space is not continuous.

10.3.4 Scheduler

The prototype tool back end contains a scheduler for dealing with through life problems. Scheduling is performed after the genotype is converted to the phenotype and the evaluation is run multiple times at different times to get the changes in capability over time as components come into and go

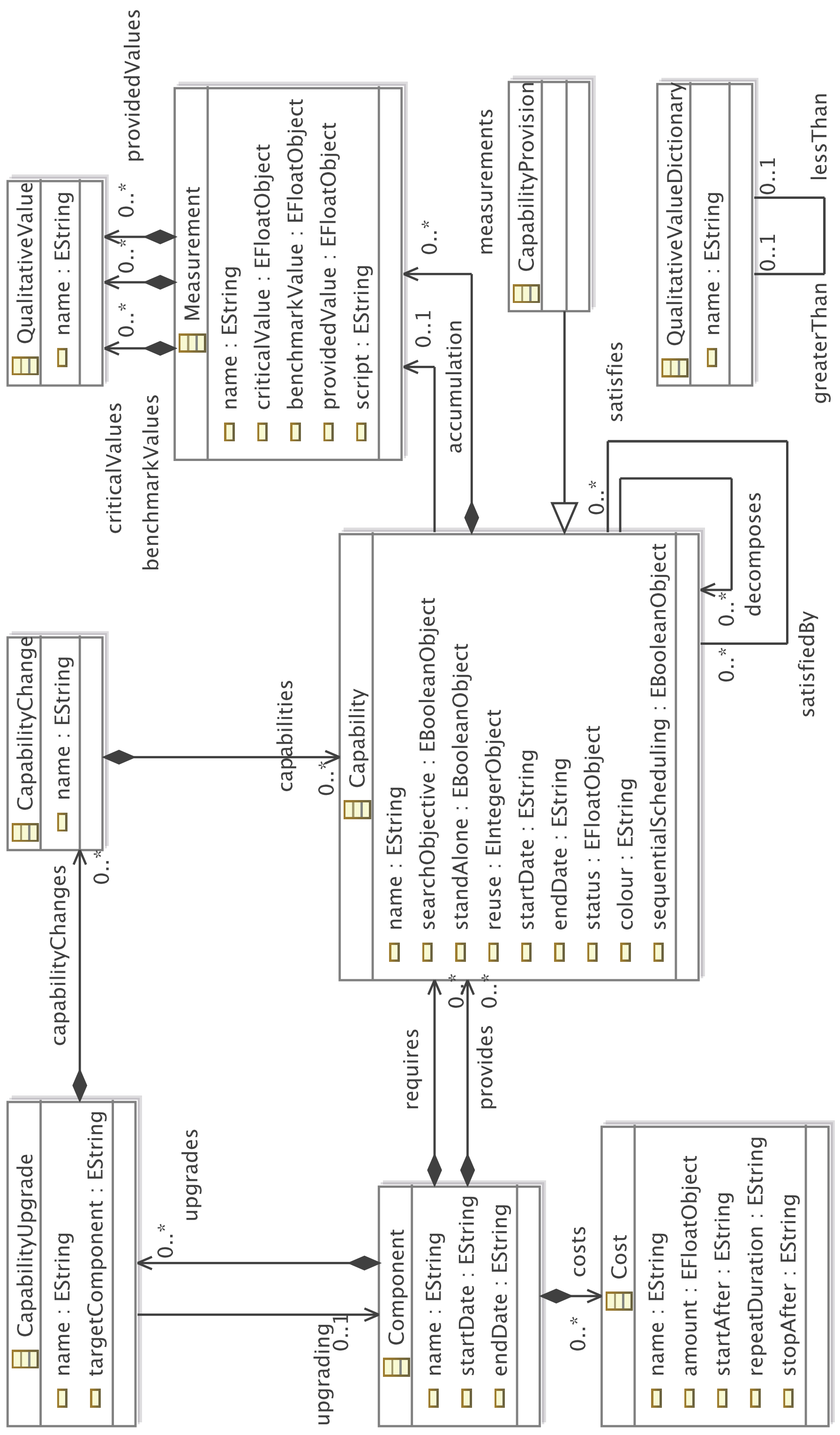


Figure 10.3: CATMOS Technique Full Meta-Model - Capabilities and Components (Repeated)

Function	Description
getCapability (string capabilityName)	Returns the satisfaction value of the capability.
getMeasure (string measurementName)	Returns the provided value or provided values for a measurement within in the same component as the caller.
getGlobalMeasure (string measurementName)	Returns the provided value or provided values for a measurement anywhere in the goal model.
getScenarioMeasure (string measurementName)	Returns the provided value or provided values for a measurement in the goal tree.
getAllMeasures (string measurementName)	Returns as a table with an entry for each measurement matching the name the provided value or provided values.
countComponent (string componentName)	Returns the number of that component included in the current solution.
canUseCapability (string capabilityName)	Returns if the capabilities dependencies are all met.

Table 10.1: Functions made available by the prototype tool to Lua

out of service. The scheduler's algorithm is defined in chapter 9.

10.3.5 Scripting

The technique requires the use of a scripting language to allow formulas to be specified on the goal model. Lua [128, 129] as described in section 7.2.2 was chosen for this purpose but almost any programming language would be suitable for this task. Lua allows the application it is embedded into to provide function hooks that allow the application and Lua to communicate with each other [128, 129]. In this case, the scripts placed on the goal model are able to access other parts of the goal model during their execution. Functions are provided by CATMOS to the Lua scripts to allow them to interact with the goal model. These functions along with descriptions of them are listed in table 10.1.

A Lua script can be added to any *Measurement* and then any time the *Measurement* is evaluated the Lua script is executed. The Lua script can change the providedValue or providedValues field of the *Measurement* it is attached to by setting the value 'output' to a value or a table of values.

For convenience and to avoid the unnecessary character escaping of Lua functions written directly in the textual DSL, the Lua functions can also be defined in a separate script file alongside the textual DSL and be loaded automatically by the prototype tool.

10.4 NSGA-II Genetic Algorithm - Implementation Details

This section extends the explanation given in section 8.2 of the customisation of the NSGA-II algorithm for the CATMOS technique, with implementation details for the custom population initialisation and the phenotype evaluation algorithms.

10.4.1 Custom Population Initialisation

The CATMOS technique uses a back tracking parser to create its initial solutions. Rather than implementing a backtracking parser from scratch, the problem of creating genotypes that give complete *Goal Models* has been formulated using SWI-Prolog [157]. The *Goal Tree* and *Component* models used as the tool's input are transformed into production rules for the parser and the parser's output is converted into the genotype for use with the multi-objective search algorithm. The population initialisation algorithm was implemented using Prolog because the problem can be easily specified as Prolog production rules and Prolog automatically provides the backtracking capabilities required by the algorithm.

The Prolog code for this is presented below. The problem specific part of the Prolog code is the desired capabilities, the decomposition list, the component list, the capability provision list and the capability requirement list. The problem specific part is generated automatically for the current problem where the rest of the Prolog code is static and used between problems. The Prolog code works by starting with the leaf goals in the goal model and then deriving satisfying components using the *providesCapability* relationship. Thereafter it derives through the *requiresCapability* links into more components until the tree is fully derived. As it derives through the tree the connections that are required for the tree derivation that is being made are stored in an accumulator and the path chosen through the tree is randomised using the *random_member* function. The first time a component upgrade is used to provide a capability, the usage of the upgrade is asserted into the *upgrade* relationship.

10.4.2 Custom Population Initialisation - Prolog Code

```
%Desired Capabilities
desiredCapabilities(['Capability1']).
%Decomposition List
capabilityDecomposes('Capability1','Capability2').
%Component List
component('Component1').
component('Component2').
%Capability Provisions List
providesCapability('Component1',['Capability1'],'blank').
providesCapability('Component2',['Capability2'],'blank').
%Capability Requirements List
requiresCapability('Component1',['Capability2']).
%Static Functions
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
canProvideCap(Comp,Cap,Acc) :- findall(C,canProvideCapNormal(C,Cap,_),R),
    random_member(Comp,R),findall(Upgrade,canProvideCapNormal(Comp,Cap,
       Upgrade),Upgrades),random_member(Upgrade,Upgrades),createUpgrade(
       Upgrade,Acc).
17  canProvideCapNormal(Comp,Cap,Upgrade) :- providesCapability(Comp,List,
       Upgrade), member(Cap,List).
18  createUpgrade('blank',[]).
19  createUpgrade([H,H2|_],Acc) :- assert(H), canSatComp(H2,Acc).
20  canSatCap(Cap,Acc) :- canSatCap(Cap,Acc,scenario).
21  canSatCap(Cap,Acc,scenario) :- capabilityDecomposes(Cap,DCaps), canSatCaps(
       DCaps,Acc,scenario).
22  canSatCap(Cap,[(DComp,Cap,Comp),AccH|AccT],DComp) :- canProvideCap(Comp,Cap
       ,AccH), canSatComp(Comp,AccT).
23  canSatComp(Comp,AccT) :- requiresCapability(Comp,Caps), canSatCaps(Caps,
       AccT,Comp).
24  canSatCaps([],[],_).
25  canSatCaps([Cap|T],[AccH|AccT],Comp) :- canSatCap(Cap,AccH,Comp),
       canSatCaps(T,AccT,Comp).
26  canMeetScenario(Acc) :- abolish(upgrade/3), dynamic(upgrade/3),
       desiredCapabilities(X), canSatCaps(X,Acc,scenario).
27  getUpgrades(R) :- findall((A,B,C),upgrade(A,B,C),R).
28  getSolution :- canMeetScenario(X), open('output.txt',write,Stream),write(
       Stream,X),nl(Stream),getUpgrades(Y),write(Stream,Y),close(Stream),halt.
```

The top of the Prolog Code (lines 1-12) is used for specifying the problem. The desired capabilities from the top-level goal model are specified on line 2 as a list. Their decompositions into sub-capabilities are specified using the *capabilityDecomposes* relationship (line 4), with the first argument being the capability name and second argument being the sub-capabilities name. The line is repeated for each decomposition. All available components are stated in the component relationship shown on lines 6 & 7. The capabilities provided by a component are stated using the *providesCapability* relationship (lines 9 & 10). The first argument is the component's name, the second its capability provisions and the third is the upgrade, if any, that is used to provide the component with those capability provisions. The capability dependencies are stated using the *requiresCapability* relationship, the first argument being the component's name and the second argument being a list of needed capabilities.

The bottom half of the Prolog code (lines 13 and down) is used for specifying the Prolog production rules that encodes the custom initialisation algorithm. Lines 14 & 15 just encode the membership test, which is whether a given term is located inside a list of terms or not. The Prolog production rules begin executing on line 28, with the *getSolution* production rule. This rule formats the output of the Prolog run into a form that can be changed later on into a genotype. This formatting rule calls the main production rule on line 26, that query whether all the desired capabilities can be met. This calls the *canSatCaps* production rules (lines 24-25), that sets up a Prolog accumulator that attempts to see if each desired capability can be satisfied by calling the *canSatCap* production rules for each of the desired capabilities. The *canSatCap* production rules (lines 20-22) query whether a given capability can be satisfied. The line 21 of the *canSatCap* production rules converts a query for if a capability in the top-level goal model can be satisfied into a new query for if all of the capabilities sub-capabilities can be satisfied instead. The line 22 of the *canSatCap* production rules changes a query for whether a capability can be satisfied into a new query asking whether there is a component that provides the capability and whether this component's capability dependencies can be satisfied.

Whether there is a component that can provide the capability is addressed by the *canProvideCap* and *canProvideCapNormal* production rules that find all of the satisfying components for a capability and choose one of these components at random. If an upgrade is used to satisfy this capability, then this upgrade is added to a list and whether the component providing the upgrade can be satisfied is then queried. The production rule *canSatComp* on line 23 handles whether a component can be satisfied by creating a new query on whether all of the capability dependencies can be satisfied.

10.4.3 Phenotype Evaluation Algorithm

As described in section 8.2 the phenotype needs to be evaluated by the search technique.

Firstly, during evaluation all *Components* and *Capabilities* have a status value that can be between 0 & 2. The meaning of the various values is given below:

Status Value	Meaning
0	Not satisfied
0-1	Partially satisfied but a dependency not met
1-2	Partially satisfied
2	Fully satisfied

The evaluation algorithm used to evaluate the phenotype is as follows.

```
1: for all Capabilities do
2:     Set capability status value to 0.
3: end for
4: for all Components do
5:     Set component status value to 0.
6: end for
7: for all Components do
8:     if Component has no dependencies then
9:         Set the component's status to 2
10:    end if
11: end for
```

The first part of the algorithm sets all status values to their starting positions. Initially, all status values are zero except where these is a component that has no dependencies and hence is fully satisfied. The algorithm in the next part will propagate the status values up the goal tree where applicable from the components with no dependencies until the status values reach the top of the goal tree.

```
12: while Changes are still being made do
13:     for all CapabilityProvisions do
14:         if CapabilityProvision is attached to Component with a status value greater than 1 then
15:             Copy the CapabilityProvision's Measurement's providedValues to all attached satisfying Capabilities' Measurement's providedValues.
```

```
16:         **end if**
17:     **end for**
18:     **for all** Capabilities **do**
19:         **for all** of the Capabilities' Measurements **do**
20:             Evaluate any attached Lua script.
```

The second part of the algorithm starts a while loop that keeps repeating until there are no further changes that can be made to the goal tree. All of the component provisions that are attached to a satisfied component have their measurements copied up into the capabilities they satisfy. Then these capabilities satisfactions are then evaluated. This involves checking the satisfaction level of each of the capabilities measurements and calling any attached Lua scripts.

```
21:             Calculate the *Measurement's* satisfaction value as follows.
22:             (Note that whether the benchmarkValue is higher or lower than the criticalValue
    decides whether the providedValue is desired to be high or low.)
23:             **if** benchmarkValue $\geq$ criticalValue **then**
24:                 **if** providedValue $<$ criticalValue **then**
        Measurement satisfaction status = 0.
25:                 **end if**
26:                 **if** providedValue $\geq$ criticalValue **then**
27:                     Measurement satisfaction = (providedValue - criticalValue) / (benchmark-
    Value - criticalValue)
28:                 **end if**
29:             **end if**
30:             **if** benchmarkValue $\leq$ criticalValue **then**
31:                 **if** providedValue $>$ criticalValue **then**
32:                     Measurement satisfaction status = 0.
33:                 **end if**
34:                 **if** providedValue $\leq$ criticalValue **then**
35:                     Measurement satisfaction status = 1.0 - (providedValue - benchmarkValue)
    / (criticalValue - benchmarkValue)
36:                 **end if**
37:             **end if**
38:         **end for**
39:         Evaluate the satisfaction level of the *Capability*, which is the average satisfaction level
    of all of its Measurements.
40:         **if** All the measurement's satisfactions are above 0 **then**
41:             Add one to the capability's satisfaction status.
42:         **end if**
43:     **end for**
```

Once the capabilities satisfaction have been evaluated, the components are then all re-evaluated to see if their capability dependencies are now satisfied and if so they become satisfied as well. The algorithm continues to loop until it goes through the goal tree once and finds there was no change to make. Once this happens the loop stops and the capability status values for the search objective capabilities are send to the multi-objective search algorithm.

```
44:     for all Components do
45:         if All capability dependencies are fully satisfied then
              Set the component's satisfaction status to 2.
46:         end if
47:         if All capability dependencies are partially satisfied then
              Set the component's satisfaction status to 1 + (number of fully satisfied capability dependencies / number of all capability dependencies).
48:         end if
49:         if Some capability dependencies are not satisfied then
              Set the component's satisfaction status to (number of fully satisfied capability dependencies / number of all capability dependencies
50:         end if
51:     end for
52: end while
53: Read out the status values for the search objective capabilities and pass them to back to the multi-objective search algorithm.
```

10.5 Metamodel Explanation

To use MDE, a metamodel must be defined for all modelling languages that are to be processed. The goal model metamodel is defined in figures 10.2 & 10.3. Using the information provided by the metamodel, EuGENia [131] and GMF [132] are able to generate a graphical user interface for the prototype tool and Xtext [133] is able to generate a textual interface for the prototype tool. Following on from the conceptual explanation of the technique in chapters 7, 8 & 9, a technical explanation of the metamodel is provided here:

10.5.1 Run Configuration

The *RunConfiguration* contains the name of the problem being run and the population size and generation count to use for the search's generic algorithm. This class is normally never created but instead either *FindTradeOffs* or *ThroughLifePlanning* is created instead. The CATMOS technique currently runs in one of two modes. The first mode described in chapter 8 is the find trade-off mode that trade-offs between achieved capability and incurred costs and the second mode described in chapter 9 is the through life mode that schedules an acquisition over time with budgetary constraints.

10.5.2 Find Trade-offs

Creating a *FindTradeOffs* object sets the prototype tool's mode to "Find Trade-offs". In this mode the tool finds the Pareto front of the trade-offs between the *Capabilities* set to be search objectives and the costs, which are either set to be desired high with *DesireHigh* or set to be desired low with *DesireLow*.

10.5.3 Through Life Planning

Creating a *ThroughLifePlanning* object sets the prototype tool's mode to "Through Life Planning". In this mode the tool attempts to schedule the acquisition of *Components* to maximise the amount of capability produced through life whilst keeping within the budget constraints specified by the *Budget* class.

10.5.4 ComponentU

The *ComponentU* class is the basic class for saying what is to be used within the prototype tool's run. The name field is the name of the *Component* to be included in the run and the quantity field is how many times it should be included in the run. This class is not normally created but instead an *ExistingComponent* or *AcquirableComponent*, which inherit from this class is created instead.

10.5.5 Existing Component

The *ExistingComponent* class is for specifying components that already exist. These components can optionally have a start date of when they were acquired and optionally an end date of when they go out of service, which are used in through life planning mode.

10.5.6 Acquirable Component

The *AcquirableComponent* class is for specifying components that can be acquired during the tools run. They can optionally have a time taken to acquire them and a life span of how long after acquired they last for, which are used for scheduling them in through life planning mode.

10.5.7 Desire Low

The *DesireLow* class is for specifying that a cost is desired to be as low as possible. *Desire Low* and *Desire High* are separately defined classes because they affect all *Costs* of their cost type.

10.5.8 Desire High

The *DesireHigh* class is for specifying that a cost is desired to be as high as possible.

10.5.9 Budget

The *Budget* class is used in through life planning mode. It specifies when and how much of a type of cost becomes available. The name field should match a name field used in a *Cost* object defined elsewhere in the model. The amount, start date, repeat duration and end date are the same as in the *Cost* class apart from these are resources being provided as oppose to being taken away.

10.5.10 Capability

A *Capability* represents a need or a provision of some ability of interest. *Capability* can be used in three contexts. The first context is as a need specified in the Scenario. In this case the *Capability* has a name, can be marked as a search objective, can be marked as not requiring fulfilment (standAlone), can be given a start date and an end date of when the capability is needed and has a status of how well it is being fulfilled in a solution with associated colour (Green, Yellow or Red). The accumulation measurement relationship can also be used in this context. This relationship specifics that the capability is an accumulation using the target measurement. This is best explained by example. Consider a Fire Fighting *Capability* that requires a certain gallons of water to be sprayed on a fire. By making the capability an accumulation multiple different *Components* for example multiple fire trucks can contribute to putting out the fire and the values they provide for the accumulation measurement are added together for determining how well the *Capability* is satisfied. The second context is as a need of a *Component* in this case, the *Capability* cannot be marked as a search objective or having a start date and end date but it can be marked as requiring sequential scheduling. Sequential scheduling says that the *Capability* must be fulfilled before work can start on acquiring the *Component* it belongs to. The third context is as a provision of a *Component*. In this case, the Capability has a name and can be marked as only being usable a certain number of times in the goal model by setting a value in the reuse field.

The CATMOS technique generates different solutions by joining *CapabilityProvisions* and *Capabilities* differently (through the satisfies relationship) to attach different goal model fragments together forming different solutions.

Once a *Capability* satisfies another *Capability*, all the values for *Measurements* on the first *Capability* are filled in on the second *Capability*, allowing the *Capabilities* satisfaction to be established.

The script field is used for including measurement annotations. In the current prototype version, Lua [128,129] is used for this purpose. The script is given access to the entire goal model and is able to set an output variable. The output variable overrides the value of the provided value field. This allows the representation of non-trivial relationships in the goal model such as those defined by physics formulas. This both allows global properties across the goal model to be evaluated and for higher level *Capabilities* to take their value from a combination of lower level *Capabilities*.

10.5.11 Component

A *Component* represents a resource within the Defence Lines of Development, e.g. A piece of equipment, a training programme, an organisational structure, etc. Each *Component* has a name, can provide capabilities, can require other capabilities and can costs. The start date and end date fields can be used to define when a *Component* comes into service and goes out of service. A *Component* can also upgrade other *Components* through the *CapabilityUpgrade* class.

10.5.12 Cost

A *Cost* represents a cost for a *Component*. Each *Cost* has a name that defines the type of cost; for example, person-hours can be defined alongside monetary cost. The last three fields are for use for specifying costs over time. The startAfter field can be used to say that a cost is not incurred when the *Component* is acquired but some time before or after it. The repeatDuration field can be used to represent repeating costs such as maintenance costs that repeat every so often. The stopAfter field can be used to specific when a repeating cost stops. For simple one-off costs the last three fields are just set to 0.

10.5.13 Measurement

A *Measurement* can be attached to a *Capability* being used in the context of a provision or a need. In the case of a provision, the *Measurement* has a name and a single quantitative provided value (providedValue) or a multitude of qualitative values attached as *QualitativeValues*. In the case of a need, the *Measurement* has a name, critical values that are required for the *Capability* to be satisfied at all and benchmark values for the *Capability* to be completed satisfied. These can be quantitative or qualitative like before. The script field can contain a Lua [128, 129] script that can override the providedValue field.

10.5.14 QualitativeValue

A *QualitativeValue* simply holds a single string value.

10.5.15 CapabilityUpgrade

A *CapabilityUpgrade* is used for cases where one *Component* changes the *Capabilities* of another *Component*. The name field is the name of the upgrade. The target component field is the name of the *Component* that is affected by the upgrade. The *CapabilityUpgrade* attaches once the goal model is formed to the *Component* it upgrades (if any) via the upgrading link. The exact changes are specified in the *CapabilityChange* class. The three supported operations are adding a new capability, modifying an existing capability or deleting an existing capability. The name field of a *CapabilityChange* class can be “add”, “mod” or “del” and the class is attached to a *Capability* class for specifying the exact changes to make in the “add” or “mod” case. In the “del” case only the *Capabilities’* name is used.

10.5.16 QualitativeValueDictionary

A *QualitativeValueDictionary* is used to indicate that certain *QualitativeValues* are greater or lesser in value than other *QualitativeValues*. For example “Stormy Weather Conditions” may have a higher value than “Calm Weather”. *Capability Provisions* automatically provide every value that is less than them to the *Capability* that they are satisfying.

10.5.17 Summary

In this chapter, we covered the implementation specific details for the current prototype tool. We covered the overall architecture of the tool and some of the concrete details used in implementing the technique described in the previous three chapters. In the next chapter, we evaluate and conclude the work in the thesis then discuss possible avenues for future work.

Chapter 11

Evaluation & Conclusions

The research presented in chapters 7, 8, 9 & 10 will be evaluated in this chapter. The research addresses the three individual research gaps using a new technique called CATMOS.

The ideal way to evaluate the CATMOS technique would be to apply the technique to an actual acquisition of a system of systems. It is infeasible to do this in an EngD because acquiring a system of systems is likely to take millions of pounds and multiple years to complete and the acquisition could extend into decades. Also it would be difficult to determine the impact of the CATMOS technique applied to such a problem. Wicked problems (section 3.3), which large-scale system acquisitions are one of, by their nature are unique [39]. Therefore it is not possible to have a 'control' large-scale system acquisition without the application of CATMOS technique alongside the large-scale system acquisition with the application of the CATMOS technique. This makes it difficult to see the effect of using the CATMOS technique. Due to these factors, an experimental evaluation of the technique isn't feasible, so another method for evaluating the technique is required.

Instead, the research has been evaluated on the two case studies (in sections 8.3 and 9.5) but to tie the evaluation together we present a summative argument here about the validity of the research based on the existing literature, how the technique extends and builds upon the existing literature and the testing of different parts of the research on the case studies.

11.1 Summative Evaluation Argument

In section 1.1.3, it was established that capability based acquisition and goal modelling contain equivalent concepts with capabilities mapping to goals and components mapping to system, people and processes. Goal modelling itself is a well-established technique within the field of early requirement engineering and has been evaluated on a wide variety of case studies and real life projects (20+ real life projects) [158].

The research presented for addressing the first research gap is an extension to existing goal modelling techniques. More specifically, it is the modularisation of goal models into parts, namely the *Goal Tree* and the individual *Components* by adding a layer of abstraction of *CapabilityProvisions* and a layer of abstraction of *Capability* dependencies to the *Components* (Agents). This additional layer of abstraction means that instead of an *Agent* directly satisfying a *Goal*, a *Component* (Agent) has a *CapabilityProvision* that satisfies a *Capability* (Goal).

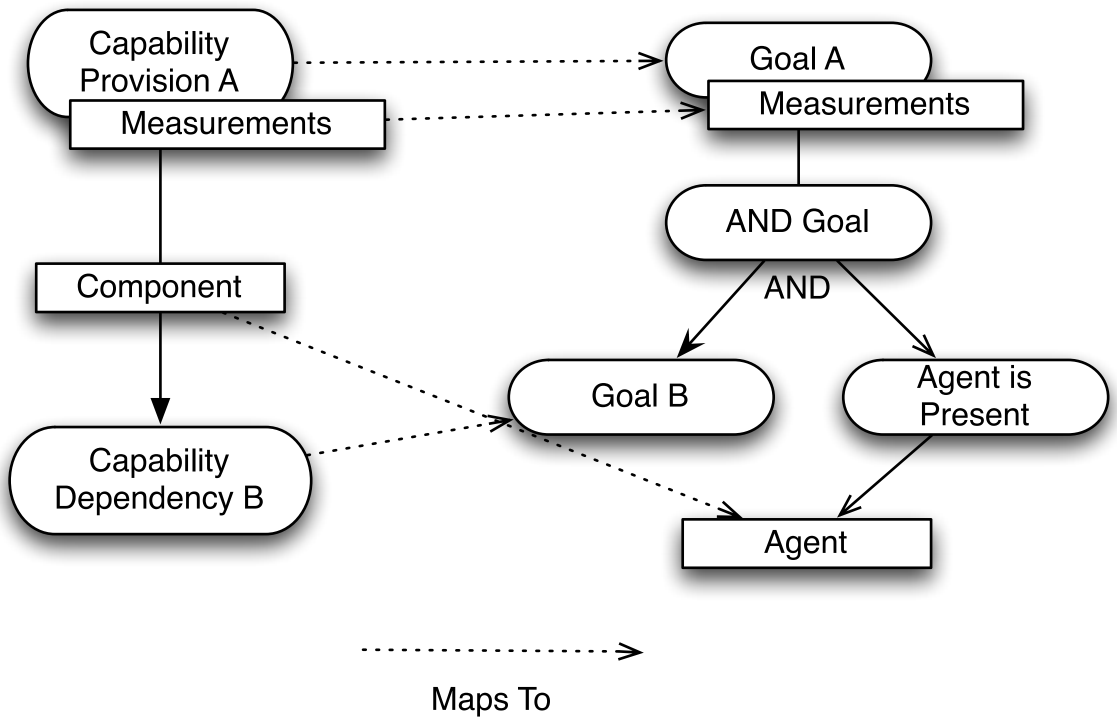


Figure 11.1: Example: Mapping CATMOS Goal Models to Normal Goal Models

When the technique forms a *Completed goal model* from the *Goal Tree* and the *Components*, that *Completed goal model* can be mapped directly to a normal AND/OR tree goal model. This means that the *Completed goal models* provided by the CATMOS technique are equivalent to goal models meaning the results are as valid as in normal goal modelling. The mapping between *Completed goal models* and normal AND/OR tree goal models is trivial.

The mapping is shown in figure 11.1. A *Component* can be converted to an *Agent* by creating a corresponding *Agent* with a new attached goal of the *Agent is present*. This attached goal *Agent is present* needs to be a sub goal of a new *AND Goal* that has as sub goals each of the *Component's* dependencies. The *AND Goal* needs to be a single sub goal of all of the *CapabilityProvisions* rewritten as *Goals*.

Essentially, the work on research gap 1 restructured goal modelling into a format more suitable for performing trade-off analysis. For meeting research gap 1, goal modelling is an objective method for establishing a relationship between the capabilities and the Defence Lines of Development (DLoD). However, it also needs to be shown that the method can handle multiple different possible solutions for going between the capabilities and the DLoD to enable it to support trade-offs. This was done by adding modularisation to goals models by splitting the top-level goal tree and the *Components* apart from each other. This is demonstrated by the Tea Making example shown in chapter 7, the Multi-Objective Next Release Problem case study shown in chapter 8 and the realistic military scenario case study shown in chapter 9.

The second identified research gap is dealing with the multi-objective nature of the problem. This was addressed in chapter 8 using multi-objective search. The multi-objective search takes advantage of the modularisation of goal models done in research gap 1 to find ‘optimal’ ways to construct the goal models from parts. The technique can then produce a Pareto front along with *Completed goal models* for each point on the Pareto front. The Pareto front allows the acquisition decision makers to see the trade-offs that can be made and the corresponding *Completed goal models* allow the acquisition decision makers to see what they need to acquire and how to use what they acquire together to obtain the wanted trade-offs.

The ability of the technique to handle the trade-offs between competing goals and costs as required by research gap 2 has been demonstrated on the Multi-Objective Next Release Problem (MONRP) in section 8.3. The MONRP, though a simpler acquisition problem than Through Life Capability Management, is a suitable candidate for dealing with the trade-off portions of the CATMOS tool. The work on applying the CATMOS technique to the MONRP has been published in [3] and has passed peer review. The question remains whether or not the CATMOS technique is sufficiently feature rich to deal with the Through Life Capability Management problem since it is a more complicated problem than the MONRP. The CATMOS technique was shown to be sufficiently feature rich by the realistic military acquisition scenario case study in section 9.5.

All solutions created using the CATMOS technique can be directly validated and verified by the decision makers. The decision makers create the initial problem as a goal tree and a set of component definitions, and the technique automatically joins them together to produce valid solutions. Though the decision makers are unable to follow how the technique automatically joins together the parts to produce the solutions, the decision makers can visualise the resulting completed solutions, which contain the goal tree defined by the decision makers, some of the component definitions defined by the decision makers and the joins between them created by the technique. The visualisation of the completed solutions allows the decision makers to manually establish whether or not the solution is valid with respect to their problem.

The third identified research gap is dealing with the through life nature of the problem. This was addressed in chapter 9 by adding additional through life annotations to the technique, adding a scheduling step in the genetic algorithm that schedules solutions before they are evaluated, modifying the evaluation to deal with the satisfaction of capabilities over time and producing both Gantt charts and capability over time charts for the acquisition plans. To show that the technique can handle the through life nature of the problem as defined in section 6.3 the technique was applied to a realistic military acquisition scenario in section 9.5 and also published in [1].

The original motivation for this research project came from issues found in TRAiDE [16]. A major motivating issue was from Andrew Daw's integrated management plan [16] that combines together a Gantt chart for an acquisition with capability over time graphs showing how the capability varies as the various acquisition projects come into and leave service. The problem with the integrated management plan was that there was no logic for generating the capability over time graphs from the scheduled acquisition programmes (categorised by the DLoD) on the Gantt chart. As an engineering achievement rather than a research achievement, the CATMOS technique can now provide this logic meaning that a major initial motivating problem for the research project has been addressed.

Problem Size	Run 1	Run 2	Run 3	Mean
1 customer 5 requirements	32s	29s	25s	29s
10 customers 50 requirements	135s	137s	143s	138s
20 customers 100 requirements	378s	401s	403s	394s
30 customers 150 requirements	737s	719s	772s	743s
40 customers 200 requirements	1389s	1349s	1185s	1308s
50 customers 250 requirements	1879s	1948s	2393s	2073s
60 customers 300 requirements	2721s	3778s	2313s	2937s
70 customers 350 requirements	3229s	3233s	2998s	3153s
80 customers 400 requirements	4385s	4587s	4184s	4385s

Table 11.1: Timing Results For The MONRP Problem in the CATMOS prototype tool

11.2 CATMOS Technique - Efficiency

A consideration in evaluating the CATMOS technique is the algorithm complexity; this will give some indication of the scalability and performance of the technique. The complexity results will be presented at the start of section 11.2.2. The Multi-objective Next Release Problem (MONRP) shown in section 8.3 is a suitable test problem to see if the technique can scale up. Some instances shown in the literature of the MONRP such as by Zhang et al [130] use up to 100 customers and 200 requirements. We are therefore going to assess the algorithm complexity for the technique.

11.2.1 Experimental Setup

All the tests will be run on the same hardware, which is a MacBook Air Mid-2011 (Intel Core i5 1.7Ghz, 4GB 1333 MHz DDR3). While this is certainly not the fastest computer available and the tests would run much faster on a modern up-to-date desktop computer the MacBook Air has a well-known hardware configuration and this means the tests can be repeated on the same hardware by a third party.

The population size and generation count of the NSGA-II algorithm greatly effects the overall runtime of the algorithm. We are therefore going to use the same settings as existing work on the MONRP by Zhang et al [130] who uses population size 200 and generation count 50.

Working out the time complexity of the CATMOS technique requires running CATMOS on multiple different problem sizes. The MONRP problem is suitable for this since it can be run at multiple different problem sizes easily using randomly generated data. The timing taken for the CATMOS technique to finish its run with 1 computing core will be measured between 10 to 80 customers, at interval 10, with 5 requirements added for every customer present in the run and random dependencies (acyclical) generated between software features. Since the CATMOS prototype tool will not start with no customer requirements loaded into it, a timing measurement will also be taken at 1 customer with 5 requirements to approximate the 0 timing measurement. The timing results are shown in table 11.1.

11.2.2 Algorithm Complexity Analysis

The problem size against average time taken has been graphed in figure 11.2.1. From equation fitting the CATMOS technique is O(n^2) where n is the problem size. Obviously, increasing the

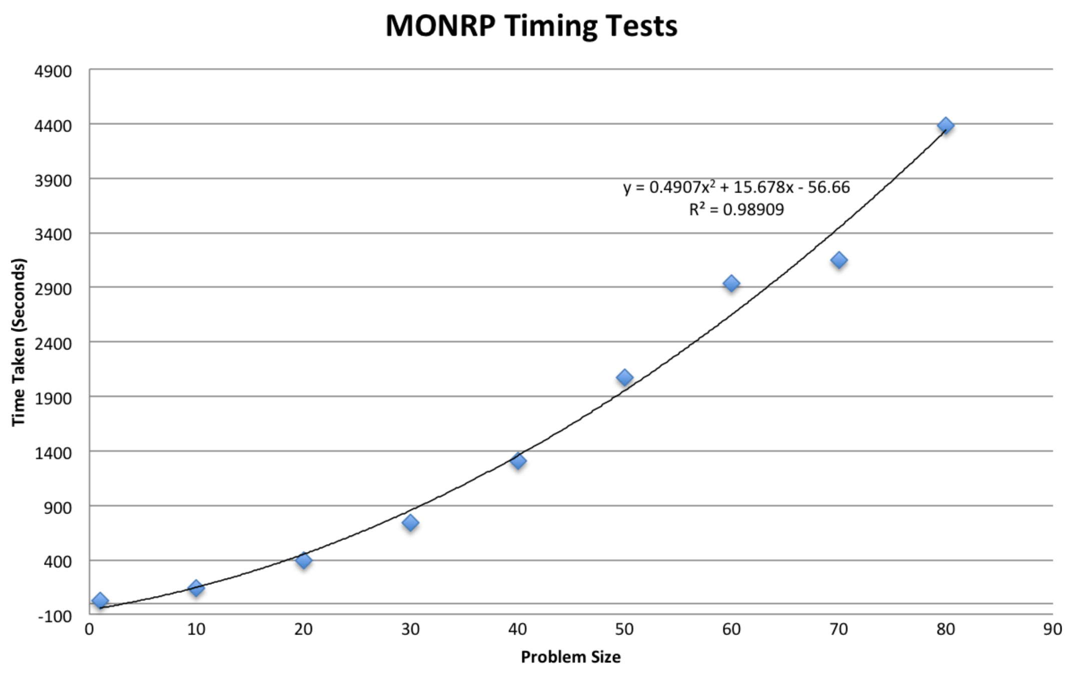


Figure 11.2: MONRP - Problem Size Against Time Taken

population size and generation count will affect the time taken but this is mostly predicable, doubling the population size will take no more than double the time and similarly with the generation count by simple inspection of the genetic algorithm used.

11.2.3 Parallelisation Discussion

One method for dealing with very large problems is parallelisation. The CATMOS prototype tool implements some basic parallelisation. The tool is split into both a GUI and backend logic server. The backend logic server can be run on multiple computing cores on either the same computer or on different computers connected by a network. When the prototype tool is run on multiple cores, one core is given the job of manager that handles the genetic algorithm and the other cores are used for evaluating solutions. The MacBook Air only has 2 cores so isn't suitable for testing parallelisation therefore a computer with Intel Core i7 3770K (3.5GHz with 4 cores and hyper-threading) and 8 GB of RAM has been used. The normal run on the MONRP problem with 100 customers and 200 requirements using this computer takes 23 minutes (exact timings 1382s, 1387s, 1391s, 1388s, 1383s). Using the 4 cores of the Intel Core i7 3770K CPU (using 8 processes) on the same problem only takes 13 minutes (exact timings 837s, 814s, 817s, 817s, 809s). This is a significant improvement over single core usage however it is far from optimal. The prototype has significant overheads in its parallelisation code. The first issue is that the code for retrieving the results from the backend and creating the corresponding models in the front-end runs sequentially and takes approximately 5 minutes due to the large number (100+) of generated results for the MONRP skewing the timings.

With this taken into account the speed is still only approximately 2.5x faster on 4 cores rather than 1 core due to overheads in the prototype tool. A major overhead in the prototype tool is sending the solutions over TCP/IP between the server processes. Another major overhead is that the code currently waits for all the processes to finish processing their current solution before sending out the next set of solutions to be evaluated. This is simply done for ease of implementation and in a non-prototype version it needs to be replaced with a better method.

The prototype tool implementation can be scaled up to handle very large problems, by simply adding more computation power in parallel. Computing power in parallel can be easily and cheaply obtained by using a Cloud computing service such as Amazon EC2 [159].

Implementing a fully working and optimal parallelisation technique for the CATMOS techniques is an area for future work. However, we will briefly discuss how this could be done. Firstly, the simple method currently used by the prototype tool of just sharing out the evaluation step should scale up quite significantly. It will only stop scaling when the single core handling the genetic algorithms population reaches it limit on performing crossovers and mutations on the population set.

To go beyond this, the concept of a spatial genetic algorithm can be used. In a spatial genetic algorithm, there are separate populations in which solutions evolve and a migration step is added where a few solutions each generation are able to migrate to another population [160]. This method can be easily parallelised with only minimal bottlenecks. Each computing core can run the NSGA-II algorithm on its small private population and at the end of a generation it can send a couple of population members randomly to other computing cores working on the same problem allowing the problem to solved in parallel.

In a practical implementation for very large problems, both techniques might be used, with multiple computing cores handling their private populations and farming out the evaluation step to other computing cores. Implementing this is out of the scope of this thesis and is a possible area of future work.

The CATMOS technique is potentially highly scalable. The time taken increases against the problem size at O(n^2), which is more than acceptable. The largest problem found in the literature has 480 randomly generated requirements, which can be implemented in the CATMOS technique as 480 capabilities and 480 components. This run only took 73:05 minutes on a MacBook Air Mid 2011. The realistic military scenario took approximately 70 minutes with population size 200 and generation count 200 on a MacBook Air Mid 2011. This suggests that the runtime is practical for the vast majority of likely problems and if larger problems did emerge than the technique could be scaled up via parallelisation.

11.3 The CATMOS Technique Benefits

The CATMOS technique provides an objective method for bridging the conceptual gap between the Defence Lines of Development and military capabilities and enables the consideration of trade-offs between the Defence Lines of Development, one of the goals for Through Life Capability Management [161]. Performing trade-offs between the Defence Lines of Development was previously considered to be a difficult problem and named the Apples and Wednesdays problem [18]. The CATMOS technique supports the through-life scheduling of acquisitions allowing it to be applied to real world problems. This research addresses very real world problems in Through Life Capability Management and has real world applications.

When applied to the simpler problem of the Multi-objective Next Release Problem, the CATMOS technique allows the consideration of trade-offs in continuous release support, visualisation of solutions, continuous variable requirements and higher overall tool flexibility (see section 8.5 or Burton et al [3]).

11.4 The CATMOS Technique Limitations

As discussed in section 6.4, Zachman [67] proposes that acquisition problems can be split up into a 'why', 'how' and 'what' stage, the CATMOS technique only has the ability to handle trade-offs that handle between the 'how' and the 'what' stage. It cannot deal with trade-offs between the 'why' and 'how' stage, which revolve around how the acquisition problem is to be formulated.

As an illustrative example of the three stage of 'why', 'how' and 'what' consider the 'why' stage to be 'Improving the reading level of children aged 10-12', the 'how' stage to be 'Providing more books to children aged 10-12' and the 'what' stage to be 'Providing a mobile library'. The CATMOS technique is suitable for considering trade-offs between the 'how' and the 'what' stage. This means comparing different solutions in the 'what' stage against the satisfaction of the 'how' stage. For example alternative 'what' stages could be 'Proving more normal libraries', 'Providing children aged 10-12 with tablets loaded with eBooks' or 'Providing downloadable books that can be accessed with existing smartphones'. The CATMOS technique isn't suitable for performing trade-offs between the 'why' and the 'how' stage, such as deciding if the 'how' stage should be either 'Providing more books to children aged 10-12', 'Promoting a new reading program in schools' or 'Providing one on one teaching for struggling students'. This is best handled by an existing collaborative technique such as TRAiDE [50] or Strategy Kinetics [51].

The CATMOS technique also depends on there being a 'how' stage, where the stakeholders create the objectives for the acquisition rather than the stakeholders just deciding what systems to acquire without any regard to what they want the systems to do.

The CATMOS technique cannot find unintended consequences such as those caused by Wicked problems [39]. The best that CATMOS can do is providing visual representations of solutions that can then be checked by human domain experts (see discussion in section 6.4).

The CATMOS technique relies heavily on domain expert knowledge especially for the aggregation of measurements in the goal tree. How the CATMOS technique converts the Measures of Performance (section 2.4) to the Measures of Effectiveness (section 2.5) needs to be specified either as formulas by the domain experts or alternatively be gathered from existing data sets such as simulation results. This type of information is likely to be domain specific and require domain expert knowledge for each domain and so is out of scope for thesis. Some examples of this being done for the military domain are shown in Urwin et al [134] and Venters et al [83].

11.5 Relationship with other research in the existing research fields

We now explain how CATMOS fits into the existing research fields. The technique is motivated by the relatively small field of Capability Based Acquisition. The CATMOS technique is also based on work from the fields of Search Based Software Engineering and Early Requirements

Engineering.

The Multi-objective Next Release Problem (MONRP) is a simpler acquisition problem than the TLCM acquisition problem and therefore MONRP can be handled by our technique as well. This is explored in an early paper by the author [3] and a more updated version of using the CATMOS technique to solve this problem is shown in section 8.3 as a case study. The technique and prototype tool offers features not available in other work on the MONRP and offers extensions to the Multi-objective Next Release Problem; however the tool is far slower. At the time of the early paper, the tool took hours to perform the same task that took dedicated MONRP tools seconds to perform. After some reimplementation work by moving the prototype tool to C++ instead of the interpreted language of the Model-driven Engineering tools the tool currently takes minutes on the same problems, which in exchange for the additional features gained over the dedicated tools is more than reasonable.

From the field of goal modelling, there are two main methods for dealing with evaluating alternative system options [27,162]. Both methods are designed to evaluate two or three alternative system designs against each other as opposed to evaluating tens of thousands as is done in our method.

The more similar of the two methods to CATMOS is work by Letier and Lamsweerde [27]. They build on top of KAOS goal modelling for their work [27]. They add measurement annotations to the KAOS goal tree decomposition, which are similar to the annotations being used by the CATMOS technique on its goal tree decomposition. Their work uses a formal logic (PCTL [163]) where our work uses an embedded scripting language (Lua [128, 129]). The CATMOS technique could similarly use PCTL for the annotations on the goal model rather than Lua but this would be an area of future work. Unlike the CATMOS technique they use a static goal structure and simply replace the satisfying agents with other agents that perform the same task. This allows them to evaluate a small number of similarly structured solutions against each other. Whereas the CATMOS technique uses a dynamic goal structure that allows the evaluation of tens of thousands of differently structured possible solutions.

Letier and Lamsweerde [27] work supports the use of Probability Density Functions (PDFs) for the measurements. These capture probability distributions for measurements rather than single values being used by the CATMOS technique. Adding the ability to use PDFs to the CATMOS technique is an area of future work. This could possibility be done by using the SciLua libraries [164] in conjunction with a server running R [165]. Letier and Lamsweerde [27] do most of the work with PDFs using separate software tools manually rather than having any sort of automated support.

The other main method for performing trade-offs in goal modelling is goal *satisficing* [162] as named in the NFR Framework [162]. This is a qualitative reasoning technique where goals can be considered to be satisfied, satisficable, deniable or denied [162]. Different system designs pass different values to the goals and they are aggregated up using different aggregation rules depending on what is appropriate for the goals involved [162].

11.6 Conclusions

This thesis has researched and contributed to the field of system of systems acquisition, most notably the Through Life Capability Management approach to system of system acquisition. This has been done by bringing in and combining techniques from the fields of goal modelling, search based software engineering and model-driven engineering to address key identified research gaps in Through Life Capability Management relating to trade-off handling.

The thesis research hypothesis is:

> The high-level trade-off decision space during system of systems acquisition can be effectively explored using a technique that generates an approximation of the Pareto front of the fulfilment of the various organisational objectives against the resources used in the context of acquiring systems for the system of systems.

The presented CATMOS technique in chapters 7, 8, 9 & 10 has succeeded in meeting the research hypothesis and has been shown to work on multiple case studies including a realistic military scenario and on the Multi-Objective Next Release Problem. How the research fits into the research field of acquisition and into the research fields that have drawn upon by the work has been discussed in sections 6.4 & 11.5.

The thesis has made contributions to the state of the art in the research fields of Through Life Capability Management, goal modelling and search based software engineering. These contributions have been detailed at the start of the thesis in section 1.6. Some of these contributions have been published with two conference papers [1, 3] covering the CATMOS technique and the second and third case study. The thesis is a successful example of cross-disciplinary research that combines multiple academic research fields to solve a problem found in industry.

As discussed in the previous sections, the research has done a great deal to address the identified research gaps of the thesis and has covered a large amount of the original purposed scope for the project. The research also ties back into one of the original motivating engineering problems of providing the necessary underlying logic to support the Integrated Management Plan [16] used in TRAiDE [16, 50], which means that the thesis has not drifted off its original scope. The thesis has not only added to the state of the art in research but also addressed engineering issues as part of a doctorate of engineering.

11.7 Future Work

Though the CATMOS technique has substantially addressed the stated research gaps there is still more work and avenues for further research.

11.7.1 Graphical Improvements

An improvement that could be made is improving the prototype tool's graphical user interface. At the moment the goal models produced by the tool are rather basic as shown in chapter 8 figure 8.6. A much better graphical notation was devised in this thesis as shown in chapter 7.2 figure 7.3. This would make the results of the prototype tool more readable. The reason that this has not been done is because it is an extensive implementation task that has little academic value in itself. After this task, there is further work to be done around testing and improving the usability of the tool itself with end users.

11.7.2 Case studies

The CATMOS technique has been applied to case studies such as the Multi-objective Next Release Problem (section 8.3) and a realistic military case study (section 9.5). A future work direction is performing case studies using real acquisitions and reviewing the results.

11.7.3 Sensitivity Analysis

There is a question of how robust are the acquisition plans to modelling uncertainly in the problem. Work on this has already begun by Williams et al [4]. Continuing and extending this work is a clear possible future research direction for the CATMOS technique.

11.7.4 Probability Density Functions

The measurements in the CATMOS technique are precise values, where in the real world there is uncertainly over the exact values of measurements and instead of asking if an objective is fulfilled, the question is how likely is it that the objective will be fulfilled. There is research that extends goal modelling in this way by Letier and Lamsweerde [27] by using probability density functions. Extending the CATMOS technique to include this is a possible future research direction.

11.7.5 Scalability

As discussed in section 11.2.3, there are extensive improvements that could be made to the prototype tools parallelisation with the prototype tool implementation using a very basic and inefficient approach to parallelisation. A future research direction is in providing better support for parallelisation of the technique.

Appendix A

Xtext Grammar Definition

The formal Xtext [133] grammar definition for the CATMOS textual domain specific language is shown below:

```
grammar org.xtext.Scenario with org.eclipse.xtext.common.Terminals

import "ScenarioModel"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore

Scenario returns Scenario:
        {Scenario}
                (OurObjects+=OurObject ( OurObjects+=OurObject)* )?
        ;


OurObject returns OurObject:
        OurObject_Impl | Capability_Impl | CapabilityProvision |
    QualitativeValueDictionary | Component | RunConfiguration_Impl |
    FindTradeOffs | ThroughLifePlanning;


Capability returns Capability:
        Capability_Impl | CapabilityProvision;

Cost returns Cost:
        Cost_Impl;

ComponentU returns ComponentU:
        ComponentU_Impl | ExistingComponent | AcquirableComponent;

Constraint returns Constraint:
        Constraint_Impl | DesireHigh | DesireLow | Budget;


OurObject_Impl returns OurObject:
        {OurObject}
        'OurObject'
        ;

```

```
Capability_Impl returns Capability:
        {Capability}
        (standAlone?='standAlone')?
        (searchObjective?='searchObjective')?
        (sequentialScheduling?='sequentialScheduling')?
        'Capability'
        name=EString
        '{'
                ('reuse' reuse=EIntegerObject)?
                ('accumulation' accumulation=[Measurement|EString])?
                ('startDate' startDate=EString)?
                ('endDate' endDate=EString)?
                ( measurements+=Measurement ( measurements+=Measurement)* )?
                ('decomposes' '(' decomposes+=[Capability|EString] ( ","
        decomposes+=[Capability|EString])* ')' )?
                ('satisfiedBy' '(' satisfiedBy+=[Capability|EString] ( ","
        satisfiedBy+=[Capability|EString])* ')' )?
        '}';

CapabilityProvision returns CapabilityProvision:
        {CapabilityProvision}
        (searchObjective?='searchObjective')?
        'CapabilityProvision'
        name=EString
        '{'
                ('reuse' reuse=EIntegerObject)?
                ('startDate' startDate=EString)?
                ('endDate' endDate=EString)?
                ( measurements+=Measurement ( measurements+=Measurement)* )?
        '}';

QualitativeValueDictionary returns QualitativeValueDictionary:
        'Value'
        name=EString
        ('<' lessThan=[QualitativeValueDictionary|EString])?
        ('>' greaterThan=[QualitativeValueDictionary|EString])?
        ;

Component returns Component:
        {Component}
        'Component'
        name=EString
        '{'
                ( provides+=CapabilityProvision ( provides+= CapabilityProvision)* )?
                ( upgrades+=CapabilityUpgrade ( upgrades+=CapabilityUpgrade )* )?
                ( requires+=Capability ( requires+=Capability)* )?
                ( costs+=Cost ( costs+=Cost)* )?
        '}';
```

```

RunConfiguration_Impl returns RunConfiguration:
        {RunConfiguration}
        'RunConfiguration'
        name=EString
        '{'
                ('popSize' popSize=EIntegerObject)?
                ('genCount' genCount=EIntegerObject)?
        '}';

FindTradeOffs returns FindTradeOffs:
        {FindTradeOffs}
        'FindTradeOffs'
        name=EString
        '{'
                ('popSize' popSize=EIntegerObject)?
                ('genCount' genCount=EIntegerObject)?
                (components+=ComponentU ( components+=ComponentU)* )?
                (constraints+=Constraint ( constraints+=Constraint)* )?
        '}';

ThroughLifePlanning returns ThroughLifePlanning:
        {ThroughLifePlanning}
        'ThroughLifePlanning'
        name=EString
        '{'
                ('popSize' popSize=EIntegerObject)?
                ('genCount' genCount=EIntegerObject)?
                ('startDate' startDate=EString)?
                ('endDate' endDate=EString)?
                (components+=ComponentU ( components+=ComponentU)* )?
                (constraints+=Constraint ( constraints+=Constraint)* )?
        '}';

EString returns ecore::EString:
        STRING | ID;

EBooleanObject returns ecore::EBooleanObject:
        'true' | 'false';

EIntegerObject returns ecore::EIntegerObject:
        '-'? INT;

Measurement returns Measurement:
        {Measurement}
        'Measurement'
        name=EString
        '{'
                ('criticalValue' criticalValue=EFloatOurObject)?
                ('benchmarkValue' benchmarkValue=EFloatOurObject)?
                ('providedValue' providedValue=EFloatOurObject)?
                ('criticalValues' '(' criticalValues+=QualitativeValue ( ",
                    "
        criticalValues+=QualitativeValue)* ')' )?
                ('benchmarkValues' '(' benchmarkValues+=QualitativeValue (
                    ","
        benchmarkValues+=QualitativeValue)* ')' )?
                ('providedValues' '(' providedValues+=QualitativeValue ( ",
                    "
        providedValues+=QualitativeValue)* ')' )?
                ('script' script=EString)?
        '}';

EFloatOurObject returns ecore::EFloatObject:
        '-'? INT? '.' INT (('E'|'e') '-'? INT)?;

QualitativeValue returns QualitativeValue:
        {QualitativeValue}
        name=EString;

CapabilityUpgrade returns CapabilityUpgrade:
        {CapabilityUpgrade}
        'CapabilityUpgrade'
        name=EString
        '{'
                ('targetComponent' targetComponent=EString)?
                ('upgrading' upgrading=[Component|EString])?
                (capabilityChanges+=CapabilityChange (
        capabilityChanges+=CapabilityChange)* )?
        '}';

Cost_Impl returns Cost:
        {Cost}
        'Cost'
        name=EString
                (amount=EFloatOurObject)?
        ('{'
        ('startAfter' startAfter=EString)?
        ('repeatDuration' repeatDuration=EString)?
        ('stopAfter' stopAfter=EString)?
        '}')?
        ;

CapabilityChange returns CapabilityChange:
        {CapabilityChange}
        'CapabilityChange'
        name=EString
        '{'
                ( capabilities+=CapabilityProvision ( capabilities+=
                    CapabilityProvision)* )?
        '}';


```

```
ComponentU_Impl returns ComponentU:
        {ComponentU}
        'Component'
        name=EString
                (quantity=EIntegerObject)?
        ;

ExistingComponent returns ExistingComponent:
        {ExistingComponent}
        'ExistingComponent'
        name=EString
                (quantity=EIntegerObject)?
                ('startDate' startDate=EString)?
                ('endDate' endDate=EString)?
        ;

AcquirableComponent returns AcquirableComponent:
        {AcquirableComponent}
        'AcquirableComponent'
        name=EString
                (quantity=EIntegerObject)?
                ('acquisitionTime' acquisitionTime=EString)?
                ('lifeSpan' lifeSpan=EString)?
        ;


Constraint_Impl returns Constraint:
        {Constraint}
        'Constraint'
        ;

DesireHigh returns DesireHigh:
        {DesireHigh}
        'DesireHigh'
        name=EString;

DesireLow returns DesireLow:
        {DesireLow}
        'DesireLow'
        name=EString;

Budget returns Budget:
        {Budget}
        'Budget'
        name=EString
        '{'
                ('amount' amount=EFloatOurObject)?
                ('startDate' startDate=EString)?
                ('repeatDuration' repeatDuration=EString)?
                ('endDate' endDate=EString)?
        '}';

```

```
EDoubleObject returns ecore::EDoubleObject:
        '-'? INT? '.' INT (('E'|'e') '-'? INT)?;
```

Appendix B

Military Case Study - Textual DSL Input

```
//Military Scenario
ThroughLifePlanning UAV { popSize 200 genCount 200
        startDate "01/01/2014" endDate "31/12/2016"

        ExistingComponent "Mastiff Fleet" 3 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Vector Fleet" 2 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Troop Regiment" 5 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "L118 Light Gun Fleet" 5 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Mobile Artillery Battlefield Radar" 5 startDate "01/01/2014" endDate "01/08/2014"
        ExistingComponent "Land Rover 101 FC Fleet" 5 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Buffalo Fleet" 1 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Engineering Team with Equipment" 1 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Oshkosh Wheeled Tanker Fuel Fleet" 2 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Oshkosh Wheeled Tanker Water Fleet" 3 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Leyland DROPS Fleet" 4 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Global Hawk" 1 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Reacher Fleet" 1 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Mowag Duro III Fleet" 1 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "Bowman" 1 startDate "01/01/2014" endDate "31/12/2016"
        ExistingComponent "L118 Existing Service Contract" 1 startDate "01/01/2014" endDate "01/01/2015"

        AcquirableComponent "Mastiff Fleet" 5 acquisitionTime "2 months" lifeSpan "15 years"
        AcquirableComponent "Vector Fleet" 5 acquisitionTime "2 months" lifeSpan "15 years"
        AcquirableComponent "L118 Light Gun Fleet New" 5 acquisitionTime "6 months" lifeSpan "15 years"
        AcquirableComponent "Land Rover 101 FC Fleet" 5 acquisitionTime "6 months" lifeSpan "15 years"
        AcquirableComponent "Global Hawk" 5 acquisitionTime "8 months" lifeSpan "15 years"
        AcquirableComponent "MQ-9 Reaper" 5 acquisitionTime "10 months" lifeSpan "15 years"
        AcquirableComponent "SAS Training" 5 acquisitionTime "9 months" lifeSpan "25 years"
        AcquirableComponent "Mobile Artillery Battlefield Radar Foreign" 5 acquisitionTime "6 months" lifeSpan "15 years"
        AcquirableComponent "L118 New Service Contract" 1 acquisitionTime "0 months" lifeSpan "10 years"

        Budget "Money" { amount 50.0 startDate "05/03/2014" }
        Budget "Money" { amount 45.0 startDate "01/09/2014" }
        Budget "Money" { amount 45.0 startDate "01/12/2014" }
        Budget "Money" { amount 45.0 startDate "01/06/2016" }

}

//searchObjective
Capability "Overall Score"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Overall Score" { criticalValue 0.0 benchmarkValue 1.0 script 'output = OverallScore()' }
                decomposes ("Route Clearance","Hold Forward Base", "Prevent Enemy Crossings")
}

searchObjective
Capability "Route Clearance"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Route Clearance"
        { criticalValue 0.0 benchmarkValue 1.0 script "output = RouteClearance()" }
        decomposes ("Ground Fire Power", "Hard Target Removal", "Mine Clearance Solution", "Command and Control")
}

Capability "Ground Fire Power"
{
        accumulation "Ground Fire Power"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Ground Fire Power" { criticalValue 0.0 benchmarkValue
            7000.0 }
}

Capability "Hard Target Removal"
{
        accumulation "Hard Target Removal"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Hard Target Removal" { criticalValue 0.0
            benchmarkValue 100.0 }
}

Capability "Mine Clearance Solution"
{
                startDate "01/01/2014" endDate "31/12/2016"
                Measurement "Chance of Death" { criticalValue 10.0
                    benchmarkValue 0.0 }
                Measurement "Mine Clearance Per Day" { criticalValue 10.0
                    benchmarkValue 40.0 }
}

searchObjective
Capability "Hold Forward Base"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Hold Forward Base"
        { criticalValue 0.0 benchmarkValue 1.0 script "output =
            HoldForwardBase()" }
        decomposes ("Establish Forward Base", "Supply Forward Base")
}

Capability "Establish Forward Base"
{
        startDate "01/01/2014" endDate "31/12/2016"
}

Capability "Supply Forward Base"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Supply Forward Base"
        { criticalValue 0.0 benchmarkValue 1.0 script "output =
            SupplyForwardBase()" }
        decomposes ("Supply Water", "Supply Fuel", "Supply Goods")
}

Capability "Supply Water"
{
        accumulation "Water Liters"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Water Liters" { criticalValue 0.0 benchmarkValue
            50000.0 }
}

Capability "Supply Fuel"
{
        accumulation "Fuel Liters"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Fuel Liters" { criticalValue 0.0 benchmarkValue
            70000.0 }
}

Capability "Supply Goods" //Includes Ammo
{
        accumulation "Goods Kilograms"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Goods Kilograms" {  criticalValue 0.0 benchmarkValue
            75000.0 }
}

searchObjective
Capability "Prevent Enemy Crossings"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Prevent Enemy Crossings"
        { criticalValue 0.75 benchmarkValue 1.0 script "output =
            PreventEnemyCrossings()" }
        decomposes ("Detect Enemy Crossings", "Stopping Enemy Crossings")
}

standAlone
Capability "Detect Enemy Crossings"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Detect Enemy Crossings" { criticalValue 0.0
            benchmarkValue 1.0
                script 'output = DetectEnemyCrossings()'
        }
}

Capability "Stopping Enemy Crossings"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Enemy Crossings Stopped Percentage" { criticalValue
            0.75 benchmarkValue 1.0
                script 'output = StopEnemyCrossings()'
        }
        decomposes ("Ground Fire Power", "Command and Control", "
            Surveillance Moving Targets")
}

Capability "Command and Control"
```

```
{
        accumulation "Command and Control Infrastructure"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Command and Control Infrastructure" { criticalValue
            0.0 benchmarkValue 100.0 }
}

Capability "Surveillance"
{
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Surveillance"
        { criticalValue 0.0 benchmarkValue 1.0 script "output =
            Surveillance()" }
        decomposes ("Surveillance Static Targets", "Surveillance Moving
            Targets")
}

Capability "Surveillance Static Targets"
{
        accumulation "Surveillance Static Targets"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Surveillance Static Targets" { criticalValue 0.0
            benchmarkValue 100.0 }
}

Capability "Surveillance Moving Targets"
{
        accumulation "Surveillance Moving Targets"
        startDate "01/01/2014" endDate "31/12/2016"
        Measurement "Surveillance Moving Targets" { criticalValue 0.0
            benchmarkValue 100.0 }
}

//Existing System
Component "Mastiff Fleet" //100 Mastiff's
{
        //12mm Heavy Machine Gun
        CapabilityProvision "Ground Fire Power" { reuse 1 Measurement "
            Ground Fire Power" { providedValue 100.0 } }

        //Carries
        CapabilityProvision "Troop Transport" { reuse 3 }

        Cost Money 25.0 // £250,000 each
}

Component "Vector Fleet" //100 Vectors
{
        //2 Small Machine Guns
        CapabilityProvision "Ground Fire Power" { reuse 1 Measurement "
            Ground Fire Power" { providedValue 60.0 } }

        //Carries
        CapabilityProvision "Troop Transport" { reuse 1 }

        Cost Money 15.0 // £150 ,000 each
}

Component "Troop Regiment"
{
        CapabilityProvision "Ground Fire Power" { reuse 1 Measurement "
            Ground Fire Power" { providedValue 1000.0 } }
        CapabilityProvision "Surveillance Static Targets" { reuse 1
            Measurement "Surveillance Static Targets" { providedValue 3.0
            }}
        CapabilityProvision "Surveillance Moving Targets" { reuse 1
            Measurement "Surveillance Moving Targets" { providedValue 3.0
            }}
        Capability "Troop Transport" {}
}

//Hard Target Removal - Existing
Component "L118 Light Gun Fleet" //10 Artillery Pieces
{
        CapabilityProvision "Hard Target Removal" { reuse 1 Measurement "
            Hard Target Removal" { script "output = LightGun()" } }
        Capability "Light Gun Tow" {} Capability "L118 Service Contract" {}
        Cost Money 5.0 // £500 ,000 each
}

Component "L118 Existing Service Contract"
{
        CapabilityProvision "L118 Service Contract" {}
}

Component "L118 New Service Contract" // £15 million maintenance
{
        CapabilityProvision "L118 Service Contract" {}
        Cost Money 15.0
}

Component "L118 Light Gun Fleet New" //10 Artillery Pieces
{
        CapabilityProvision "Hard Target Removal" { reuse 1 Measurement "
            Hard Target Removal" { script "output = LightGun()" } }
        Capability "Light Gun Tow" {}
        Cost Money 5.0 // £500 ,000 each
}

Component "Mobile Artillery Battlefield Radar"
{
        CapabilityProvision "Surveillance Moving Targets" { reuse 1
            Measurement "Surveillance Moving Targets" { providedValue 5.0
            }}
        Capability "Radar Mount" {}
}

Component "Mobile Artillery Battlefield Radar Foreign"
{
        CapabilityProvision "Surveillance Moving Targets" { reuse 1
            Measurement "Surveillance Moving Targets" { providedValue 5.0
            }}
        Capability "Radar Mount" {}
        Cost Money 15.0
}

Component "Land Rover 101 FC Fleet" //Tows The Artillery Piece Into
    Position
{
        CapabilityProvision "Light Gun Tow" { reuse 1 }
        CapabilityProvision "Radar Mount" {}
        Cost Money 12.0 //£120,000 each
}

//Mine Clearance
Component "Buffalo Fleet"
{
        CapabilityProvision "Mine Clearance Solution" {
                Measurement "Chance of Death" { providedValue 5.0 }
                Measurement "Mine Clearance Per Day" { providedValue 50.0 }
                    }
}

//Establish Forward Base
Component "Engineering Team with Equipment"
{
        CapabilityProvision "Establish Forward Base" {
                Measurement "Base Quality" { providedValue 0.0 script "
                    output = BaseQuality()"}
        }
        CapabilityProvision "Command and Control" { reuse 1
                Measurement "Command and Control Infrastructure" {
                    providedValue 40.0 } }

        Cost Money 5.0
}

//Supply Forward Base
Component "Oshkosh Wheeled Tanker Fuel Fleet" //10
{
        CapabilityProvision "Supply Fuel" { reuse 1 Measurement "Fuel
            Liters" { providedValue 20000.0 } }
        Cost Money 3.0
}

Component "Oshkosh Wheeled Tanker Water Fleet" //10
```

```
{
        CapabilityProvision "Supply Water" { reuse 1 Measurement "Water
            Liters" { providedValue 18000.0 } }
        Cost Money 3.0
}

Component "Leyland DROPS Fleet" //10
{
        CapabilityProvision "Supply Goods" { reuse 1 Measurement "Goods
            Kilograms" { providedValue 15000.0 } }
        Cost Money 3.0
}

Component "Global Hawk"
{
        CapabilityProvision "Surveillance Static Targets" { reuse 1
            Measurement "Surveillance Static Targets" { providedValue 45.0
            }}
        CapabilityProvision "Surveillance Moving Targets" { reuse 1
            Measurement "Surveillance Moving Targets" { providedValue 25.0
            }}
        Cost Money 125.0 // £125 million
}

Component "Reacher Fleet"
{
        CapabilityProvision "Command and Control" { reuse 1
                Measurement "Command and Control Infrastructure" {
                    providedValue 20.0 } }
}

Component "Mowag Duro III Fleet"
{
        CapabilityProvision ReacherMount { reuse 1 }
}

Component "Bowman"
{
        CapabilityProvision "Command and Control" { reuse 1
                Measurement "Command and Control Infrastructure" {
                    providedValue 40.0 } }
}

Component "MQ-9 Reaper"
{
        CapabilityProvision "Hard Target Removal" { reuse 1 Measurement "
            Hard Target Removal" { providedValue 30.0 } }
        CapabilityProvision "Surveillance Static Targets" { reuse 1
            Measurement "Surveillance Static Targets" { providedValue 10.0
            }}
        CapabilityProvision "Surveillance Moving Targets" { reuse 1
            Measurement "Surveillance Moving Targets" { providedValue 5.0
            }}
        Cost Money 10.5 // £10.5 million
}

Component "SAS Training"
{
        CapabilityUpgrade "Better Training"
        {
                targetComponent "Troop Regiment"
                CapabilityChange "mod"
                {
                        CapabilityProvision "Ground Fire Power" { reuse 1
                            Measurement "Ground Fire Power" { providedValue
                             1500.0 } }
                        CapabilityProvision "Surveillance Static Targets" {
                             reuse 1 Measurement "Surveillance Static
                            Targets" { providedValue 5.0 }}
                        CapabilityProvision "Surveillance Moving Targets" {
                             reuse 1 Measurement "Surveillance Moving
                            Targets" { providedValue 5.0 }}
                }
        }
        Cost Money 8.0 // £8 Million
}
```

B.1 Military Acquisition Scenario - Lua Script

The Lua script that defines the functions used by the textual DSL input above is shown below:

```
function RouteClearance()
        return 0.25 * getCapability("Ground Fire Power") + 0.25 *
            getCapability("Hard Target Removal") + 0.25 * getCapability("
            Mine Clearance Solution") + 0.25 * getCapability("Command and
            Control")
end

function LightGun()
        return 5.0 * getCapability("Surveillance Static Targets") + 5.0 *
            getCapability("Surveillance Moving Targets")
end

function MineClearance()
        return 0.8 * getCapability("Mine Clearance Chance of Death") + 0.2
            * getCapability("Mine Clearance Per Day")
end

function SupplyForwardBase()
        return 0.34 * getCapability("Supply Water") + 0.33 * getCapability(
            "Supply Fuel") + 0.33 * getCapability("Supply Goods");
end

function Surveillance()
```

```
        return 0.5 * getCapability("Surveillance Static Targets") + 0.5 *
            getCapability("Surveillance Moving Targets");
end

function BaseQuality()
        if getCapability("Supply Goods") > 0.3 then
                return 1.0
        end
        return getCapability("Supply Goods") * 0.3;
end

function DetectEnemyCrossings()
        return getCapability("Surveillance Moving Targets");
end

function StopEnemyCrossings()
        return math.min(getCapability("Ground Fire Power")*7.0,
            getCapability("Command and Control"));
end

function HoldForwardBase()
        return 0.5 * getCapability("Establish Forward Base") + 0.5 *
            getCapability("Supply Forward Base");
end

function PreventEnemyCrossings()
        return math.min(DetectEnemyCrossings(), StopEnemyCrossings())
end

function OverallScore()
        return 0.333 * getCapability("Route Clearance") + 0.333 *
            getCapability("Hold Forward Base") + 0.334 * getCapability("
            Stopping Enemy Crossings")
end
```

Glossary

Word	Definition
Agent	A system, person or process that can satisfy a goal. [21]
Capability	An acquisition objective. Equivalent concept to a goal but using Through Life Capability Management terminology. [7]
CATMOS	Capability Acquisition Technique with Multi-objective Search - The main work provided by this thesis.
Component	An acquirable thing from one of the Defence Lines of Development. Similar concept to agent but with wider implicit scope.
Defence Lines Of Development (DLoD)	The eight types of things that work together to produce military capability. Training, Equipment, Personnel, Information, Doctrine and Concepts, Organisational Structure, Infrastructure and Logistics. [17]
Goal	An acquisition objective. [21]
Large Scale Complex IT System (LSCITS)	A very large scale system that has been created from the combination of numerous other systems working together to achieve a common goal. For more in-depth information see the Ultra Large Scale Systems Report [45].
Meta-heuristic search	A method for finding approximately correct answers for problems where the calculation of exact answers is computationally infeasible. [111]
Multi-objective search	An extension to meta-heuristic search for finding a Pareto front of results rather than a single good result. [112]

Word	Definition
Pareto Front	A common technique used in the field of economics named after its inventor Vilfredo Pareto (1848 - 1923). It focuses on the concept of there being multiple objectives and maximising objectives without lowering the value of other objectives whilst doing so.
System of systems (SoS)	Same as Large Scale Complex IT Systems.
Through Life Capability Management (TLCM)	The management of acquisition by using the abstraction notion of capability and maintaining it through life. [7]
Through Life Management (TLM)	The management of acquisition by periodically updating equipment. [7]

Bibliography

[1] F. Burton, R. Paige, S. Poulding, and S. Smith, "System of systems acquisition trade-offs," in *2014 Conference on Systems Engineering Research (CSER 2014)*, (Los Angeles, USA), Mar. 2014.

[2] F. R. Burton and S. Poulding, "Complementing metaheuristic search with higher abstraction techniques," *Proc. 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, 2013.

[3] F. R. Burton, R. F. Paige, L. M. Rose, D. S. Kolovos, S. Poulding, and S. Smith, "Solving acquisition problems using model-driven engineering," in *Modelling Foundations and Applications*, pp. 428–443, Springer, 2012.

[4] J. R. Williams, F. R. Burton, R. F. Paige, and F. A. Polack, "Sensitivity analysis in model-driven engineering," in *Model Driven Engineering Languages and Systems*, pp. 743–758, Springer, 2012.

[5] R. F. Paige, P. J. Brooke, X. Ge, C. D. Power, F. R. Burton, and S. Poulding, "Revealing complexity through domain-specific modelling and analysis," in *Large-Scale Complex IT Systems. Development, Operation and Management*, pp. 251–265, Springer, 2012.

[6] D. Cliff, J. Keen, M. Kwiatkowska, J. McDermid, and I. Sommerville, "Large scale complex it systems (LSCITS) research programme. research proposal to the uk engineering and physical sciences research council (2006)."

[7] T. McKane, "Enabling acquisition change - an examination of the Ministry of Defence's ability to undertake Through Life Capability Management," tech. rep., Ministry of Defence, June 2006.

[8] Ministry of Defence, "Acquisition operating framework - through life capability management." `http://www.aof.mod.uk/aofcontent/tactical/tlcm/content/introductiontotlcm.htm` Version 1.1.6, July 2010.

[9] Command and Battlespace Management Board, "Network enabled capability - JSP 777 edition 1," tech. rep., Ministry of Defense, January 2005.

[10] S. Butler, "Network Enabled Capability: Alive and Well in the MoD," tech. rep., RUSI C4ISTAR Conference, October 2008.

[11] E. Quintana, "Is NEC Dead? An Analysis of Industry's Perspective on the UK's NEC Programme," tech. rep., Military Sciences Department, Royal United Services Institute for Defence and Stragetic Studies, 2007.

[12] Y. Yue and M. Henshaw, "A Holistic View of UK Military Capability Development," *Defense & Security Analysis*, vol. 25, pp. 53 – 67, 2009.

[13] J. Bourn, "Through-life management," tech. rep., National Audit Office, May 2003.

[14] House of Commons Committee of Public Accounts, *Building an Air Manoeuvre Capability: The Introduction of the Apache Helicopter (House of Commons Papers)*. Stationery Office Books, 2002.

[15] J. Reid, A. Johnson, D. Browne, Lord Drayson, and A. Michael, "Defence industial strategy - Defence white paper," tech. rep., Ministry of Defence, December 2005.

[16] A. J. Daw, "On the use of synthetic environments for the through life delivery of capability," in *Analytical Support to Defence Transformation*, pp. Meeting Proceedings RTO–MP–SAS–055, Paper 9, 2005.

[17] Ministry of Defence, "Defence lines of development." `http://www.aof.mod.uk/aofcontent/strategic/guide/sg_dlod.htm` [Last accessed January 2010].

[18] B. Barton and D. Whittington, "Informing high level trades - some novel techniques," in *13th ICCRTS*, March 2008.

[19] B. J. Brittain, "Through-life capability management one year on," *Royal United Services Institute Defence Systems*, pp. 30–32, June 2008.

[20] D. Cliff, J. Keen, M. Kwiatkowska, J. McDermid, and I. Sommerville, "LSCITS initiative overview." `http://lscits.cs.bris.ac.uk/overview.html` [Last Accessed August 2014].

[21] A. Lamsweerde, A. Dardenne, B. Delcourt, and F. Dubisy, "The KAOS Project: Knowledge acquisition in automated specifications of software, proceeding AAAI Spring Symposium series, Track: Design of composite systems," 1991.

[22] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," *Requirements Engineering, IEEE International Conference on*, vol. 0, pp. 226–236, 1997.

[23] MODAF Group, Ministry of Defence, "MOD architecture framework." `http://www.mod.uk/DefenceInternet/AboutDefence/WhatWeDo/InformationManagement/MODAF/` [Last Accessed January 2013].

[24] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pp. 249–262, IEEE, 2001.

[25] A. Wyer and D. Long, "Modelling capability in support of transition management," in *SimTecT 2006, Simulation Conference*, 2006.

[26] F. Bernier, M. Couture, G. Dussault, C. Lalancette, F. Lemieux, M. Lizotte, M. Mokhtari, and S. Lam, "Capdem - toward a capability engineering process," tech. rep., Defence R & D Canada, September 2005.

[27] E. Letier and A. Van Lamsweerde, "Reasoning about partial goal satisfaction for requirements and design engineering," in *ACM SIGSOFT Software Engineering Notes*, vol. 29, pp. 53–62, ACM, 2004.

[28] Y. Zhang, A. Finkelstein, and M. Harman, "Search based requirements optimisation: Existing work and challenges," *Requirements Engineering: Foundation for Software Quality*, pp. 88–94, 2008.

[29] Oxford University Press, "Oxford dictionaries." `http://oxforddictionaries.com/view/entry/m_en_gb0121230` [Last Accessed August 2014], April 2014.

[30] Ministry of Defence, "Governing policy (gp) 1.1 - logistics readiness and sustainability requirements." `http://www.aof.mod.uk/aofcontent/tactical/sse/content/ksa1/gp101.htm` [Last Accessed August 2010].

[31] D. J. Hurley, "Defence capability development manual," tech. rep., Australian Department of Defence, 2006.

[32] S. A. Fry, "Joint publication 1-02 - Department of Defence dictionary of military and associated terms," tech. rep., Department of Defence, July 2010 (Last Amended).

[33] N. Sproles, "Coming to grips with measures of effectiveness," *Systems Engineering*, vol. 3, no. 1, pp. 50–58, 2000.

[34] A. Korzybski, "A non-aristotelian system and its necessity for rigour in mathematics and physics," *Science and Sanity*, pp. 747 – 761, 1933.

[35] R. G. Sargent, "Verification and validation of simulation models," in *WSC '05: Proceedings of the 37th conference on Winter simulation*, pp. 130–143, Winter Simulation Conference, 2005.

[36] B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003.

[37] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE Software*, vol. 20, pp. 42–45, 2003.

[38] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, pp. 25–31, 2006.

[39] H. W. J. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Policy Sciences*, vol. 4, pp. 155–169, June 1973.

[40] B. Newsome, "Don't get your mass kicked: A management theory of military capability," *Defense & Security Analysis*, vol. 19, pp. 131 – 148, 2003.

[41] C. von Clausewitz, *On War*. Princeton University Press, 1989.

[42] A. J. Daw, "New process and structure thinking for capability development," in *9th International Command and Control Research and Technology Symposium*, September 2004.

[43] Ministry of Defence, "Acquisition operating framework - introduction to urgent operational requirements (uor)." `http://www.aof.mod.uk/aofcontent/tactical/tlcm/content/uor/uor_intro.htm` [Last Accessed September 2010].

[44] A. Fox, D. Cliff, T. Hoare, M. Fordham, N. Masterson-Jones, J. Mcdermid, M. Richmond, M. Rodd, and M. Thomas, "Engineering Values in IT," tech. rep., The Royal Academy of Engineering, July 2009.

[45] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, "Ultra-Large-Scale Systems - The Software Challenge of the Future," tech. rep., Software Engineering Institute, Carnegie Mellon, June 2006.

[46] A. J. Daw, "On the wicked problem of defence acquisition," in *AIAA Aviation Technology, Integration and Operations Conference Challenges in Systems Engineering for Advanced Technology Programmes*, BAE Systems, 2007.

[47] Working group from the The Royal Academy of Engineering and The British Computer Society, "The challenges of complex IT projects," tech. rep., Royal Academy of Engineering and The British Computer Society, January 2010.

[48] N. Roberts, "Wicked problems and network approaches to resolution," *International Public Management Review*, pp. 1–19, 2000.

[49] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, *et al.*, "Manifesto for Agile Software Developmenti." `http://www.agilemanifesto.org` [Last Accessed September 2014], 2001.

[50] G. Symes and A. J. Daw, "On the use of information management in the context of Through Life Capability Management (TLCM)," *Journal of Naval Engineering*, vol. 45 Book 2, 2009.

[51] R. E. Horn and R. P. Weber, "New tools for resolving wicked problems: Mess mapping and resolution mapping processes." `http://www.strategykinetics.com/New_Tools_For_Resolving_Wicked_Problems.pdf` [Last accessed January 2010].

[52] R. Dawkins and J. R. Krebs, "Arms races between and within species," *Proceedings of the Royal Society of London. Series B, Biological Sciences*, vol. 205, no. 1161, pp. 489–511, 1979.

[53] B. Gray, "Review of acquisition for the secretary of state for defence." http://www.bipsolutions.com/docstore/ReviewAcquisitionGrayreport.pdf [Last Accessed August 2014], October 2009.

[54] L. van Valen, "A new evolutionary law," *Evolutionary Theory*, vol. 1, pp. 1–30, 1973.

[55] M. Benton, "Red queen hypothesis," *Palaeobiology (ed. DEG Briggs & PR Crowther). Blackwells. Oxford*, 1995.

[56] L. Caroll, *Through the looking glass, Signet Classic Edition*, pp. 146 – 147. New Amercian Library, 2000.

[57] P. R. Newton, "Future air and space operational concept," tech. rep., Ministry of Defence, August 2009.

[58] J. Cartlidge and S. Bullock, "Unpicking tartan CIAO plots: Understanding irregular coevolutionary cycling," *Adaptive Behavior*, vol. 12, no. 2, pp. 69–92, 2004.

[59] R. A. Watson and J. B. Pollack, "Coevolutionary dynamics in a minimal substrate," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pp. 702–709, Morgan Kaufmann, 2001.

[60] J. Cartlidge and S. Bullock, "Learning lessons from the common cold: How reducing parasite virulence improves coevolutionary optimization," in *In Fogel, D. (Ed.), Congress on Evolutionary Computation*, pp. 1420–1425, IEEE Press, 2002.

[61] J. Cartlidge and S. Bullock, "Combating coevolutionary disengagement by reducing parasite virulence," *Evolutionary Computation*, vol. 12, pp. 193–222, 2004.

[62] B. Sinervo and C. Lively, "The rock-paper-scissors game and the evolution of alternative male strategies," *Nature*, vol. 380, pp. 240–243, 1996.

[63] A. J. Daw, "Challenges in systems engineering for advanced technology programmes - the wicked problem of defence acquisition," in *AIAA Aviation Technology, Integration and Operations Conference*, September 2007.

[64] I. Bailey, "Brief introduction to MODAF with v1.2 updates." http://www.modelfutures.com/file_download/6/Intro%20to%20MODAF%20v1_2.pdf, September 2008.

[65] MODAF Group, Ministry of Defence, "Defence lines of development analysis with modaf." http://www.mod.uk/NR/rdonlyres/1F17FAAA-04C3-4CDD-B2C8-037AC0EA63F6/0/20090210_MODAFDLODAnalysis_V1_0_U.pdf [Last Accessed January 2013].

[66] J. Zachman, "Keynote speech, integrated enterprise architecture conference," 2010.

[67] J. A. Zachman, "A framework for information systems architecture," *IBM Systems Journal*, vol. 26, 1987.

[68] C. W. Holsapple and M. P. Sena, “Erp plans and decision-support benefits,” *Decision Support Systems*, vol. 38, no. 4, pp. 575 – 590, 2005.

[69] MODAF Group, Ministry of Defence, “MOD architecture framework (MODAF) meta model (m3).” `http://www.mod.uk/DefenceInternet/AboutDefence/CorporatePublications/InformationManagement/MODAF/ModafMetaModel.htm` [Last Accessed January 2013].

[70] L. H. Chiew, “The singapore ministry of defence’s approach to enterprise architecture,” in *Integrated Enterprise Architecture Conference 2010*.

[71] T. Engevall, “Enterprise architecture in the Swedish armed forces,” in *Integrated Enterprise Architecture Conference 2010*, 2010.

[72] MODAF Group, Ministry of Defence, “MOD architecture framework - viewpoints and views.” `http://www.mod.uk/DefenceInternet/AboutDefence/WhatWeDo/InformationManagement/MODAF/ViewpointsAndViews.htm` [Last Accessed January 2013].

[73] J. L. Jaensch and D. P. Mahoney, “Modeling and simulation master plan,” tech. rep., Department of Defence, October 1995.

[74] J. Logsdon and R. Wittman, “Standardization, Transformation, & OneSAF,” *Improving M&S Interoperability, Reuse and Efficiency in Support of Current and Future Forces*, pp. pp. 20–1 – 20–14, RTO–MP–MS–056, Paper 20, 2007.

[75] “One Semi-Automated Forces (OneSAF) Mission Needs Statement (MNS),” May 1997.

[76] C. R. Karr, “OneSAF behaviours, OneSAF users conference.” `http://www.onesaf.net/community/documents/Papers_Presentations/OneSAF_UsersConference_2004/OneSAF-UC04_Behaviors.pdf` [Last Accessed August 2014], 2004.

[77] IEEE-SA Standards Board, “IEEE 1516-2000 - standard for modeling and simulation high level architecture - framework and rules,” tech. rep., September 2000.

[78] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, “The Department of Defense high level architecture,” in *Proceedings of the 29th conference on Winter simulation*, pp. 142–149, IEEE Computer Society, 1997.

[79] J. Dingel, D. Garlan, and C. Damon, “Bridging the HLA: A Case Study in Composing Publish-Subscribe Systems,” *International Conference on Software Engineering 2002*, 2002.

[80] R. Kewley, J. Cook, N. Goerger, D. Henderson, and E. Teague, “Federated simulations for systems of systems integration,” in *WSC ’08: Proceedings of the 40th Conference on Winter Simulation*, pp. 1121–1129, Winter Simulation Conference, 2008.

[81] R. C. Zittel, “The reality of simulation-based acquisition–and an example of US military implementation,” tech. rep., DTIC Document, 2001.

[82] D. Webster, N. Looker, D. Russell, L. Liu, and J. Xu, “An Ontology for Evaluation of Network Enabled Capability Architectures,” *RNEC’08: Realising Network Enabled Capability*, 2008.

[83] C. Venters, D. Russell, L. Liu, Z. Luo, D. Webster, and J. Xu, “A Scenario-Based Architecture Evaluation Framework for Network Enabled Capability,” in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, pp. 9–12, IEEE, 2009.

[84] R. G. Ingalls, “Introduction to simulation,” in *WSC ’01: Proceedings of the 33nd conference on Winter simulation*, (Washington, DC, USA), pp. 7–16, IEEE Computer Society, 2001.

[85] A. M. Law and D. W. Kelton, *Simulation Modelling and Analysis*. McGraw-Hill Education - Europe, April 2000.

[86] J. H. Holland, “Complex adaptive systems,” *Daedalus*, vol. 121, no. 1, pp. 17–30, 1992.

[87] M. Kuhl, N. Steiger, F. Armstrong, and J. Joines, “Tutorial on agent-based modeling and simulation,” in *Proceedings of the 2005 Winter Simulation Conference*.

[88] P. Davidsson, “Agent based social simulation: A computer science view,” *Journal of Artificial Societies and Social Simulation*, vol. 5, no. 1, p. 7, 2002.

[89] T. Hoverd and S. Stepney, “Environment orientation: An architecture for simulating complex systems,” in *Proceedings of the 2009 Workshop on Complex Systems Modelling and Simulation*, pp. 67–82, August 2010.

[90] R. M. Fujimoto, “Parallel discrete event simulation,” in *WSC ’89: Proceedings of the 21st conference on Winter simulation*, (New York, NY, USA), pp. 19–28, ACM, 1989.

[91] S. Schlesinger, R. Crosbie, R. Gagne, G. Innis, C. Lalwani, J. Loch, R. Sylvester, R. Wright, N. Kheir, and D. Bartos, “Terminology for model credibility,” *Simulation*, vol. 32, no. 3, pp. 103–104, 1979.

[92] A. Anton, “Goal-based requirements analysis,” *Proceedings of International Conference on Requirements Engineering 1996*, pp. 136 – 144.

[93] E. Letier, *Reasoning about agents in goal-oriented requirements engineering*. PhD thesis, Universite catholique de Louvain, 2001.

[94] T. P. Kelly, *Arguing Safety A Systematic Approach to Safety Case Management*. PhD thesis, Department of Computer Science, University of York, UK, 1998.

[95] T. Kelly and R. Weaver, “The goal structuring notation–a safety argument notation,” in *Proc. DSN 2004 Workshop on Assurance Cases*, Citeseer, 2004.

[96] J. Miller, J. Mukerji, *et al.*, “Model driven architecture (MDA),” Tech. Rep. ormsc/2001-07-01, Object Management Group, 2001.

[97] M. Mernik, J. Heering, and A. M. Sloane, “When and how to develop domain-specific languages,” *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, 2005.

[98] C. Wienands and M. Golm, “Anatomy of a visual domain-specific language project in an industrial context,” *Model Driven Engineering Languages and Systems*, vol. 5795, pp. 453–467, 2009.

[99] R. Lemesle, “Transformation rules based on meta-modeling,” in *Enterprise Distributed Object Computing Workshop, 1998. EDOC ’98. Proceedings. Second International*, pp. 113 –122, November 1998.

[100] F. Jouault and I. Kurtev, “Transforming models with ATL,” in *Satellite Events at the MoDELS 2005 Conference* (J.-M. Bruel, ed.), vol. 3844 of *Lecture Notes in Computer Science*, pp. 128–138, Springer Berlin / Heidelberg, 2006.

[101] D. Kolovos, *An Extensible Platform for Specification of Integrated Languages for Model Management*. PhD thesis, PhD thesis, University of York, 2008.

[102] Object Management Group, “Meta object facility (MOF) 2.0 query/view/transformation specification.” http://www.omg.org/spec/QVT/1.0/PDF/, April 2008.

[103] A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood, “Transformation: The missing link of mda,” in *Graph Transformation*, pp. 90–105, Springer, 2002.

[104] J. Bodeveix, M. Filali, J. Lawall, and G. Muller, “Formal methods meet domain specific languages,” in *Integrated Formal Methods*, pp. 187–206, Springer, 2005.

[105] “Eclipse.” http://www.eclipse.org/.

[106] “Eclipse EMF - Eclipse modelling framework.” http://www.eclipse.org/emf.

[107] D. Kolovos, R. Paige, and F. Polack, “The Epsilon Transformation Language,” *Theory and Practice of Model Transformations*, pp. 46–60, 2008.

[108] L. Rose, D. Kolovos, R. Paige, and F. Polack, “Model migration with Epsilon Flock,” *Theory and Practice of Model Transformations*, pp. 184–198, 2010.

[109] D. Kolovos, L. Rose, R. Paige, and F. Polack, “Raising the level of abstraction in the development of GMF-based graphical model editors,” in *Proceedings of the ICSE Workshop on Modeling in Software Engineering*, pp. 13–19, 2009.

[110] S. Kirkpatrick, J. Gelatt, C. D., and M. P. Vecchi, “Optimization by Simulated Annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[111] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, 2003.

[112] J. D. Schaffer, “Multiple objective optimization with vector evaluated genetic algorithms,” in *Proceedings of the 1st International Conference on Genetic Algorithms*, (Hillsdale, NJ, USA), pp. 93–100, L. Erlbaum Associates Inc., 1985.

[113] M. Pirlot, “General local search methods,” *European Journal of Operational Research*, vol. 92, no. 3, pp. 493 – 511, 1996.

[114] S. Stepney, "Non Standard Computation Lecture Slides - Local Search." `http://www-module.cs.york.ac.uk/nstc/lectures/02%20local.pdf` [Last Accessed August 2014].

[115] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput. Oper. Res.*, vol. 13, no. 5, pp. 533–549, 1986.

[116] E. Teller, N. Metropolis, and A. Rosenbluth, "Equation of state calculations by fast computing machines," *J. Chem. Phys*, vol. 21, no. 13, pp. 1087–1092, 1953.

[117] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[118] A. Konak, D. Coit, and A. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[119] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[120] A. Saltelli and T. Homma, "Sensitivity analysis for model output : Performance of black box techniques on three international benchmark exercises," *Computational Statistics & Data Analysis*, vol. 13, no. 1, pp. 73 – 94, 1992.

[121] A. Saltelli, *Sensitivity analysis in practice: a guide to assessing scientific models*. John Wiley & Sons Inc, 2004.

[122] A. Saltelli, S. Tarantola, and K. P.-S. Chan, "A quantitative model-independent method for global sensitivity analysis of model output," *Technometrics*, vol. 41, no. 1, pp. pp. 39–56, 1999.

[123] G. A. Gorry and M. S. S. Morton, "A framework for management information systems," *Sloan Management Review*, pp. 55–70, 1971.

[124] R. Anthony, "Planning and Control Systems: A Framework for Analysis, Cambridge, MA, Harvard University Graduate School of Business Administration," *Studies in Management Control*, 1965.

[125] S. Herbert and S. M. Dillon, "The new science of management decision," in *In Proceedings of the 33 rd Conference of the Operational Research Society of New Zealand*, 1960.

[126] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," tech. rep., DTIC Document, 1990.

[127] Ministry of Defence, "The MODAF strategic viewpoint." `https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/38710/20100426MODAFStVViewpoint1_2_004U.pdf` [Last Accessed August 2014], April 2010.

[128] L. H. De Figueiredo, R. Ierusalimschy, and W. Celes Filho, "The design and implementation of a language for extending applications," *Anais do XXI Semish*, pp. 273–283, 1994.

[129] R. Ierusalimschy, L. H. De Figueiredo, and W. Celes Filho, "Lua-an extensible extension language," *Softw., Pract. Exper.*, vol. 26, no. 6, pp. 635–652, 1996.

[130] Y. Zhang, M. Harman, and S. Mansouri, "The multi-objective next release problem," in *Proceedings of the 9th annual conference on Genetic and Evolutionary computation*, pp. 1129–1137, 2007.

[131] D. Kolovos, L. Rose, S. Abid, R. Paige, F. Polack, and G. Botterweck, "Taming EMF and GMF using model transformation," *Model Driven Engineering Languages and Systems*, pp. 211–225, 2010.

[132] "Eclipse graphical modeling project." `http://www.eclipse.org/modeling/gmp/`.

[133] S. Efftinge and M. Völter, "oaw xtext: A framework for textual dsls," in *Workshop on Modeling Symposium at Eclipse Summit*, vol. 32, 2006.

[134] E. Urwin, C. Venters, D. Russell, L. Liu, Z. Luo, D. Webster, M. Henshaw, and J. Xu, "Scenario-based design and evaluation for capability," in *System of Systems Engineering (SoSE), 2010 5th International Conference on*, pp. 1–6, June 2010.

[135] K. Deb, "Multi-objective optimization," in *Search Methodologies* (E. K. Burke and G. Kendall, eds.), pp. 273–316, Springer US, 2005.

[136] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.

[137] J. del Sagrado, I. del A andguila, and F. Orellana, "Ant colony optimization for the next release problem: A comparative study," in *Second International Symposium on Search Based Software Engineering*, pp. 67 –76, 2010.

[138] A. Bagnall, V. Rayward-Smith, and I. Whittley, "The Next Release Problem," *Information and Software Technology*, vol. 43, no. 14, pp. 883–890, 2001.

[139] D. Greer and G. Ruhe, "Software release planning: an evolutionary and iterative approach," *Information and Software Technology*, vol. 46, no. 4, pp. 243–253, 2004.

[140] J. Durillo, Y. Zhang, E. Alba, and A. Nebro, "A study of the multi-objective next release problem," in *1st International Symposium on Search Based Software Engineering*, pp. 49–58, 2009.

[141] British Army, "Mastiff - British Army website." `https://www.army.mod.uk/equipment/23248.aspx` [Last Accessed January 2014].

[142] British Army, "Vector - British Army website." `https://www.army.mod.uk/equipment/23246.aspx` [Last Accessed January 2014].

[143] British Army, "Mastiff - British Army website." `http://www.army.mod.uk/equipment/23275.aspx` [Last Accessed January 2014].

[144] T. Gander, *Jane's Military Vehicles and Ground Support Equipment 1985 (Jane's Yearbooks).* Jane's Information Group, 1985.

[145] British Army, "5 Regiment - British Army website." `http://www.army.mod.uk/artillery/regiments/24678.aspx` [Last Accessed January 2014].

[146] Armed Forces International, "Buffalo armoured vehicle - explosives removal." `http://www.armedforces-int.com/projects/buffalo_armoured_vehicle.html` [Last Accessed January 2014].

[147] British Army, "Close support tanker - British Army website." `http://www.army.mod.uk/equipment/23269.aspx` [Last Accessed January 2014].

[148] Armed Forces.co.uk, "British Army - the royal logistics corps - vehicles - drops - truck utility." `http://www.armedforces.co.uk/army/listings/l0146.html` [Last Accessed January 2014].

[149] Northrop Grumman, "Global hawk." `http://www.northropgrumman.com/capabilities/globalhawk/Pages/default.aspx` [Last Accessed January 2014].

[150] Armed Forces.co.uk, "Reacher satellite ground terminal." `http://www.armedforces.co.uk/army/listings/l0103.html#Reacher` [Last Accessed January 2014].

[151] British Army, *British Army Vehicles and Equipment.* March 2009.

[152] Royal Air Force, "RAF - MQ-9 reaper." `http://www.raf.mod.uk/equipment/mq9reaper.cfm` [Last Accessed January 2014].

[153] D. Kolovos, R. Paige, and F. Polack, "The Epsilon Object Language (EOL)," in *Model Driven Architecture: Foundations and Applications* (A. Rensink and J. Warmer, eds.), vol. 4066 of *Lecture Notes in Computer Science*, pp. 128–142, Springer Berlin / Heidelberg, 2006.

[154] "JFreeChart." `http://sourceforge.net/projects/jfreechart` [Last Accessed August 2014].

[155] C. Daly, M. Garcia, and L. Bigeardel, "Emfatic language for EMF developmentibm alphaworks, 2004." `http://www.alphaworks.ibm.com/tech/emfatic`.

[156] M. Wenz, C. Brand, F. Velasco, J. Pasch, M. Gorning, T. Kaiser, and C. Brun, "Graphiti - a graphical tooling infrastructure." `http://www.eclipse.org/graphiti/` [Last Accessed August 2014].

[157] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager, "SWI-Prolog," *Theory and Practice of Logic Programming*, vol. 12, no. 1-2, pp. 67–96, 2012.

[158] A. van Lamsweerde, “Goal-oriented requirements engineering: a roundtrip from research to practice,” in *Requirements Engineering Conference, 2004. Proceedings. 12th IEEE International*, pp. 4–7, IEEE, 2004.

[159] Amazon, “Amazon elastic compute cloud (ec2).” `http://aws.amazon.com/ec2/` [Last Accessed January 2014].

[160] E. Cantú-Paz, “A summary of research on parallel genetic algorithms,” tech. rep., Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL., 1995.

[161] A. Morse, “Strategic financial management of the defence budget,” tech. rep., National Audit Office, July 2010.

[162] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and using nonfunctional requirements: A process-oriented approach,” *Software Engineering, IEEE Transactions on*, vol. 18, no. 6, pp. 483–497, 1992.

[163] H. Hansson and B. Jonsson, “A logic for reasoning about time and reliability,” *Formal aspects of computing*, vol. 6, no. 5, pp. 512–535, 1994.

[164] S. Peluchetti, “SciLua.” `http://www.scilua.org` [Last Accessed January 2014].

[165] R Development Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.