Product modularity: a multi-objective configuration approach

Product Modularity:
A Multi-objective Configuration Approach

By

Michael Lee

A Doctoral Thesis

Submitted in Partial Fulfilment of the

requirements for the Award of

Doctor of Philosophy

of Loughborough University

Wolfson School of Mechanical and Manufacturing Engineering

May 2010

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Keith Case for his valuable advice, guidance and continuous support throughout my PhD. I am grateful for his patience and for him believing in my ability to produce good quality work.

I would like to thank my second supervisor Dr. Russell Marshall for his guidance and valuable advice. I am grateful to Prof. Shahin Rahimifard for his encouragement and support during my write-up period.

I would also like to thank my family for all their love, support and encouragement to pursue my interests. Finally, I am forever grateful to my loving, supportive and encouraging partner Simona for believing in me and always being there for me when I needed her.

ABSTRACT

Product modularity is often seen as a means by which a product system can be decomposed into smaller, more manageable chunks in order to better manage design, manufacturing and after-sales complexity. The most common approach is to decompose the product down to component level and then group the components to form modules. The rationale for module grouping can vary, from the more technical physical and functional component interactions, to any number of strategic objectives such as variety, maintenance and recycling. The problem lies with the complexity of product modularity under these multiple (often conflicting) objectives.

The research in this thesis presents a holistic multi-objective computer aided modularity optimisation (CAMO) framework. The framework consists of four main steps: 1) product decomposition; 2) interaction analysis; 3) formation of modular architectures and; 4) scenario analysis. In summary of these steps: the product is first decomposed into a number a basic components by analysis of both the physical and functional product domains. The various dependencies and strategic similarities that occur between the product's components are then analysed and entered into a number of interaction matrixes. A specially developed multi-objective grouping genetic algorithm (MOGGA) then searches the matrices and provides a whole set of alternative (yet optimal) modular product configurations. The solution set is then evaluated and explored (scenario analysis) using the principles of Analytic Hierarchy Process.

A software prototype has been created for the CAMO framework using Visual Basic to create a multi-objective genetic algorithm (GA) based optimiser within an excel environment. A case study has been followed to demonstrate the various steps of the framework and make comparisons with previous works. Unlike previous works, that have used simplistic optimisation algorithms and have in general only considered a limited number of modularisation objectives, the developed framework provides a true multi-objective approach to the product modularisation problem.

Abbreviations

DfX	:	Design for x
DM	:	Decision maker
DSM	:	Design structure matrix
EOL	:	End of life
GA	:	Genetic algorithm
MFD	:	Modular function deployment
MOGA	:	Multi-objective genetic algorithm
MOGGA	:	Multi-objective grouping genetic algorithm
QFD	:	Quality function deployment

Table of Contents

CHAPTER 1

1. Introduction

1.1. Research Context

To design and manufacture successful and profitable products in an increasingly competitive, globalised society, driven by a consumer-based economy presents many manufacturing companies with huge challenges. In most industries product development times are becoming shorter as companies are forced to offer an increasing number of new product offerings to the market in order to remain competitive. In the automotive industry for example product development times for new products have shrunk from 60 months in 1988 to 35 months in 1999, and this trend is set to continue well into the foreseeable future (Nepal et al., 2006). In addition product development now takes place on a global level with many companies outsourcing the design and manufacture of product parts to various suppliers around the world. Tighter environmental legislation is also having effects on the way products are developed - in some industries the producer now has ensure their products are able to meet strict recycling and reuse targets. In order for companies to remain profitable under these demanding conditions a number of strategic design and manufacture strategies are often sought - the careful consideration of the product architecture is often seen as a one such key strategy.

Broadly speaking product architecture can be defined as either integral (closed) or modular (open) and is often defined in terms of two characteristics of product design: the similarity between the physical and functional architecture of the design and the incidental interactions between physical components (Ulrich and Tung, 1991). An integral architecture has one-to-many or many-to-one relationships between functional and physical elements and complex interactions between components. In contrast a modular architecture has a one-to-one mapping of function to form and well defined interfaces between modules. These

characteristics help give a modular product architecture a number of advantages such as: increased product variety at lower costs, ease of outsourcing, reduced order lead-times, decoupling of design and assembly tasks, ease of product upgrade and change and ease of service and recycling.

However, the very fact that modularity attempts to address so many strategic objectives has given rise to many different definitions over the years and a large range of measures, methods and frameworks have been created in an attempt to guide the development of modular product architectures. Generally speaking however, product modularity is primarily seen as a product structuring concept, in which the product system is decomposed into smaller more manageable chunks (modules) in order to better manage design and manufacturing complexity. The most common way this is done is by the decomposition of the product down to component level and then grouping of the components to form modules. As well as Ulrich and Tung's (1991) two objectives of function binding and physical component interactions, the rationale for module grouping also includes any number of strategic objectives, such as: variance and commonality, maintenance and reliability, outsourceability, reusability and recyclability.

As can be expected, creating a suitable modular product architecture under these many strategic considerations is a complex task, as there are often a huge number of potential ways to partition a product's architecture into modules. And despite the range of developed techniques there are still problems to be solved. In summary, the basic principles of product modularity have been well studied, yet there have been few frameworks created to guide the product modularisation process under multiple (potentially conflicting) strategic objectives. These methods generally use simple clustering algorithms or (aggregate objective-based) grouping algorithms. Due to the simplistic nature of these models they do not provide a true multi-objective optimisation and hence cannot guarantee that truly optimal modular product architectures can be found.

To better address the complex nature of product modularisation a more holistic multi-objective modularity framework will be developed in this thesis. It is intended that the framework will be useful to industry, enabling product developers to produce a well thought-out modular product architecture that is aligned with the strategic needs of the product throughout the whole product lifecycle. In fact, it is argued that for all but the most basic of products, the proposed modularisation framework would be beneficial. Although it must be stated that products with a relatively high level of complexity (such as modern mechatronic based consumer products), would benefit the most from modularisation via a multi-objective optimisation framework. Such products typically have complex functionally, a mixture of technologies and will often have many different modularisation objectives, which will subsequently lead to a large number of alternative modular configurations that can be generated and explored using the proposed framework.

1.2. Research Aims and Objectives

The overall aim of the research is to develop a multi-objective optimisation framework for product modularisation. This has resulted in the following research objectives:

- *to critically review the literature on product modularity.* A thorough literature study is required to gain a deep understanding of what defines a modular product and what are the main advantages and disadvantages of modular product architecture, both from academic and industrial perspectives. The focus of the review is to critically assess previous product modularisation methods and to indentify a number of key principles that can be developed/integrated into a product modularisation framework.

- *to technically review GA-based optimisation algorithms.* Genetic algorithms are considered a promising type of optimisation algorithm for the product modularisation problem. A technical review of the fundamental operational principles of genetic algorithm based optimisation is required together with an

examination of the current state-of-the-art in genetic algorithm based multi-objective optimisation.

- *to analyse the main drivers of product modularisation.* The main drivers of product modularisation must be identified in order to establish a logical set of modularisation objectives. For each objective a suitable analysis and evaluation method must then be developed.

- *to develop a representation method and metrics.* For modularisation to occur there must be a method of product representation that effectively captures the complex physical and functional interactions and other strategic interactions that occur between the product components. Based upon this representation, metrics must then be developed to measure the 'goodness' of candidate modular solutions.

- *to produce a software prototype for multi-objective product modularisation.* This requires the following sub-tasks: a) coding of the optimisation algorithm using the appropriate software language; b) the production of a suitable software interface for data input and storage of product modularisation attributes c) a method to visualise the candidate modular architectures and a means to compare and contrast alternative solutions.

- *to demonstrate the approach on a case study example.* A case study is required in order to test and validate the method and to comparisons and contrasts to previous works within the field.

1.3. Thesis Structure

The overall structure of the thesis can be seen in figure 1.1. The thesis is comprised of four main sections.

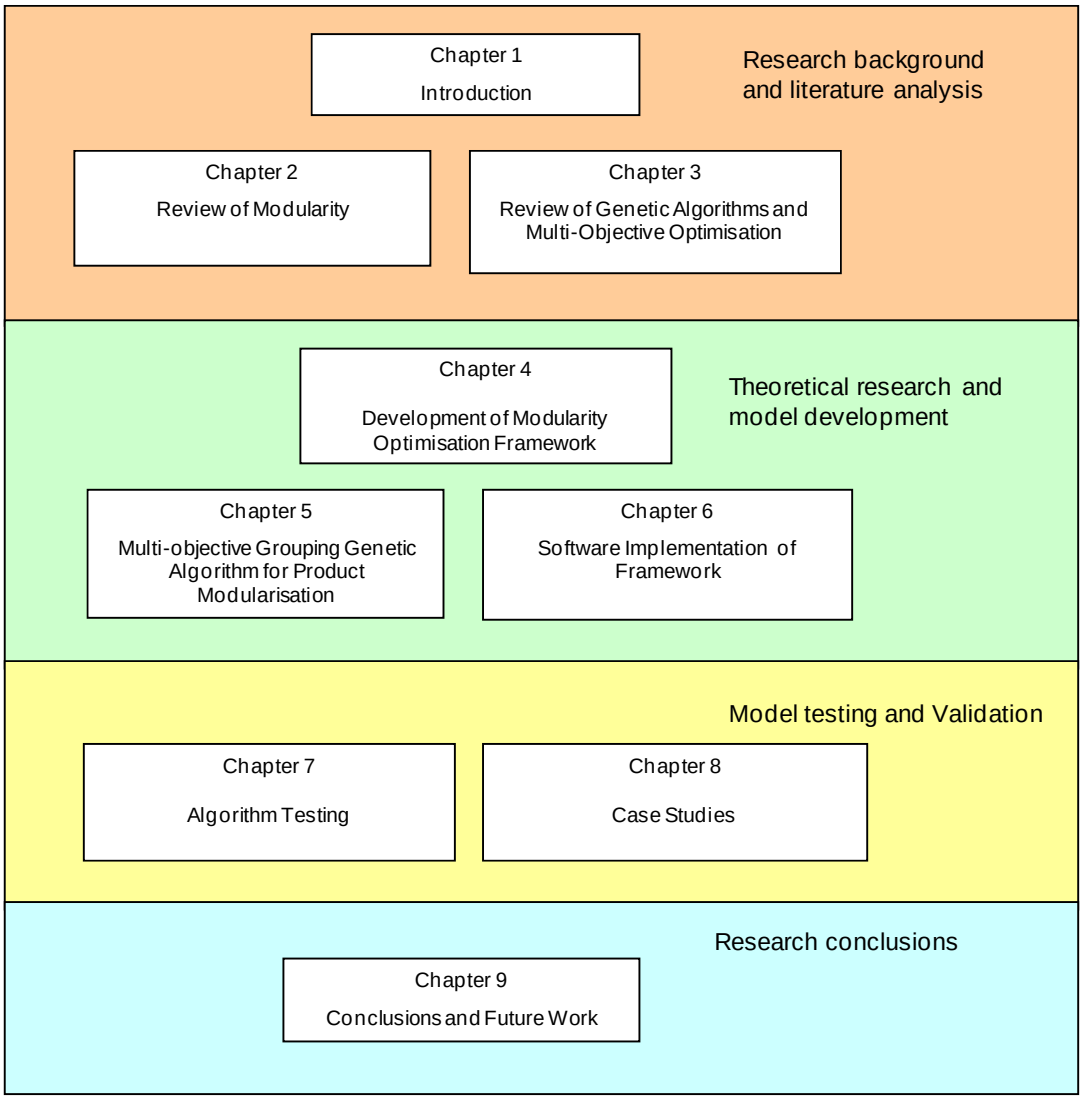

Figure 1.1. Thesis structure

In the first section of the thesis the research aims and objectives are established and the relevant literature is analysed.

- Chapter 1 presents the research context, the aims and objectives and the structure of the thesis.

- Chapter 2 provides a review of modularity, looking first at the advantages of modularity and the various definitions and viewpoints, then moving on to provide a critical review of the existing modularisation methods.
- Chapter 3 examines the fundamental operation principles of GAs and looks at GA based multi-objective optimisation.

Section 2 of the thesis contains the chapters relating to the theoretical research and development of the modularisation model.

- Chapter 4 provides a general overview of the product modularisation framework in which key aspects of the framework are discussed.
- Chapter 5 outlines the development of the multi-objective genetic algorithm, customised specifically for the multi-objective product modularisation problem.
- Chapter 6 Discusses the software implementation of the framework and outlines the evaluation guidelines for the recommended modularisation objectives.

In section 3 are the chapters relating to testing and validation of the developed modularisation model.

- Chapter 7 tests the various performance aspects of the algorithm and makes comparisons with a traditional aggregated (weighted objective) GA.
- Chapter 8 demonstrates the application of the modularisation framework using a case study taken from the literature. Comparisons are also drawn between other case study results and frameworks from previous works.

In the last section of the thesis, that is chapter 9, the research conclusions are drawn and recommendations for future work are presented.

CHAPTER 2

2. Review of Modularity

2.1. Introduction

Modularity has been given many definitions over the years and a broad range of measures, methods and techniques have been created in attempts to guide the development of modular product architectures. The literature surrounding modularity is diverse and large, with researchers approaching modularity from various viewpoints. The question then asked is are there any major fallacies or weaknesses in the previous approaches? Or indeed can any of these different ideas be integrated into a more holistic framework to optimise product modularity? To answer these questions a thorough literature review has been conducted, examining the major English-language modularity literature from the past three decades.

This chapter provides a detailed review of product modularity. The first part of the review examines the fundamental modularity benefits and viewpoints in a bid to unravel what actually constitutes a modular product. The focus of the review then moves on to consider the various modularisation methods that have been produced to guide the development of modular products.

2.2. Modularity Benefits

A huge number of benefits can be achieved from modularity- which have been exemplified by numerous researchers and can be seen throughout the whole of the product lifecycle. The starting point for this chapter is therefore to discuss the key modularity benefits in relation to each of the four main stages of the product lifecycle; namely, design, production, use and end of life.

From a design or product development point of view, modular design allows for the dividing of the overall product design into smaller sub-tasks so that design teams

can carry out the sub-tasks in parallel speeding up product development. (Gu and Sosale, 1999). This is primarily achieved through the reduction of interactions between modules (or chunks) and has been found to positively affect product development time (Loch et al, 2003; Griffin 2002; Yassine et al, 2003; Carrascosa et al, 1998; Sosa et al, 2003).

The reuse of existing design and production processes with minimal changes is also possible through the use of modular design (Gu and Sosale, 1999; Ericsson and Erixon, 1999). In addition, modularity enables efficient responses to design changes that may occur during the product's life cycle. (Martin and Ishii, 2000; Tate et al, 1998; Kusiak, 2002).

In production, a module is predominately seen as a kind of strategically formed sub-assembly, where modules can be assembled in the most convenient locations and then put together to reduce the total assembly time and costs (Fixson, 2003). For example, when the Golf II was developed the focus of modularisation was on ease of assembly (Wilhelm, 1997). This was also observed at Mazda by Kinutani (1997). In fact, in some industries, such as the automotive industry, manufacturers arrange the manufacture and assembly of automobile components at different sites and bring them together for final assembly (Sako and Murray, 2000; Warburton and Sako, 1999; Takeishi and Fujimoto, 2001).

Modularity at the production phase can be seen to offer other advantages. According to Ishii (1998), the benefits include 'streamlined suppliers, reduced inventory, fewer works in progress, faster process time..etc…' For, example, if modules can be reused across product families or multiple product generations (commonality), scale effects can reduce the cost per unit by distributing the fixed cost across larger volumes.

For the product use phase modularity can be seen to offer two main advantages. Firstly, from the customer perspective 'modularity in use', as defined by Baldwin and Clark (1997), allows customers to mix and match a variety of product offerings

to suit their needs. Secondly, modularity facilitates ease of maintenance, service and repair (Gershenson and Prasad, 1997b; Gu and Sosale, 1999; Ishii, 1998; Newcomb et al, 1996).

Lastly, at the end of the product's life, modularity can be used as a means by which to facilitate the ease of recycling, remanufacture and reuse. For example high material homogeneity within modules can dramatically reduce the number of disassembly operations necessary to separate the product into its various material streams for retirement (Zhang and Gershenson, 2003; Gu and Solace, 1999; Ishii, 1998; Newcomb et al, 1996).

It must be noted however that there are potential costs of modularity, these include: (1) The lack of globalised product performance due to decreased function sharing. (2) The product may have excessive capability due to over standardisation (over designing modules for the most rigorous application). (3) There is potential for static product architectures and excessive product similarity. (4) Reverse engineering is simpler which may lead to increased competition. (Ulrich, 1995).

2.3. Modularity Viewpoints

Examination of the modularity literature has revealed that there are four primary viewpoints of product modularity and these are function, coupling, variety and life-cycle.

Function

From a product development perspective, perhaps the most common view of product modularity, product modularity relates to product function (Erens and Verhulst, 1997; Stone, 1997; Baldwin and Clark, 1997; Huang and Kusiak, 1999; Sanchez, 2002; Suh, 2001; Pahl and Beitz, 1984; and Jiao and Tseng, 2000; Ulrich and Tung, 1991). For example, as part of Suh's axiomatic design method, the first

axiom states that ‘in good design the independence of functional requirements is maintained’. Indeed, a functional perspective is often adopted during the design process as it presents a natural means of converting customer needs into product requirements (Pahl and Beitz, 1984).

The functional perspective implies that a complex product can be decomposed into clearly defined functions and that each function can be mapped to physical components. From this perspective product architecture is defined by the way in which functional elements correspond to physical components. The product architecture is said to be modular when it exhibits a one to one mapping between functional and physical elements (Ulrich and Tung, 1991). This leads to the notion that each module should carry out a discrete function, becoming functionally independent. Functionally independent modules can be seen to offer a number of advantages to product development, that include ease of product upgrade, improved product configurability and function sharing among product families (functional modules can be mixed and matched to address a range of customer needs), and improved product reuse - should a function change then only part of the product needs to be redesigned. Conversely, an integral architecture can be defined when each function is implemented by multiple physical components or when each physical component implements numerous functions. Ulrich (1995) uses an example of a trailer to illustrate the principles (see figures 2.1 and 2.2).

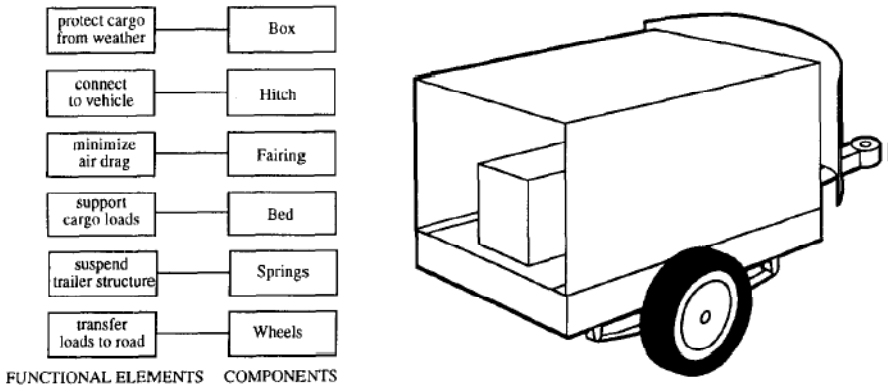

Figure 2.1 Modular Architecture (Ulrich, 1995)

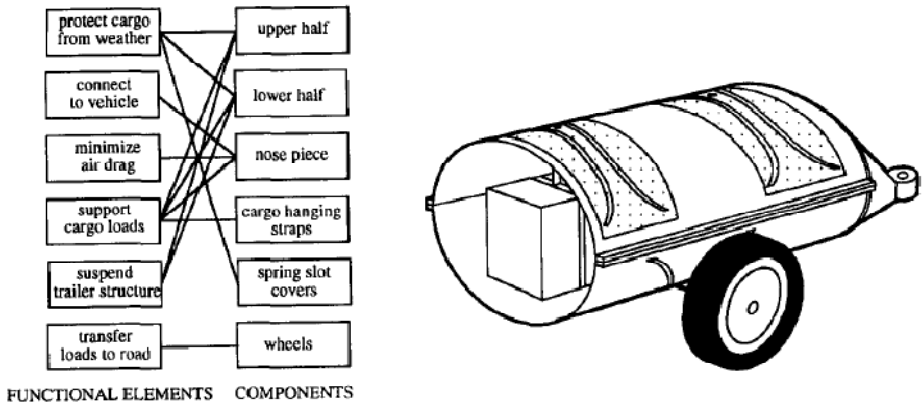

Figure 2.2. Integral Architecture (Ulrich, 1995)

One of the main advantages of ensuring that physical elements are aligned to functions (i.e. each module performs a discrete function) is that each module is more closely aligned to a specific customer need (Pahl and Beitz, 1984), which, in turn, enables a greater variety of products to be produced at a minimal level of complexity and cost. Because in a modular product functions are less integrated (spread among components), different customer needs can be addressed by different modules, allowing a mix and match of modules to enable product variety at low costs (Ulrich, 1995; Pahl and Beitz, 1984). In other words, firms can rearrange and/or add new modules to achieve the desired functions without changing the whole product.

Achieving functionally discrete modules is desirable. However, product functions can be analysed in numerous ways and at various levels of abstraction and hence

we are told that functional decomposition is very subjective (Baldwin and Clark, 2000). Consider the hair dryer example (Fixson, 2003). “Its main function is to dry hair. If to dry hair were selected as a function, the result would be the allocation of this function to all components, for all components of the hair dryer would not exist in the first place if they were not contributing to the product functionality. On the other hand, if the function is chosen on a very low, detailed level: “Hold part A in position X relative to part B with force F”, then exactly one and only one component delivers these functions. In contrast if one begins to define functions like “to generate air flow, heat airflow, control heat, control air flow, supply energy”..., then it becomes meaningful to investigate how functions are mapped to components”.

Coupling

The coupling view of modularity is perhaps one of the most prevailing among scholars. From this perspective, modularity is viewed as the number/complexity of interactions (couplings) between components; the fewer the interactions between modules the more modular the product becomes (Ulrich and Eppinger, 1995). Figure 2.3 illustrates the principle, where product A can be seen to be more modular than product B. Lowering the degree of coupling between modules can be seen to positively affect various aspects throughout the product life cycle. During the design phase development time can be reduced (Loch et al, 2003; Griffin 2002; Yassine et al, 2003; Carrascosa et al, 1998; Sosa et al, 2003). During production modules can be assembled with relative independence, enabling the benefits of flexible assembly lines and module outsourcing (Sako and Murray, 2000; Hsuan, 1999). Loosely coupled modules also enable simpler removal of modules for service operations, easier replacement of failed modules and simpler disassembly operations for module recycling and remanufacturing (Newcomb et al, 1996; Gu and Sosale, 1999; Gershenson et al., 1999; Marks et al, 1993).

Numerous measures of component couplings have been developed and vary significantly, both in qualitative and quantitative terms. Martin and Ishii (2000) developed a metric called the coupling index, that is used to measure if the components are sensitive to changes in another component's design specifications. They use a 1-9 scale to assess the sensitivity of a component to each design specification flow between components (sub-assemblies).

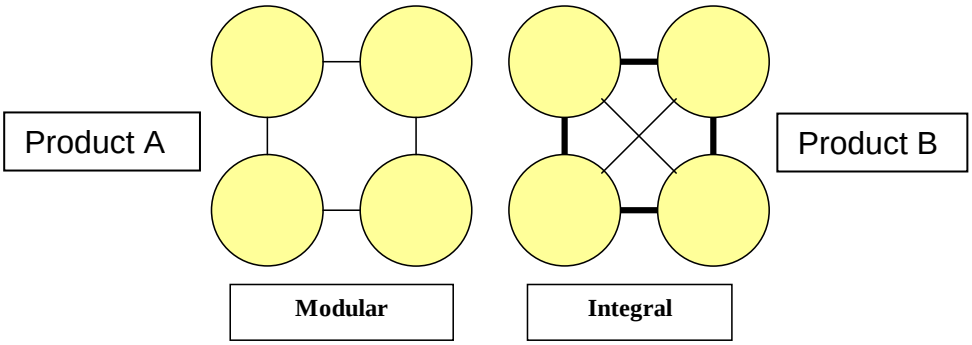

Figure 2.3. Component interactions for modular and integral architectures

In Pimmler and Eppinger's (1994) DSM based approach couplings are evaluated in terms of functional and physical interactions (physical interaction, flows of material, energy, information), with each being quantified on a four point scale (+2 to -2). Holtta and Otto (2005) also use a functional flow method to define coupling, but extend the principle to also quantify the level of component redesign effort needed should one or more of the functional flow intensities change. The works of Gu and Sosale (1999) and more recently Lai and Gershenson (2008) present a measure of coupling that concentrates on the physical interactions. The aim here is to ensure that efficient assembly is possible and that each module is easily detachable. The measures are based upon the geometric mating complexity and the joining methods used between components.

Some modularity definitions also include the notion of standardised interfaces (Sanchez , 2002; Mikkola and Gassman, 2003). Module interfaces can be seen as the physical manifestations that arise from the component couplings. When interfaces are standardised or 'fixed' it becomes possible to exploit the notion of inter-changeability between modules to offer the customer an increased range of end products, at minimum cost to the company.

Variance

One of the key aspects of a modular design approach it that it gives the manufacturer the ability to offer an increased number of product variants to the market place without accruing huge costs. By breaking down a product into a number of discrete modules, a huge range of product variants can be offered to the market at low cost through a module 'mix and match' approach. For example, Kusiak (2002) uses the term modularity to describe the use of common units (modules) to create product variants. Kusiak defines modularity as the identification of independent, standardised or interchangeable units to satisfy a variety of functions. From this perspective, a product becomes increasingly modular when a larger number of modules can be readily reconfigured or shared with other products (Mikkola and Gassman, 2003). The personal computer (PC) is a good example. By using a well defined modular product architecture, with standardised interfaces, all manner of PC configurations can be achieved by using a mixture of different modules.

This modularity viewpoint has been discussed by many other researchers including: Sanchez, (1999, 2002); Baldwin and Clark, (1997, 2000); Jiao and Tseng, (2000); Mikkola (2001); Mikkola and Gassman, (2003); O'Grady, (1999); Pine, (1993); Pahl and Beitz, (1984); Fujita, (2002); Fujita and Yoshida, (2001); Salvador et al, (2002); Robertson and Ulrich, (1998); Chakravarti and Balkrishan, (2001); Dahmus et al, (2001); Duray et al, (2000); Ulrich and Tung, (1995); Jose and Tollenaere, (2005).

Developing modular product platforms is often the way modular product family development is implemented. With this approach a family of products are created based upon common (shared) product architecture (Simpson et al. 2001; Muffato, 1999; Muffato and Roveda, 2000; Nayak et al, 2000; Farrel and Simpson, 2003; Gonzalez-Zugasti et al, 1999, 2001; Meyer and Lehnerd, 1997). Central to the modular product platform approach is the identification of common platform modules, that is modules that can be made common across the product family (Simpson et al, 2001). Common platform modules offer the advantages of greater

economies of scale, reduced product inventories and improved product reuse leading to a faster time to market (Nobelius and Sundgren; 2002). Integrating a number of components into a common platform module can also offer increased product performance though reduced part count and a reduced number of interfaces. Another key point with the platform approach is that production lead times can also be reduced by postponing the product differentiation point i.e. common platforms can be assembled first then variant modules can be added later in the production cycle (Salvador et al, 2002; Feitzinger and Lee, 1997). This is commonly referred to as a late point product differentiation strategy.

Generally speaking the product should contain an optimal balance of both variant and platform modules (Krishnan and Gupta, 2001). Many common modules may lead to lower costs but this may also mean that the product family does not successfully address the different market needs and that each individual product loses its identify. In other words, the product range becomes internally cannibalised (Kim and Chajed, 2000; Silveria et al, 2001). On the other hand, if the level of variety is too high then problems will also occur – primarily the design and production costs are likely to increase to unacceptable levels. Too much variance can also be overwhelming for the customer and may not actually lead to greater sales. Deciding upon the right level of variety to offer the customer is thus a complex issue. Market research tools and approaches such as Voice of the Customer (VOC), conjoint analysis (Moore et al, 1999; Tatikonda, 1999) and data mining (Agard and Kusiak, 2004) can be used. A detailed discussion of these principles is however out of the scope of this research.

It has also been discussed by numerous scholars that for a modular approach to product variety management to be fully successful the module interfaces should be standardised and carefully managed after modules have been indentified, allowing the product family mix to be carefully controlled. This can in part be achieved by looking at the ‘coupling’ viewpoint to ensure functional and physical interactions between modules are kept as simple as possible. A certain amount of overdesign or bandwidth may also need to be incorporated into certain modules to allow a range

of different specification modules to be used together. For example the mother board of a PC needs a certain design bandwidth to support a range of CPU speeds, RAM capacities and graphics card speeds.

Life-cycle

The modularity definitions proposed so far are related primarily to the functional and physical characteristics of modules. However, other researchers have chosen to include other product lifecycle based aspects in their definitions of modular products. The lifecycle viewpoint of modularity is primary concerned with the after-sales aspects of modularity i.e. the service, replacement, recycling, reuse and remanufacturing of modules. For these after-sales lifecycle factors some researchers have even gone so far as to use these aspects as the main drivers of product modularisation. The Newcomb et al (1996) modularisation method for example was primarily focused on product modularisation to facilitate ease of recycling. Newcomb et al (1996) uses a three point scale to evaluate the material similarity between component pairs in an interaction matrix.

Gershenson et al. (1999) however view modularity from a whole lifecycle viewpoint. They define lifecycle modularity as 'modules and interactions that arise from the various processes the components undergo during their life-cycle including development, testing, manufacturing, assembly, packaging, shipping, service and retirement'. Their methods have been used in pursuit of service (Gershenson and Prasad, 1997b), manufacturing (Gershenson and Prasad, 1997a), retirement (Zhang and Gershenson, 2003) and assembly (Lai and Gershenson, 2008). Similarly, the bodies of work by Gu and Solace (1999), Ishii (1998), Nepal (2005) and Nepal et al (2006 and 2007) also see modularity as a means of improving various product life cycle goals.

2.4. Product Modularisation Methods

There has been a large number of modularity methods developed over the years. Rather than reviewing each and every method in a linear fashion, the various works have been examined for similarities. In this thesis three different categories of modularity methods have been indentified, namely: configuration methods, domain mapping approaches and step-wise redesign methods.

The configuration approaches are by far the most prevalent in the literature. With these methods the product is decomposed into a number of smaller elements (components) which are then grouped to form larger product elements (modules). The rationale for modular configuration varies, depending upon the modularity viewpoint taken.

The configuration based methods are predominately matrix based, with most employing some form of grouping or clustering algorithm to form modules. Matrix representations are considered a somewhat natural way of representing component interactions, as they are highly visual and can readily be manipulated with algorithms. The component interactions are generally functionally and physically based, but also based upon the similarity of components in regards to any number of strategic modular drivers. For the majority of the configuration based methods it is assumed that the basic product elements are known. The objective of these methods is to configure the product elements into optimal modules according to the objectives being sought. The methods presume that each component has a clear function at the indentified level of decomposition. The possibility that a component's function can be carried out by a different component does not exist. For example, for a vacuum cleaner, the *disposable bag* has the function of *collect dust*, which could be carried out by a different functional component, such as a *plastic bucket collector*.

The domain mapping approaches deal with product modularisation at a higher level of abstraction and are most useful for new product development. These methods

are predominantly based upon the functional viewpoint of modularity- whereby the modular product is developed though mapping between the functional and physical domains. For example, Pahl and Bietz (1984) suggest the use of a functional mapping approach as a means to decompose the conceptual product family into a number of modules. In this way a number of products can be built up simultaneously by considering commonalty and function sharing between product family members.

For some of the modularity methods the primary focus is on the analysis and redesign of existing modular structures. These methods have been labelled *step-wise redesign* methods. With these methods it is often presumed that the product elements have already been grouped to form modules. Hence the methods merely act as an analysis and redesign tool, seeking to guide the designer towards an improved modular structure by adjusting the existing module attributes. For example Martin and Ishii's (2000) design for variety (DFV) method is a widely quoted method used to improve the robustness of existing modular product architectures to accommodate future product versions.

2.5. Configuration Based Methods

As mentioned the configuration based methods are by far the most prevalent in the literature and thus this is where the focus of this review will lie. With all the configuration methods the principle is to group lower level components into higher level modules. As will be seen the objectives for the grouping, as well as the means of performing grouping, vary greatly.

Design Structure Matrix (DSM) Based Clustering Methods

The Design Structure Matrix (DSM) is a powerful tool for representing product architectures. This representation allows for the analysis and development of modular products by clustering of the DSM (Yassine et. al., 2003). The DSM was

first proposed by Steward (1981) for design activity planning. It consists of a square matrix headed by a list of the product's components (can also be activities, processes or functions) that are represented in the same order in both the row and column of the matrix. The matrix represents the interactions between the components. These interactions can be represented as a simple binary number or a weighted amount that represents the degree of dependency. Figure 2.4 shows an example DSM. Interactions between the components can be physical, functional or even strategic based. Strategic interactions are the interactions that arise from the component similarities in regards to any number of lifecycle/ strategic factors e.g. recycling, maintenance and variety.

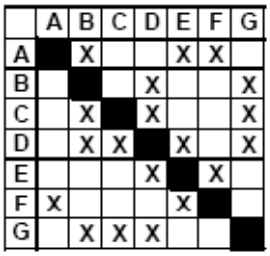

Figure 2.4. Un-clustered DSM (Yassine, 2003)

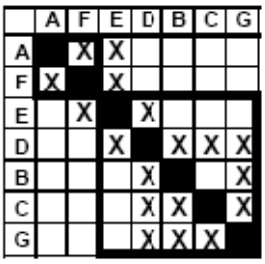

Figure 2.5. Clustered DSM (Yassine, 2003)

Once the DSM has been populated with component interactions, the aim is to cluster the matrix such that groups of highly interactive components can be identified. These highly interactive clusters of components then become candidate modules. The identified modules can therefore be seen to conform to the second part of the Ulrich and Eppinger's (1995) modularity definition: chunks (modules) should contain few if any interactions between chunks (modules).

The seminal work of Pimmler and Eppinger (1994) applies the DSM for evaluating product architecture, primarily for product development purposes. The goal of their method is to reduce interactions that occur between clusters (chunks as they put it). The idea is that coordination complexity of the development effort can be reduced if interactions predominately occur within chunks rather than between chunks. These interactions are quantified in terms of:

- Spatial. The need for adjacency or orientation between elements.
- Energy. The need for energy transfer between two elements.
- Information. The need for information or signal transfer between two elements.
- Material. The need for material exchange between two elements.

Each of these interaction types is given a +2 to -2 score. For an example of spatial interactions see table 2.1.

Table 2.1 Physical component interaction scoring- Pimmler and Eppinger (1994)

Required: (+2)	Physical adjacency is necessary for functionality.
Desired: (+1)	Physical adjacency is beneficial, but not absolutely necessary for functionality
Indifferent: (0)	Physical adjacency does not affect functionality.
Undesired: (-1)	Physical adjacency causes negative effects but does not prevent functionality.
Detrimental: (-2).	Physical adjacency must be prevented to achieve functionality

Although the method draws parallels with the aims of modular design, Pimmler and Eppinger (1994) do not actually refer to the method as a modular design method. In Pimmler and Eppinger's (1994) work chunks are allowed to overlap each other as can been seen in figure 2.6. For module forming however components should exclusively become part of the same module as seen in figure 2.5.

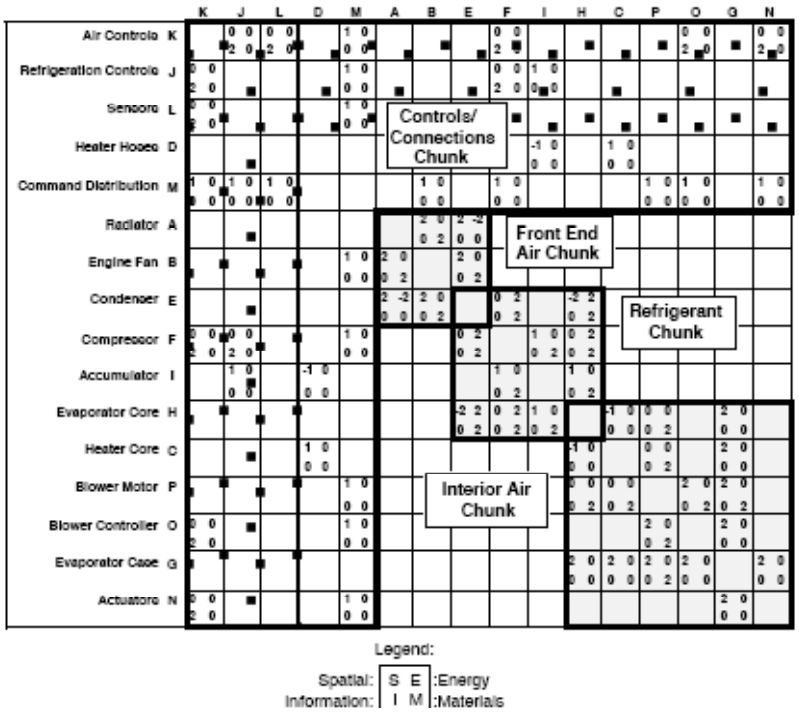

Figure 2.6. Clustering DSM: car climate control system (Pimmler and Eppinger, 1994)

In Pimmler and Eppinger's (1994) work clustering is done manually. If DSM clustering is done manually, the process can be inefficient and extremely time consuming. A number of algorithms such as hill-climbing and simulated annealing have been developed and tested by Whitfield et al (2002) for the optimisation of the order of components within the DSM. Of these methods, Whitfield et al (2002) state that 'the difficulty in optimising the DSM lies in the number of combinations of possible component orders. For example, a matrix containing 30 components has 6.652*1032 possible combinations. An exhaustive search for this type of problem is clearly inappropriate'. Genetic algorithms (GAs) is one method that is particularly suited to such combinatorial optimisation problems. Various researchers have therefore adopted Genetic algorithms for the clustering of the DSM.

Yassine et. al. (2003), have developed a DSM clustering method, used for modular product design, based on a genetic algorithm (GA), which they claim is capable of solving DSM clustering problems with overlapping and bus modules and a three dimensional structure. They have demonstrated the use of their method to cluster a

real-world problem - a 10MW gas turbine - and claim that it produces superior clustering compared to traditional/ manual algorithms. Although the technique has shown promise there are limitations. Firstly, the method is binary based, so it presents no means of distinguishing between the strengths of the dependencies between components. Furthermore, the method does not distinguish between different types of dependencies (physical, material, energy, information).

Whitfield et al (2002), have also developed a GA-based clustering algorithm. The technique uses a weighted method of assessing the strength of dependencies between components. They apply a Module Strength Indicator (MSI) which results in an alternative representation of the DSM- *Module Structure Matrix (MSM)* (as seen in figure 2.7). The focus of the method is towards identifying modules that have a maximum number of internal dependencies between components and a minimum number of external dependencies between components. The resulting *'Module Structure Matrix' (MSM*) uses different coloured cells to depict the relative modularity of all available modules within the matrix (Whitfield et al, 2002).

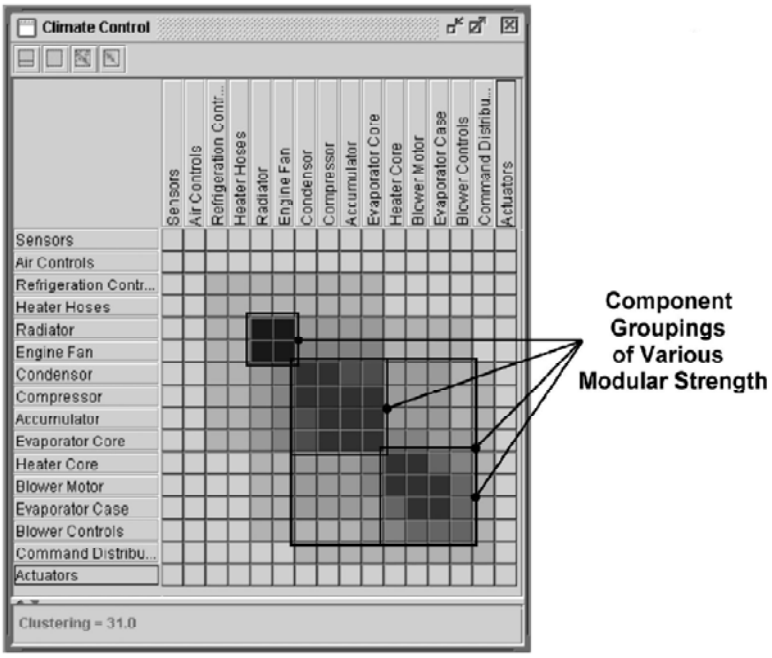

Figure 2.7. MSI-DSM of car climate control system (Whitfield et al, 2002)

Table 2.2. Module Catalogue for Climate System (Whitfield et al, 2002)

Modular Strength	Module	Components
1.0	M_1	Radiator, Engine Fan
0.9	M_2	Condenser, Compressor
	M_3	Compressor, Accumulator, Evaporator Core
	M_4	Heater Core, Blower Motor
	M_5	Blower Motor, Evaporator Case
0.8	M_6	Components within M_2 and M_3
0.7	M_7	Components within M_4 and M_5
	M_8	Components within M_5 and Blower Controls
0.6	M_9	Components within M_8 and Heater Core
	M_{10}	Components within M_4 and Evaporator Core
	M_{11}	Components within M_6 and Engine Fan
0.5	M_{12}	Components within M_{11} and Radiator
	M_{13}	Components within M_6 and M_7
	M_{14}	Components within M_7, Evaporator Core and Blower Controls
0.4	M_{15}	Components within M_1, M_6, Refrigeration Controls and Heater Hoses
	M_{16}	Components within M_1 and M_{13} and Blower Controls
	M_{17}	Components within M_{13}, Blower Controls and Command Distributor
0.2	M_{18}	All

This approach is useful for the identification of a modular hierarchy within the product structure as can be seen in table 2.2. But, it still fails to include different types of dependencies such as: functional, physical or lifecycle/strategic, concentrating only on physical dependencies.

Kamrani and Gonzalez (2003) have developed a similarity matrix for indentifying the optimal modular structure of two similar products. The method is effectively a DSM clustering method that also uses a GA-based clustering algorithm. However unlike previous methods, where modules are easily identifiable, module groupings are hard to visualise.

Kusiak (2002) has developed a modularity method that also looks at the functional interactions between components as a driving force for forming modules. Again a DSM is used to represent component interactions (the interaction matrix). However, Kusiak also includes a second matrix, the suitability matrix (see table 2.3 and figure 2.8), to assess whether the components should be placed into the same module as each other. Component suitability is measured in terms of a four-point scale, a, e, o, u, ('a' representing strongly desired and 'u' strongly undesired). The suitability matrix is used to determine modules that are suitable for sharing across the product

family. i.e if two components are needed in the same product they are deemed suitable for inclusion in the same module. The suitability idea could of course be extended to other factors such as suitability for recycling, maintenance or service.

In Kusiak's method module formation consists of three main steps, which can be applied in an iterative fashion. The first step employs Kusiak and Chow's (1987) triangularisation algorithm to perform clustering of the interaction matrix and establish a 'rough' modular structure. Next the suitability matrix is analysed for 'goodness' of module clusters. Based upon the analysis a number of operations can then be applied to improve modularity. These operations are: the reconfiguration of incompatible components, either to other modules or to form new modules; the elimination of components; and the duplication of components if they are needed in more than one module.

However, with their method there is no guarantee that one would continue to move a design towards a more modular product during the module improvement stage. A measure of modularity could be added to ensure that each move is an improvement. Kusiak also discusses that module cost should be a determining factor for choosing a suitable modular structure but does not actually provide such a module cost metric.

Table 2.3. Summary of the application of the modularity matrix (Kusiak, 2002)

		Interaction matrix			Suitability matrix
Design phase	Product type	Entry interpretation	Column name	Entry value: value interpretation	Entry value: value interpretation
Conceptual phase	All	Function flow	Mechanism/ Subsystem	Integer number: frequency of application	a: strongly desired e: desired o: strongly undesired u: undesired
Detailed phase	Electrical	Electrical flow	Component		
	Mechanical	Force flow, thermal, function, etc.			
	Mixed	Force flow, thermal, function, etc.			

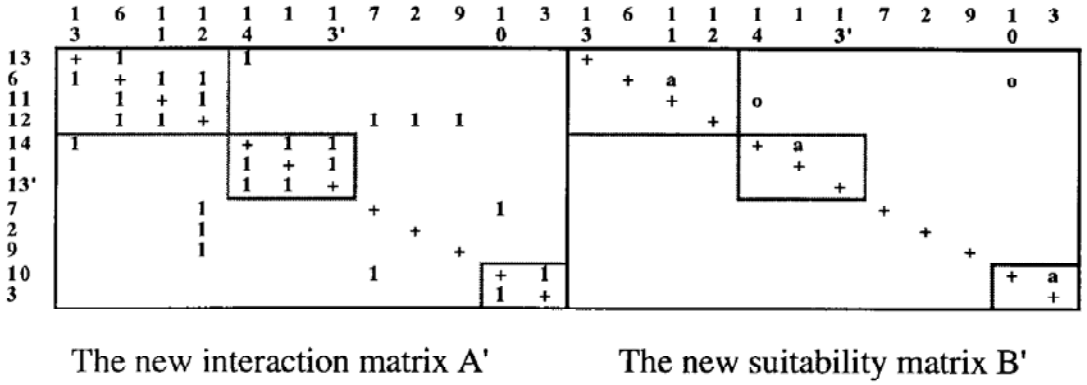

Figure 2.8. Interaction matrix and suitability matrix (Kusiak, 2002)

Modular Function Deployment (MFD) Method

Ericsson and Erixon (1999) have developed a Quality Functional Deployment (QFD) based method known as Modular Function Deployment (MFD) to support the generation and evaluation of modules. This consists of the following steps:

1. Clarify customer requirements (QFD).
2. Select technical solutions.
3. Generate modular concepts (Module-Indication-Matrix).
4. Evaluate concepts, (interface matrix, evaluation chart).
5. Improve each module.

The Ericsson and Erixon (1999) method focuses on the strategic aspects of modularity. The central notion to the method is to use a QFD-style approach to map the influence of various strategic modular drivers (see figure 2.9.) on each 'technical solution' of the product (scored on a 1-9 scale). The matrix produced is known as the Module-Indication-Matrix (MIM). The MIM technical solutions, which are influenced by the same modular drivers, are then considered as potential candidates for grouping into modules. Figure 2.10. shows an example MIM for a vacuum cleaner.

Product development and design	Carryover	A part or a subsystem of a product that most likely will not be exposed to any design changes during the life of the product platform. Enables heavy investment in production technology.
	Technology evolution	Parts that are likely to undergo changes as a result of changing customer demands or technology shift. It will be important to accommodate the interfaces so that new technology can be introduced and replace the module in question.
	Planned product changes	Parts of the product that the company intends to develop and change. (Sony Walkman's consecutive model introductions)
Variance	Different specification	To handle product variation and customization effectively, a designer should strive to allocate all variations to as few product parts as possible. (An example is different specifications for voltage in different parts of the world)
	Styling	Styling modules typically contain visible parts of the product that can be altered to create different variations of the product.
Production	Common unit	Common unit is similar to the shared functions across products described by Dahmus et. al., i.e. parts or subsystems that can be used for the entire product assortment.
	Process and/or organization	Parts requiring the same production process are clustered together. For example, all parts requiring welding may be moved into a single module to enable atomization.
Quality	Separate testing	The possibility of separately testing each module before delivery to final assembly may contribute to significant quality improvements, due to reduced feedback times.
Purchase	Supplier availability	Purchase standard modules from external vendors
After sales	Service and maintenance	Parts exposed to service and maintenance may be clustered together to form a service module to be able to quickly replace and repair/replace it.
	Upgrading	Give customers the possibility of changing the product in the future
	Recycling	The number of materials in each module should be limited. Easily recyclable material can be kept in separate recycling modules.

Figure 2.9. The Modular Drivers - Ericsson and Erixon (1999)

There is no measure of modularity used in the MFD method. The method works on the assumption that the product will become more modular as the designer forms modules that have similar modular driver influences. One weakness of the method is that it is left to the designer to manually group technical solutions into modules. There are no heuristics or clustering algorithms developed to support this task. Another weakness is that it does not consider the functional interactions between technical solutions during the module grouping phase. This may lead to sub-optimal or infeasible module groupings.

Function carrier / Module driver		Fan	Noise absorbent, fan	Electric motor	Damper	Noise absorbent, motor	Chassis	Bag	Filter	Tiristor+knob	Switch+knob	Housing	Wire+contact	Grip	Rear wheel	Front wheel	Accessories	Bumper	Cover	Indicator	Seal, cover	O-ring	Wire collector	Bag lock	Brake+knob
Design and Development	Carry-over	●		●						●	⊘		⊘			●	○			●			●		⊘
	Technology push							●	●																
	Product Planning																								
Variance	Diff. specification	○	○	○					○	○	○		⊘												
	Styling									●	●	●		●	○				●						●
Manuf.	Common unit	⊘	⊘	⊘	●	●	●	●	⊘	⊘	⊘		○		⊘	●	●	●		●	●	●	●	●	⊘
	Process/Org.	●		●			●	●				●													
Quality	Separate testing			●							○														
Purchase	Black-box engineer.								●	●			●												
After sales	Service/maint.			⊘					○	⊘	○														
	Upgrading								●																
	Recycling			●			●					●											○		
●=9 ⊘=3 ○=1	**Weight of Driver vertically summarised**	**22**	**4**	**43**	**9**	**9**	**27**	**27**	**32**	**34**	**18**	**27**	**16**	**9**	**4**	**18**	**10**	**9**	**9**	**18**	**9**	**9**	**19**	**9**	**15**
	Module candidates	√		√			√	√	√	√		√											√		

Figure 2.10. Completed MIM for a vacuum cleaner - Ericsson and Erixon (1999)

Ericsson and Erixon (1999) do however provide an additional step that examines the module interactions after modules have been formed. In the matrix shown in figure 2.11 (A) represents an attachment interface and (T) a transfer interface. The matrix serves as a pointer for the interfaces which should be given consideration and eventually improved. The evaluation is carried out from an assembly point of view. Two ideal assembly types are mentioned: "hamburger assembly" which is an ideal assembly type for DFA reasons, and "base part assembly" which is an ideal assembly type when it comes to maintenance and replacement of parts.

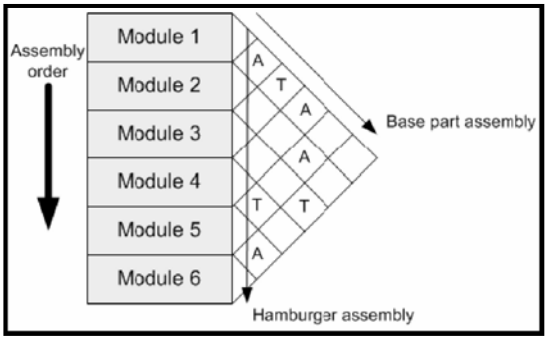

Figure 2.11. Interface evaluation matrix - Ericsson and Erixon (1999)

Derivatives of the MFD Method

Blackenfelt (2001) has attempted to address some of the weakness of MFD by the use of two interaction matrices (DSMs); one for functional interactions and one for the strategic aspects. The functional-based matrix is similar to that of Pimmler and Eppinger (1994) representing component interactions in terms of spatial, energy, material and information. The strategic matrix represents the similarity of modularity influences between components. For this purpose Blackenfelt (2001) uses a reduced set of modular drivers taken from the MFD method. He argues that there are overlaps and contradictions between the original drivers in the MFD and hence justifies his reduced set as being more appropriate. The drivers he uses are: 'variant versus common', 'reuse versus develop', 'make versus buy' and 'carry-over versus change' To quantify the component interactions in respect to these modular drivers the affects on component pairs are quantified on a +3 to -3 scale. If there is a positive correlation between modular drivers, for example if both components should be common, then a +3 is entered into the corresponding position in the matrix. Likewise, if two components have conflicting modular drivers, one component should be common, whilst the other should be variant then a -3 is entered. This can be seen in figure 2.12. Blackenfelt also provides simple evaluation guides for each of his four modular driver pairs (example in figure 2.13.)

	TS 1		TS 2		TS 3		TS 4		TS 5		TS 6	
TS 1	RD	CV										
	CC	MB										
TS 2		-2	RD	CV								
	-1		CC	MB								
TS 3		-1		1	RD	CV						
	-1		1		CC	MB						
TS 4		-1		1		1	RD	CV				
	-1		2		1		CC	MB				
TS 5		2		-2		-1		-1	RD	CV		
	-1	2	-1		-1		-1		CC	MB		
TS 6		1		-1		-1		-1		1	RD	CV
	1	-2	-1		-1		-1		-1	-2	CC	MB

Figure 2.12 Strategic DSM (Blackenfelt, 2001)

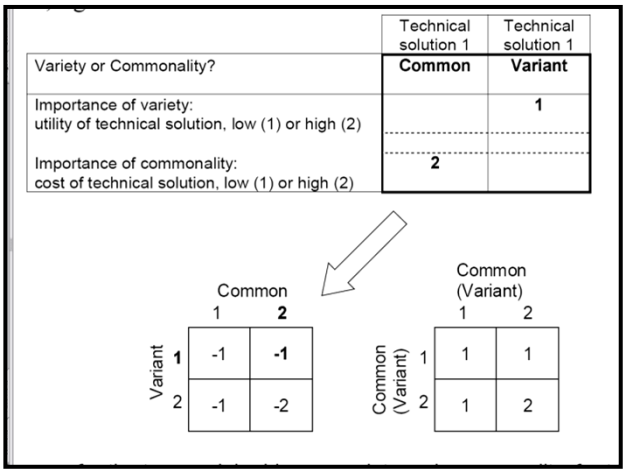

Figure 2.13. Modular driver scoring for strategic DSM (Blackenfelt, 2001)

Once the matrix has been populated the idea is to use a clustering algorithm to form modules that maximise the positives within modules whilst at the same time minimising the negatives within modules. To guide the clustering Blackenfelt also proposes two measures. However, although Blackenfelt (2001) developed some module grouping guidelines he did not develop a suitable optimisation algorithm, so like Ericsson and Erixon's (1999) method of grouping it must be done manually, making it difficult to find optimal module groupings. This could of course be overcome by developing a suitable grouping algorithm.

Kreng and Lee (2004) and Kreng and Tseng (2004) have developed an extended QFD-based modular configuration approach. For their method a modular driver matrix (like the MIM) is used to represent the impact of modular drivers on each component. Additionally, a QFD method is followed to assess the importance of modular drivers based upon the identified customer and company needs – as depicted in figure 2.14. Like Pimmler and Eppinger's (1994) DSM approach a functional matrix is also used to represent functional and physical interactions between components. In an additional improvement upon previous methods they have developed a genetic algorithm based non-linear programming model to perform module grouping. Groupings are formed based upon the maximisation of a weighted sum of two metrics. One metric measures the functional interactions within modules and one measures the modular driver similarities within modules. However, both metrics only measure module interactions within modules and

ignore the influences of external module (between module) interactions, which may lead to sub-optimal modules. Another weakness of their approach is that the identified modules may well end up containing components with conflicting strategic modular driver influences, as the method does not include negative modular driver influences, only positive ones. Also, unlike Blackenfelt's (2001) method they have not considered the potential conflicts and similarities of the actual modularity drivers themselves. The method also fails to give any detailed guidance on how to score the modular driver influences on components.

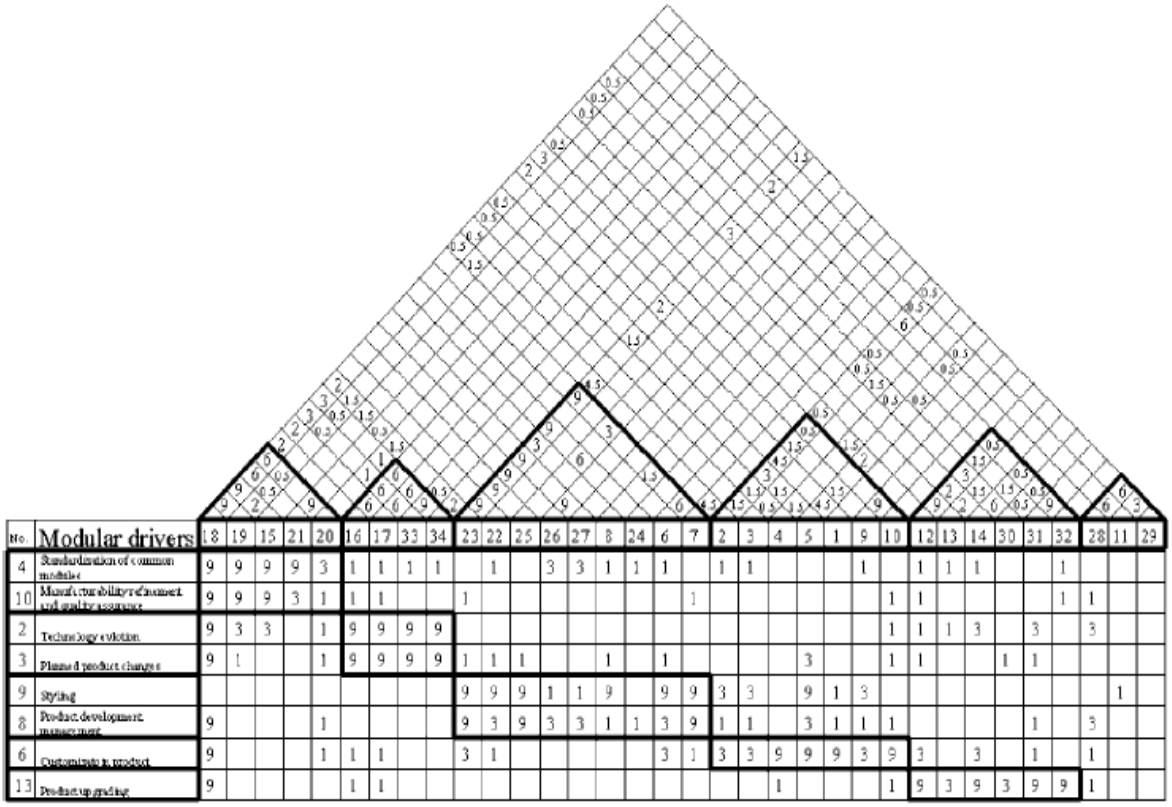

Figure 2.14. QFD modular design method: clustered modular driver matrix (Kreng and Lee, 2004)

Lifecycle Based Approaches

Gershenson et al (1999) have developed a method of module design from a life-cycle perspective. The approach focuses on module independence and similarity across the product life-cycle and includes a step-wise configuration and redesign methodology to guide designers towards greater modularity of products. The goal

of their modular design method is to group all components that undergo the same life-cycle processes into the same module as well as decoupling the modules from module dependences that arise from the various life-cycle processes.

Their method begins by firstly examining a number of manufacturing process graphs and component assembly trees, from which two modularity matrices are constructed, one to record similarities between processes and one to record dependencies between components. Modularity is evaluated based on a relative modularity measure they have developed, which is the ratio of intra-module similarities to all similarities, both intra- and inter-module, added to the ratio of intra-module dependencies to all dependencies, both intra- and inter-module. The similarities considered are component–process similarities while the dependencies are both component–component and component–process dependencies (Gershenson et al., 1999). To improve relative modularity, components or modules can be eliminated, rearranged or redesigned in order to increase the modularity measure. The redesign step focuses on either reducing the intra-module similarities and dependencies or increasing inter-module similarities and dependencies.

One shortcoming of the method is the work necessary to apply it. The matrices need deep product knowledge and tedious re-design work is needed to move towards better modularity. Gershenson et al (1999) state that the method could be improved by semi-automating the evaluation and reconfiguration process, which has been part of the later works of Zhang and Gershenson (2003) who also include a module clustering step employing Kusiak and Chow's (1987) clustering algorithm.

More recently, Lai and Gershenson (2008) have attempted to address some of the shortcomings of their group's previous efforts by integrating a number of previous ideas into their method. The improved method makes use of two matrixes, one for component dependences and one for component lifecycle similarities. After these matrices are populated they are normalised and averaged to form one matrix. The single matrix is then clustered by again employing Kusiak and Chow's (1987)

clustering algorithm. Once initial clusters have been formed a reconfiguration stage takes place to further improve module clusters in respect to their previously defined modularity metric. These newly improved module clusters are then checked for feasibility using a simplified assembly precedence graph. Lastly a redesign step is applied to further improved modularity by changing component attributes. Although the method is aimed at modular design for assembly, the method could easily be extended to other lifecycle phases, which is actually what Gershenson's group intend to do next. The method appears to improve upon Gershenson's previous work but the method is still lacking a certain level of automation during the module clustering stage. It would be expected that a mathematical programming based module grouping method, such as the GA-based integer programming approach used by Kreng and Lee (2004) would perform better module grouping. It would also eliminate the need for the post-cluster reconfiguration stage.

Gu and Solace (1999) also pursue a life-cycle based approach to modularity. They aim to produce various different product modularisations for the various life-cycle characteristics of a product including for example assembly, reusability and recyclability. Their design method has three phases:

1) *Problem definition:* this includes identification of type and characteristics of design problems, decomposing the problem into sub-problems, and determining the objectives of modularisation. For conceptual design this will include the decomposition of the product based upon a functional structure of the product. For product redesign the physical structure will already be known, so decomposition is the identification of the components or sub-systems. They define this 'decomposition' step as a prerequisite for modularising a product. They further discuss that defining the objective of modularity will depend upon the type of product. For some products the aim will be ease of assembly, for others it may be ease of maintenance.

2) *Interaction analysis:* To evaluate the interactions for the objective, values are assigned to each objective. For each objective an interaction matrix is created to

record the interactions between components which are scaled to lie between 0 and 10. A single matrix is then produced that represents a total interaction score (for all objectives) between components. Like Pimmler and Eppinger (1994) they consider functional interactions among components in terms of exchanges of materials, energy, and signals, or spatial interactions and extend their work to include geometric relationships. The geometric relationships include attachment, positioning, motion and containment. The lifecycle based component interactions they consider include recycling and reuse and maintenance and service.

3) *Module formulation*: a simulated annealing based mathematical programming algorithm is then implemented to cluster the components into modules such that the component to component interactions within each module are maximised. Different solutions can be obtained by changing the preference weights for the various modularity objectives.

One of the problems with the method is that it only considers interactions within modules and does not consider interactions between modules, hence modules may not be optimal. Furthermore because the method combines the various matrices to produce a single matrix (which is optimised) it ignores the fact that there may in fact be modular driver (modularisation objective) conflicts within modules (negative interactions) and hence the produced modules may not even be feasible or desirable.

Newcomb et al (1996) present a method that aims to improve modularity in terms of three main lifecycle factors, function, service and end of life needs. In their work they attempt to create a modular structure that has a high level of correspondence between the modules from the various life-cycle viewpoints. This is based upon the hypothesis that the product will have more than one modular structure and there may in fact be a different set of modules for each life-cycle phase. Their method is carried out in two main stages. Firstly, like others, the initial step consists of the clustering of an interactions matrix to identify possible modules. For this Kusiak and Chow's (1987) clustering algorithm is employed.

Once the module clusters have been chosen two modularity metrics are used to access module 'goodness'. The first metric evaluates the level of module coupling and the second metric measures the correspondence of the modular structure with regard to service and recyclability. Based upon these measures a limiting factors heuristic is used to indentify how the modules could be improved. For example, the limiting factors for a module maybe its poor recyclability between a pair of components, and hence changes to the component's materials would be suggested to improve recyclability of the module. Using this limiting factors approach the process of redesign and improvement for modules continues in an iterative manner until a stopping point is reached or no further improvements can be made. One problem with this approach is that after each iterative redesign is performed the original limiting factors may no longer be limiting factors, so the limiting factors will need constant updating. It could also be said that the limiting factors will be highly dependant upon the chosen module groupings, which is unfortunately done by manually choosing 'optimal modules' from the clustered interaction matrix.

Nepal's Fuzzy Logic Method

Nepal (2005) presents a structured process based on fuzzy logic and goal programming models for developing modular products. The aim is to form optimal module configurations based upon cost (Nepal, 2005), quality (Nepal et al., 2006) and reliability and maintainability (Nepal et al, 2007). Like previous methods the product is decomposed into lower level components and then they are grouped to form modules. Modules are formed based upon the maximisation of similarities between components. Again, like other methods, component similarities are represented in a DSM-type matrix. The interesting facet of their work is the inclusion of fuzzy logic into the component - component similarity assessment. For each objective (cost, quality, and reliability and maintainability) a set of three performance metrics is used to evaluate each component pair (module candidates as they refer to it). Using fuzzy logic an overall 'performance index' is quantified for each component pair according to how the three metrics are scored. For example, for the module quality objective there are three metrics: 'perceived quality';

'robustness' and 'compliance to axiomatic design principles'. Each of these metrics has an associated evaluation chart. In table 2.4 an example evaluation chart for 'perceived quality' can be seen.

Table 2.4. Evaluation chart for 'perceived quality' (Nepal et al, 2006)

Rating	Quality Level	Description of Perceived Quality
1, 2	Poor	The candidate module does not or poorly complies with the customers' expectations about product size, shape, serviceability etc. and also has some safety-related issues.
3,4	Fair	The candidate module complies with the customers' expectations about product size, shape, and distinctiveness but is somewhat flawed.
5,6	Moderate	The candidate module partially complies with the customers' expectations about product flawlessness, serviceability, maintainability etc., but it may not be elegant from an aesthetic point of view.
7,8	High	The candidate module fairly complies with the customers' expectations about product size, shape, flawlessness, distinctiveness etc.
9,10	Very High	The candidate module significantly complies with the customers' expectations about product size, shape, flawlessness, distinctiveness etc.

Once the component pairs have been evaluated for each metric the fuzzy logic is applied. The fuzzy logic works in the following way: first the user makes judgements for each of the various combinations of metric evaluation scores and a number of rules are generated, for example:

Rule# 1: If (*Perceived Quality* is *Very Low*) AND (*Robustness* is *Very Low*) AND (*Axiomatic Compliance* is *Very Low*) THEN (*Quality Performance Index* is *Very Low*)

Rule# 42: If (*Perceived Quality* is *Low*) AND (*Robustness* is *High*) AND (*Axiomatic Compliance* is *Very Low*) THEN (*Quality Performance Index* is *Moderate*)

These fuzzy outputs (the THEN statements) are then converted using defuzzification mathematics into 'crisp' numeric values. These values then form the component interaction values for each component pair in the matrix. To form modules a Chebychev (min-max) goal programming model is applied to search the matrix for the optimal groupings that maximise the overall product quality index (and/or reliability and maintainability) and lower the cost performance index. With

goal programming 'aspiration levels' are used to represent the desired achievement of the goals (cost, quality and reliability and maintainability in this case) by adjusting the 'aspiration levels' and trade-off analysis can be performed to find alternative modular structures. The results of the trade-off between cost and quality can be seen in figure 2.15.

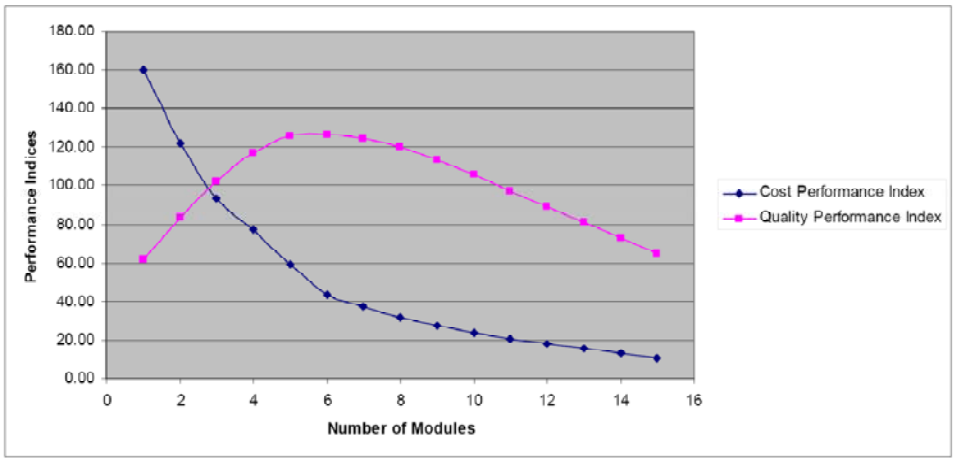

Figure 2.15. The trade-off between cost and quality ((Nepal et al, 2006)

The very fact that trade-off analysis can be performed in a logical manner gives Nepal's work an advantage over previous works. However the method, like others, requires a lot of work to evaluate the various metrics and would need considerable time and knowledge to set up the various fuzzy relationships (rules) between the metrics. In fact by integrating the various modularity metrics in such a way a certain amount of granularity maybe lost during the trade-off process. The method also fails to consider other key modularity drivers such as variety, design change, recyclability and reuse and outsourcing. However, it can be presumed that metrics could be developed for inclusion into the framework.

Stone's Functional Heuristic Method

One of the few modular design methods that does not use a matrix representation of the product is the functional modelling based approach of Stone (1997). The main aim of the method is to create an optimal modular structure by identifying low level sub-functions which are grouped into modules. The Stone (1997) approach uses a function structure diagram based upon the previous works of Pahl and Beitz (1984). The principle of this is the tracing of functional flows (material, energy and signal) through the product. For every customer need, a flow is identified and a black box model of a product's overall function and input/output flows is drawn up (see figure 2.16). Each flow identified in the black box model is then traced through the product, as it would flow during use, through a sequence of sub-functions that change the flow. A completed functional model can be seen in figure 2.17.

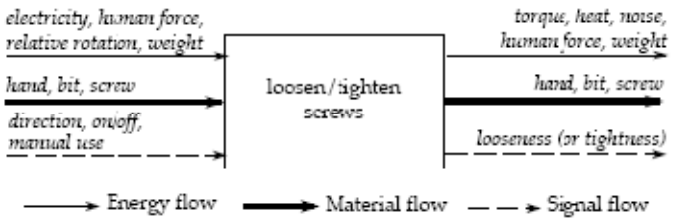

Figure 2.16. Black box model for an electric screwdriver (Stone et al, 2000)

Once the functional model has been completed modules can be identified by grouping sub-functions that have strong functional interactions with one another. For this purpose Stone (1997) has developed three heuristics: 1) dominant flows: a set of sub-functions for a flow that passes through from system entry/flow initiation to system exit/flow conversion; 2) branching flows: a set of sub-functions for making a parallel function chain associated with a branched flow; and 3) conversion-transmission flows: a set of sub-functions responsible for the transition between flows.

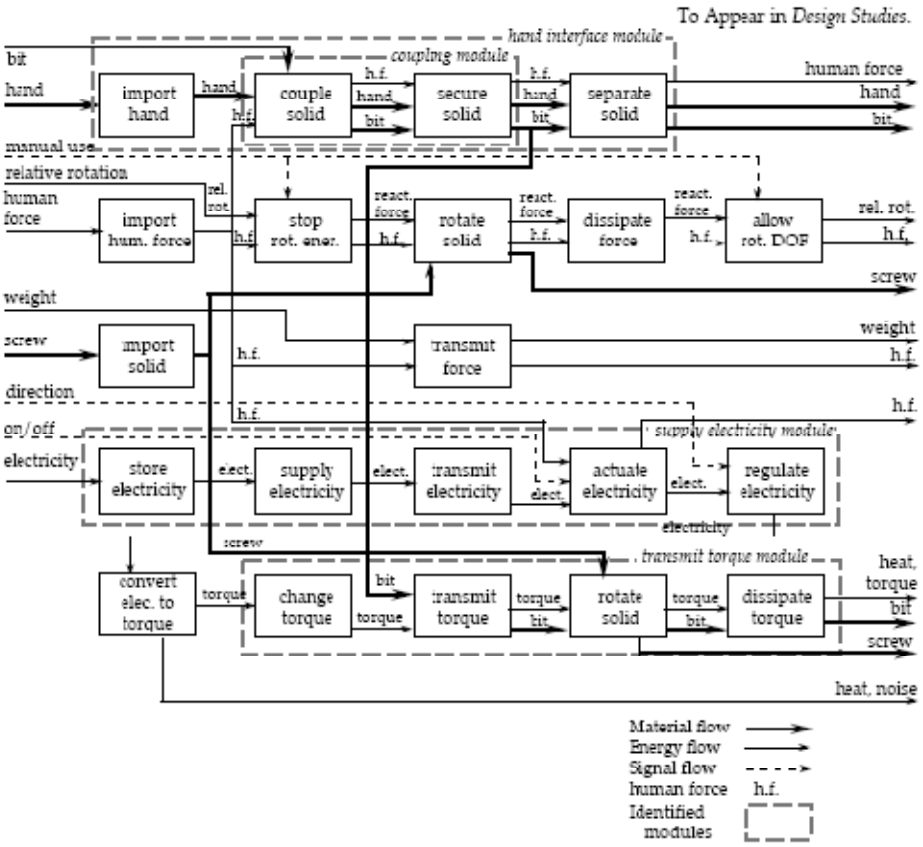

Figure 2.17. The functional model for an electric screwdriver (Stone et al, 2000)

By using the heuristics method it is up to the designer to choose which heuristics to use to identify modules and because it is not an algorithm and is rule based it tends to produce a high level of ambiguity during module formation. Furthermore, the method can be difficult to follow, especially when the product is complex. The production of the functional diagrams is quite complex and is a method that most designers will be unfamiliar with. Another primary weakness of the approach is that it only considers functionality and neglects strategic factors during module formation. Although there have been extensions to the method which do consider factors such as assembly of the modules (Dahmus et al, 2000) and commonality across product range (Gonzalez-Zugasti and Otto, 2000), these extended works still form the modules from a functionality perspective first and only after these modules have been identified do they attempt to improve the modules' characteristics according to strategic/lifecycle aims.

2.6. Decomposition based Modular Design Methods

The next sub-sections will look at the methods that have been developed primarily for the modular design of new products, where there may be radical new product structures or where the designer is interested in designing a whole range of products that share functions.

Ulrich and Tung's Decomposition Approach

The seminal works of Ulrich and Tung recommend the use of a function to physical alignment to create modular structures. Although they do not provide much guidance for how this is actually achieved, they do define the various types of modular product architectures that can be created.

These different types of modularity are based upon how the product is configured for product variety. They define four different types: *Component swapping modularity:* in which product variance is obtained by swapping one of more components on a common product platform. *Fabricate to fit modularity:* product variants are obtained by changing a design variable of a module. By scaling a module up or down the module can be altered before it is combined with other modules. *Bus modularity:* uses a standard structure (or base) on to which any number of different components can be attached to create variety; *Sectional modularity:* any number of product configurations can be obtained by combining components in an arbitrary way providing they are connected via their interfaces.

The later work of Urich (1995) defines the three types of modularity which are similar to his previous work. However, they are now defined by the nature of interfaces between components. These three types can be seen in figure 2.18. In *slot modularity* the interfaces between components are different so they cannot be combined in an arbitrary way. For example the car radio has a unique interface so cannot be combined with other components in the car's dashboard as they have different interfaces. For *bus modularity* there is a common component (bus) that

other components connect to via a standard interface type. A type of bus modularity can been in personal computers where the motherboard is the bus and the expansion cards are connected via standardised interfaces to create variety. In *sectional modularity* there is no base component that all components are attached to, the product can be built by connecting any number of components via their identical interfaces as for example in sofa or kitchen units.

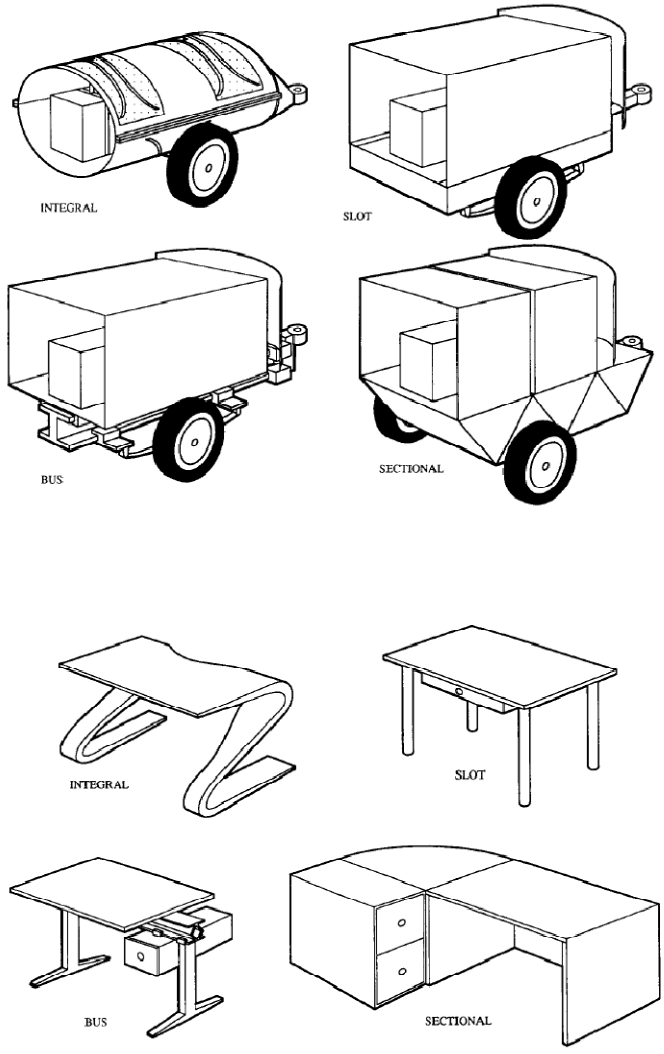

Figure 2.18. The four types of product architecture according to Ulrich (1995)

Pahl and Beitz's Decomposition Method

According to Pahl and Bietz (1984), function should be aligned to form. They discuss the process of modular design as primarily a means to create product variety with various different types of functional modules. They prescribe that the modular product should be built based upon four types of modules: *basic modules:* these modules implement functions that are common across the product family; *auxiliary modules:* are modules that are used in conjunction with the basic modules to create product variety; *adaptive modules:* are modules which are adaptable to changing customer needs/ system constraints; *Special modules:* are modules that implement specific customer needs which may not appear in all product variants.

Axiomatic Design

Although not actually claiming to be a modular design method, Suh's (1995) axiomatic design method does present a means of creating a well-defined product architecture. It provides a structured method of aligning the various domains of the product design. The primary goal of axiomatic design is to create designs which are better aligned to the customer needs and to decouple the complex relationships between the various domains. Thus axiomatic design is sometimes referred to as a domain theory based design methodology. This is somewhat similar to the ideas presented earlier in which function is aligned to form in order to reduce re-design complexities etc.

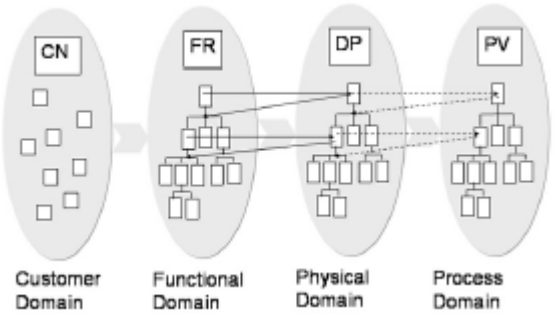

Figure 2.19. The four domains of axiomatic design (Suh, 1995)

As can be seen in figure 2.19 the axiomatic design process zigzags between four domains: Customer, Functional, Physical and Process. The process begins by converting customer's needs (CNs) into Functional Requirements (FRs) which are then embodied into Design Parameters (DPs). DPs then determine (and are affected by) the manufacturing or Process Variables (PVs).

There are two axioms used in axiomatic design: Axiom 1: to maintain the independence of the functional requirements. Axiom 2: to minimise the information content of the design. Axiom 1 requires that the Functional Requirements (FRs) be independent of one another in order to create decoupled and simple designs. Axiom 2 is often used as a quantitative measure of a design solutions, that can be used to select the best designs which satisfy axiom one (Suh, 2001). Generally, the design that uses the least information is superior.

2.7. Step-wise Redesign Methods

A number of modularity methods exist that are primarily aimed at the analysis and redesign of existing products. Generally the methods assume that the functionally of the product is clearly known and the physical solutions are defined.

Modularity Evaluation Charts

Ishii (1998) presents a modularity method with the primary aim of enhancing life-cycle modularity of products. For this purpose he has developed a number of evaluation charts for analysis of the product's sub-assemblies and their associated modularity. The charts are aimed at helping designers in analysis of existing product modularity and assist in the grouping of subassemblies by identifying 1) core platforms, 2) flexible modules and 3) mating interfaces.

Ishii presents four primary categories of evaluation charts: Design flexibility, manufacturing complexity, serviceability and recycling. The modularity evaluation

for manufacturing plots part commonality against lead time. Service modularity gauges service complexity versus frequency. A recyclability chart plots sort complexity against material recovery. An example chart can be seen in figure 2.20 which presents an example for the serviceability of an inkjet printer.

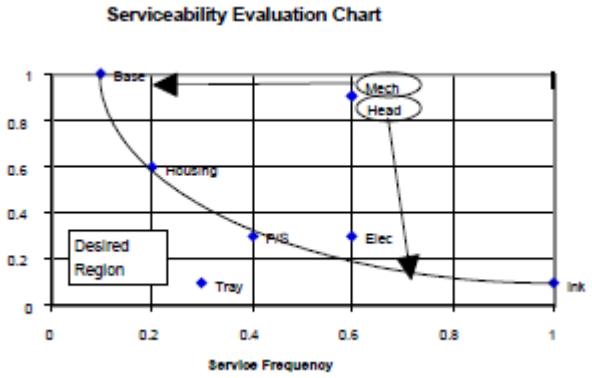

Figure 2.20. Module evaluation chart for serviceability (Ishii, 1998)

The method is useful for analysis and redesign of existing products, but does however assume that the product has already been defined at the sub-assembly level and information exists on assembly and disassembly sequences.

Environmental Modularity Analysis

Qian (2003) proposes a quantitative environmental analysis methodology for modular design. The approach considers all stages of the product life cycle and a number of environmental focused objectives are used as criteria for the modularity analysis. The method presumes that the functional structure of the product is defined and that the modules are in fact the product's sub-assemblies. A graph (matrix) based approach is used to represent various interaction relationships between the product's components. For the analysis, the weight of each environmental criterion is calculated using a fuzzy Analytical Hierarchical Process and relationships between components are measured according to each criterion. In their research the environmental modularity analysis consists of two parts:

- Similarity Analysis. The similarity/compatibility of the components within the same modules are analysed, and components incompatible with others are identified. Based on analysis results, suggestions are provided to improve the similarity/compatibility of modules.
- Independence Analysis. Dependencies of components between the components in other modules are analysed, and components with strong dependencies are identified. Based on the analysis results, suggestions are then made to improve the independence of modules.

Design for Variety

Martin and Ishii (2000) proposed the Design for Variety (DFV) method which is a quality function deployment (QFD) based approach for improving the robustness of a product to meet future design changes. They claim that by understanding the drivers for change the components or modules that are influenced by change can be identified. This enables a greater understanding of future redesign efforts and allows a common product structure to be levered for future product generations. Unlike the majority of previously reviewed methods the focus of DFV is not to create an optimal product architecture by the grouping of components into modules. The focus of DVF is to create a more decoupled product architecture (reduce the coupling between components and modules) and to ensure that the modules are more robust in regards to future design changes (changing customer needs). This is primarily achieved though the redesigning of component/ module attributes.

At the core of the DFV method are two simple metrics. These metrics are used to guide the design team towards developing a more decoupled product architecture that requires less design effort for follow-on products. The first metric is the generational variety index (GVI), which is a measure for the amount of redesign effort required for future designs of the product. The second measure is the coupling index (CI), that measures the degree of coupling among the product components.

Once the GVI and CIs are calculated the DFV method moves on to look at ways of reducing the indices and improving the product's robustness to future design changes. This is done primarily through redesign of the components by either making the components more modular (only part of the component needs to be redesigned) or standardising (overdesigning) the component so that it can accommodate future design changes.

2.8. Discussion

Product modularity has been seen to be a complex issue with many definitions and exemplified advantages. To date there is still no definitive definition of what actually constitutes a modular product. However, within the engineering community the understanding of modularity seems to converge towards the seminal works of Ulrich and Tung (1991), who state that modules should carry out discrete functions and that each module should be decoupled from other modules. However we are told by some that we should take caution when following these works. Decomposing a product into discrete functional modules is not always possible or even necessary. And as has been seen during the review, function is only one of many other decomposition logics possible.

The majority of the reviewed methods choose to decompose the product into lower level elements and then group these elements into modules based upon 1) technical and/or 2) strategic based reasons.

The technical reasons are firmly based on Ulrich and Tung's principle of decoupled interfaces. That is to reduce the number of component dependencies between modules in a bid to create modules that are as independent as possible from one another. These dependencies are generally based upon the physical and functional (flow) interactions that occur between the components.

Strategic reasons are based upon similar lifecycle characteristics/ modular drivers, such as similar maintenance needs or similar materials for recycling. That is to

ensure that all components within the same module have similar strategic modularity needs.

Furthermore all the methods reviewed follow one of two principles for module grouping: module independence and module coherence. Module independence pursues loose coupling between modules (external interactions); and module coherence aims for a high similarity (internal interactions) of components within modules. This was also concluded by Gershenson et al. (2004) in their review of modularisation methods.

As regards to how the module optimisation (grouping) is actually preformed, one of three approaches is followed: manually (as in MFD), through the use of a clustering algorithm (DSM based methods) or through the use of a mathematical programming method (Gu and Sosale, 1999 and Nepal, 2005).

With the clustering algorithms, the component interaction matrix (DSM) is reordered so that component interactions are as close together as possible. Once the matrix is clustered the DM must then identify suitable modules, which can be a difficult task as module boundaries are often ambiguous. To aid this task a number of modularity measures have been proposed for assessing the modularity of potential component clusters. However another problem that also exists (as with most clustering algorithms) is the poor handing of constraints during cluster formation. Hence, the chosen module clusters have no guarantee of feasibility.

The mathematical programming methods all employ a search algorithm (such as a GA) to directly find optimal module groupings within the component interaction matrix (DSM). This is achieved by adjusting the component-to-module memberships until the given modularity measure is maximised. These methods can be considered superior to clustering algorithms for a number of reasons. Firstly, the DM does not have to manually partition a clustered matrix into suitable modules, as the algorithm finds optimal groupings directly. Secondly, they can handle multiple modularity objectives and explore ‘trade-off’ solutions. For example, by adjusting

the preference weights of the various objectives and rerunning the algorithm it becomes possible to explore alternative modular structures and examine 'what-if' scenarios.

However, there are problems with the existing mathematical programming methods. Firstly, all of the methods use some form of aggregated scalar objective function. That is all of the objectives are weighted (DM preferences) and added together to form a single optimisation objective function. This is not the ideal approach to multi-objective optimisation - as will be discussed in detail later in the thesis. One of the main problems is that the preference weights can be subjective, if certain objectives are under or over stated then the grouping result may favour a certain objective resulting in a sub-optimal modular configuration. A second problem is that performing a trade-off analysis between competing objectives can be tedious and laborious. Different solutions can be obtained by changing the weights, but the algorithm must be re-run each time to produce solutions. Also the DM will have no prior knowledge of the maximum attainment values of each objective (which is needed to normalise each of the objectives), so setting preference weights becomes even more problematic.

In addition to the problems with the grouping algorithms, the configuration methods do not appropriately deal with the complexities of product family design. They do not consider the fact that within a product family there may be certain functions that can be shared or performed by different sets of components. This somewhat limits the configuration approaches to modularisation of single products or parametric product families. That is a family of products that have exactly the same functional components, whereby variety is achieved by changing the parameters of certain components. An example would be a family of electric drills, that have different motor torques or different battery capacities. To better address the product family configuration, a better method needs to be found, by, for example, combining the concepts of axiomatic design (cross-domain mapping) with DSM based (inter-domain) matrix configuration. That is analysing the product family at the functional level and then mapping to the physical component level. In

this way a greater level of abstraction is possible, to enable better product family configuration.

There is another general issue with many of the developed modularisation methods (that consider multi-objectives) in that they require a considerable amount of product knowledge for them to be applied successfully. Often the evaluation guidelines that they include are vague and ambiguous. Secondly, they often require a lot of input effort to enter all of the component dependencies and similarities into the matrices. Consider a product with only 16 components and with only three modularity objectives (physical interactions, material compatibility and maintenance similarity) being analysed. The user will be required to quantify and enter some 384 interactions into the various matrices. What would be useful is a way to automate (semi-automate) this process, maybe based upon previous product versions or some kind of knowledge based system.

Lastly there is the issue of product redesign to improve modularity. Redesign methods are generally used to improve the modularity of an existing modular product architecture i.e. once modules have already been formed. Redesign suggestions include: elimination and integration of components or modules (much like DFA), reconfiguring components to other modules and changing the attributes of the components (such as material properties). Redesign methods are also used to complement the configuration and optimisation approaches. These methods provide redesign suggestions during the configuration process in a step-wise manner, to gradually move towards better modularity. The problem with performing redesign in this way is that by changing the attributes of certain components in order to improve modularity in regards to one objective, one may end up worsening the modularity in regards to another objective. The same applies to reconfiguring components to different modules or combining components/ modules. Following these approaches the DM may therefore end up in a ‘never-ending’ loop of redesign and reconfiguration. Indeed, there may in fact be certain components that it is not possible to redesign to improve modularity. For example, if the objective of product variety is being considered then certain components may have to be variants and

hence their attributes cannot be changed, and these components may have to be isolated into separate modules away from the common product platform. The best approach to product modularisation may therefore be to perform optimisation followed by redesign. That is to perform component grouping to form the 'best' modular architecture possible, after which a redesign step can be applied to further improve modularity.

2.9. Conclusions

It can be concluded that modularity is commonly seen as a means of controlling product complexity by decomposing the product system into smaller more manageable chunks. However, the logic of decomposition can be seen to vary considerably. Some researchers choose to base the decomposition on physical interactions between components, whilst others choose to base the decomposition on product function or any number of lifecycle drivers.

Two areas of consensus have arisen from the review: firstly, a module should have an element of independence (interactions between modules) and secondly a module should have an element of coherence (similarity of components within modules).

It has also been seen that numerous modularisation methods and frameworks have been created, often pursuing very similar goals. The vast majority of these methods are matrix based, using a design structure matrix approach to represent the complex functional, physical, and strategic based interactions that occur between components. Matrix representations are a suitably visual way of representing product modularity and, more importantly, can be readily manipulated with optimisation algorithms to identify modules. However no modularisation method has yet been created that cleanly and clearly spans the whole product lifecycle and provides a true multi objective optimisation. Some questions that need to be answered are:

- Is there a suitable way in which product function can be mapped to physical components such that optimal modules can be formed based upon the alignment of function to form?

- Can the ideas of technical modularity and strategic modularity be used to form a more integrated modular design framework that spans the whole product lifecycle? Furthermore, are there some modularisation objectives that are best evaluated in terms of module independence? Similarly are there objectives that are best measured in terms of module coherence?

- What are the most relevant modularisation objectives at various product lifecycle phases? And how can they be evaluated? Indeed are there any modularisation objectives that can be seen as conflicting or pursuing the same goals? Can the objectives be reconciled into a comprehensive set of objectives that span the whole product lifecycle?

- Is there a better algorithm that can be used/ developed to perform product modularisation for multiple objectives?

- Can the evaluation burden for quantifying component interactions be reduced in some way?

The research undertaken in the next chapters will address these questions to produce a more holistic modularisation method.

CHAPTER 3:

3. Review of Genetic Algorithms and Multi-Objective Optimisation

3.1. Introduction

Developing an algorithm for product modularisation is not a trivial task as there are often multiple, potentially conflicting modularisation objectives that must be simultaneously considered when grouping components to form modules. A promising method of handling such problems is to use a multi-objective genetic algorithm (MOGA) to generate, in one single run, a whole set of optimal solutions.

The purpose of this review chapter is thus to identify suitable multi-objective (GA based) approaches that could be used as a basis for the development of a suitable product modularisation model. This chapter begins with a technical review of GAs and then moves on to look at multi-objective optimisation and how GAs have been adapted and modified to tackle such multi-objective optimisation problems.

3.2. Overview of Genetic Algorithms

The product modularisation problem (component grouping) can be defined as a non-linear combinatory optimisation problem which could be solved in a number of ways, for example by using a traditional search method such as simulated annealing or goal programming. So the question is why use a GA based approach instead of a classical approach? The short answer is that it appears GAs work rather well.

GAs have proved themselves as a technique that consistently achieves good results (when compared to other techniques) having been applied to a vast range of optimisation problems such as engineering design parameter optimisation, fluid flow and structural optimisation and production planning and scheduling. Because GAs have, through extensive trials, been shown to be both reusable and robust

(Goldberg, 1989) it seems appropriate to use a suitably adapted GA for the product modularisation problem.

However, what is a GA and how does it work? A GA can be defined as a stochastic search technique based on the mechanisms of natural selection and natural evolution. There are many variations to the standard GA but they are all united by a common thread in that they all mimic the way biological evolution works.

Genetic algorithms became popular through the work of John Holland (1975). He designed a genetic algorithm which he described as an 'artificial system' based upon a 'natural system'. Since then an enormous amount of work has been done on the development of GAs, which can perhaps be accredited to their robustness and their ability to solve all manner of problems.

The mathematical terminology used to describe GAs comes directly from their biologically inspired roots. In a GA each possible solution to the optimisation problem is coded using a data structure known as a 'chromosome'. A chromosome is made up of a string of genes, each gene representing a specific input variable. Collectively the genes are used to evaluate the 'fitness' of an individual solution. An encoded chromosome is referred to as an 'individual' and a set of individuals in each 'generation' is known as a 'population'. Individuals from the population are selectively chosen for reproduction based upon their fitness (fitter individuals more likely to be selected). The selected chromosomes are then reproduced using the genetic operators of 'crossover' and 'mutation' to produce 'offspring' for the next generation. In this way the 'offspring' will inherent the good genetic traits from their parents, and, over a number of successive generations, the average fitness of the whole population improves and the GA can be expected to breed an optimal solution to the problem.

The standard GA can be summarised by the following steps:

Start

1. Create an initial population of chromosomes.
2. Evaluate the fitness of each of the chromosomes to select the best parents for reproduction.
3. Reproduction of parents
   -Crossover chromosomes to produce new offspring
   -Mutate the genes of the offspring chromosomes.
4. Evaluate fitness of each offspring chromosome.
5. Repeat steps 2 through 5 until some termination condition has been met

End

The population based approach gives the GA advantages over traditional search algorithms. GAs can find solutions to linear and nonlinear problems by maintaining a whole population of solutions which is used to simultaneously explore multiple regions of the solution space, exploiting promising areas through the genetic operations of cross-over and mutation. Traditional search algorithms on the other hand only search the solution space in one dimension, meaning that only one search direction is being explored at once. This means that if the search hits a local optimum or constraints are violated then the search may have to be abandoned and started again (Michalewicz et al, 1996).

The very fact that the GAs maintain a population of solutions also makes them particularly well suited to multi-objective optimisation problems. In multi-objective optimisation it is often impossible to simultaneously optimise all objectives, so inevitability trade-offs between objectives will need to be made. By using a suitability modified GA it becomes possible to maintain a population of diverse solutions that covers the approximated trade-off region in the objective space. In other words a population of solutions with each solution having different achievements of the various objectives. This approach will present the DM with a whole set of optimal ‘trade-off solutions’ that can be further examined to gain a deeper understanding of the problem. This will be looked at in more detail later in this chapter.

3.3. The Core Characteristics of Genetic Algorithms

The fundamental characteristics that affect the GA's performance are: chromosome representation (encoding), initialisation of the population, selection strategy, genetic operators, constraint handling and fitness function. In the following sub-sections, these characteristics will be reviewed.

Representation: Encoding

The first step in creating a successful GA is to create a suitable representation of the problem. For any given problem possible solutions may be represented as a set of parameters. These parameters must be mapped from the design space to the solution space. In other words the parameters of the optimisation problem must be given a suitable representation within the coding of the GA. This is known as encoding of the problem. Within the GA, the encoded design parameters are known as genes and are joined together to form a chromosome. The way in which a chromosome is encoded will vary and is generally problem specific. In some cases the encoding may be fairly simple, whilst in other cases, such as 3D object optimisation, the encoding may be complex.

The design of the encoding scheme is often seen as an important part of the GA implementation strategy as it provides the crucial link between the real world and the GA solution space. It must therefore be as simple as possible. A common approach is to code the design parameters into a string of binary numbers.

Population Size and Initialisation

The size of the population is important because it influences whether the GA can find good solutions as well as the time it takes to reach them. If the population is too small there might not be an adequate supply of genes and it will be difficult to

identify good solutions. If the population is too big the GA will waste computational resources processing unnecessary individuals.

Initialisation of the population is also another important part of the evolutionary strategy. If the GA starts the search with a poor initial population then the time taken to converge on an optimal solution may be adversely effected. Hence, in some cases, if sufficient domain knowledge is known about the problem, initial populations can be created using a suitable heuristic, or the search can be started from promising areas by 'seeding' the initial population. On the other hand a randomly generated initial population can introduce a greater level of potential diversity and the GA is more free to create novel and unconventional solutions.

Selection

Choosing parents for reproduction of new chromosomes (offspring) from the population is called selection. Selection can be based on many different criteria but it is usually based on the fitness values of the chromosomes. The idea behind this is to select the best chromosomes for parents in the hope that combining them will produce better offspring chromosomes. But selecting only the best chromosomes has one major disadvantage as all chromosomes in the population will start to look the same very quickly. This narrows exploration space, pushes the search into regions of local optima and prevents the genetic algorithm finding possibly better solutions that reside in unexplored areas of the search space. To preserve diversity of chromosomes within the population, selection operations usually introduce a factor of randomness in the selection process. Although there are many types of selection operators used in GAs, the two most commonly used are roulette wheel selection and tournament based selection.

Roulette Wheel Selection

For this technique the slots on the wheel are created based upon the fitness of the individuals in the population. The size of each slot is directly proportional to the corresponding individual's fitness. (i.e. like a pie chart of all the fitness values).

Following this principle, each solution will ultimately have a numerical range associated with it. Selection then begins with a randomly generated number (i.e. the wheel is spun) and the solution whose range the value lies within is selected. Hence higher ranking individuals have a greater likelihood of being selected. However, the roulette wheel method of 'proportionally allocated' fitness can lead to premature convergence of the solution space. Proportionally allocated fitness can also give rise to population stagnation problems towards the end of the search as individuals that are far better than their neighbours may not be given a large enough slice of the wheel.

Tournament Selection

In tournament selection, as the name suggests, a "tournament" is set up by randomly selecting a number of individuals from the population and selecting the winner. The winner is always the solution with the highest fitness. The chances of weaker (less fit) individuals being chosen can easily be controlled by changing the tournament size. i.e. there are less likely to be stronger individuals present in the tournament.

Elitism

The basic GA will simply create a new population by crossover and mutation. The new population then replacing the old population at each generation. With this approach there is however a chance of losing the best chromosome. To overcome this problem an elitist operator can be used, which works by modifying the population replacement procedure. At the beginning of the selection process the best individual or a number of best individuals from the old population are first copied to the new population and selection then proceeds as normal. Elitism can drastically improve the performance of a GA because it prevents the loss of the best found solutions.

Genetic Operators

The basic notion of a GA is that it will breed an optimal solution through the mimicry of the biological reproductive process. This is achieved though the genetic operators of crossover (mating) and mutation. Crossover and mutation form the basic mechanisms of exploration in the search space of the GA. In fact the GA would have little chance of performing better than the other meta-heuristics (simulated annealing, tabu search etc.) if it were not for the crossover operator.

Crossover

Crossover is always used in conjunction with a suitable selection operator (roulette wheel, tournament, etc.) to find suitable parents. Firstly, two fit individuals (parents) are selected from the population using the chosen selection operator. Once two parents have been selected their chromosomes are recombined by using the mechanisms of crossover. Crossover is not applied to all parents selected for mating which gives each individual a chance of passing on its genes without the disruption of crossover.

An example crossover operation can be seen in figure 3.1. Firstly the two parents are selected and their chromosome strings are split at some randomly chosen position to produce two head segments and two tail segments. The segments are then swapped over to produce two new full length chromosomes.

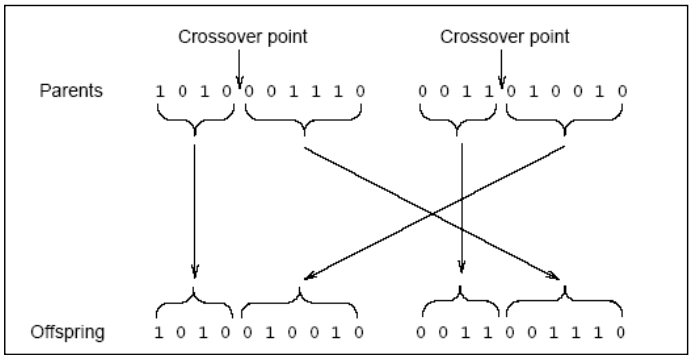

*Figure 3.1 Crossover mechanism of GA (Beasley et al, 1993***)**

Mutation

Mutation provides a small amount of randomness into the search space and helps ensure all gene combinations are considered. This is also a key component of the GA's diversity preservation strategy. Without mutation there is a chance that some of the mated solutions (offspring) would end up being very similar, if not identical, to their progeny particularly towards the end of the search when solutions begin to look similar. Mutation is generally randomly applied (i.e. not to all chromosomes) to a chromosome, straight after crossover on the newly generated child. Mutation usually involves only small changes to a small number of a chromosome's genes and the degree of mutation can be adjusted via a mutation rate parameter. (See Figure 3.2)

Figure 3.2. Mutation mechanism of GA (Beasley et al, 1993)

Genetic Operators for Grouping Problems

Product modularisation can be described as a grouping problem, as we are trying to group a predefined number of components into a number of modules in a manner that will maximise modularity according to a given objective function. However traditional GAs are poorly equipped to deal with grouping problems. This is mainly due to the traditional GA's inability to preserve the integrity of groups during mating and pass on useful genes to the next generation (Falkenauer 1998). To overcome these problems Falkenauer (1998) proposed a modified GA especially for grouping problems - the grouping genetic algorithm (GGA). The GGA uses a different encoding scheme and genetic operators, the important point being that these genetic operators will work with the group part of the chromosomes, the

standard item part of the chromosomes merely serving to identify which items actually form which group.

Constraint handling

Many real world engineering optimisation problems will more than likely involve any number of constraints. This will often severely limit the number of feasible solutions possible for the problem and so the set of feasible solutions can end up being extremely small compared to the total solution space. For highly constrained solution spaces the population initialisation and genetic operators must be carefully designed as any random generation of a solution will be a hopeless task, inevitably leading to many infeasible solutions. To handle constrained problems the GA has first to check whether the candidate solution is feasible. If not feasible then solutions need to be dealt with in an appropriate manner so that they do not pass on their 'disruptive genes' to future generations. It is argued that for the modular design problem there will be a number of constraints: the number of modules must not exceed a maximum number, the number of components in each module must not exceed a certain level, certain components most never be placed with other components for geometric or functional reasons and there may also be assembly precedence requirements. There are a number of ways in which the problem of constraints can be overcome. The possible options include:

1) Discarding infeasible solutions (the 'death penalty'). Using this approach the GA may take substantially longer to evolve solutions, since infeasible ones are often discarded. The GA may end up getting trapped in an endless loop of creating and discarding infeasible solutions.
2) Reducing the fitness of infeasible solutions by using a penalty function. This is probably one of the most widely used methods due to its simple application and effectiveness. The idea is to allocate to infeasible solutions a penalty that reduces the solution's overall fitness and reduces the likelihood of selection. If the infeasible solutions are mated then there is a chance that

the bad parts of the parent solutions may be swapped out to provide feasible child solutions.

3) Crafting genetic operators to always produce feasible solutions. Heuristics can be developed so that infeasible solutions are never allowed to be generated in the first place.
4) Transforming infeasible solutions to feasible solutions i.e. to 'repair' infeasible solutions. By adjusting one or two of the individual's genes an infeasible solution can often be converted into a feasible one.

Fitness Function

In order to find better solutions a fitness function is needed for evaluating and selecting good chromosomes. The fitness function gives domain specific information about the value of each chromosome. For this reason, it is usually a good idea to define it in the form of a mathematical formulation, either a maximisation or a minimisation of some parameters and constraints.

For a complex optimisation problem, such as the modularisation problem, the fitness function may be a combination of several objectives, such as component coupling, variety, outsourcing, recycling, etc. This is defined as a multi-objective design problem and specialised GAs have been developed for this process. MOGAs often use a very different method of fitness assignment to their single objective cousins. Instead of maximising a single mathematical fitness function they often use comparative based methods of fitness assignment, where each solution in the population is compared to each other in order to decide how superior a particular solution is in terms of its dominance (improvement) over other solutions and how diverse (different) the solution is compared to its neighbours. These principles will be looked at in more detail in the next sub-section.

3.4. Overview of Multi-Objective Optimisation

Solving a single objective optimisation problem is generally quite straightforward and is usually a case of either minimising or maximising the objective function in order to find an optimal solution. However, when solving most real world problems there is often a need to optimise more than just one objective. For example, the product modularisation problem is in fact comprised of many different objectives. For such multi-objective problems it is often impossible to find a solution that maximises all objectives at the same time, as some of the objectives will be in conflict and hence competing against each other. The maximisation of one objective may have a detrimental effect on some of the other objectives. These competing objectives will almost certainly give rise to a potentially vast set of suitable compromise solutions. That is a set of solutions with each solution having different combinations of objective achievements. Any one of these compromise solutions could be chosen as a suitable solution to the problem.

However, finding such compromise solutions using exact methods can be a difficult task especially when the search space is large and complex. A suitable method is to use an efficient search algorithm (such as a GA) to generate a number of solutions that are considered ‘optimal compromises’ and then let the DM examine the trade-offs between them. The DM will then choose the most suitable solution using this trade-off information.

The set of optimal solutions that are found to examine the trade-offs between objectives is commonly referred to as the Pareto-optimal set. The concept of Pareto optimum was formulated by Vilfredo Pareto in 1896 and established the foundations of all subsequent research in multiple objective optimisation.

A solution is Pareto optimal when we cannot improve any objective further without at the same time worsening another. When all such solutions are found then we have what is called the 'Pareto-optimal set' or 'Pareto front' (of objective vectors). In such a case all the other inferior solutions are said to be ‘dominated’ by the Pareto-

optimal (non-dominated) solutions and we can discard them. For example, in figure 3.3 the solution ‘c’ dominates the solutions ‘g, j, k’. Solution ‘c’ however does not dominate solution ‘b’ but is an alternative trade off between objectives. The set of these ‘cannot do better’ trade-off solutions is often referred to as the non-dominated set. The non-dominated set hence contains all the Pareto-optimal solutions found during the search, each solution of the set having different combinations of the objectives or ‘niches’ within the solution space.

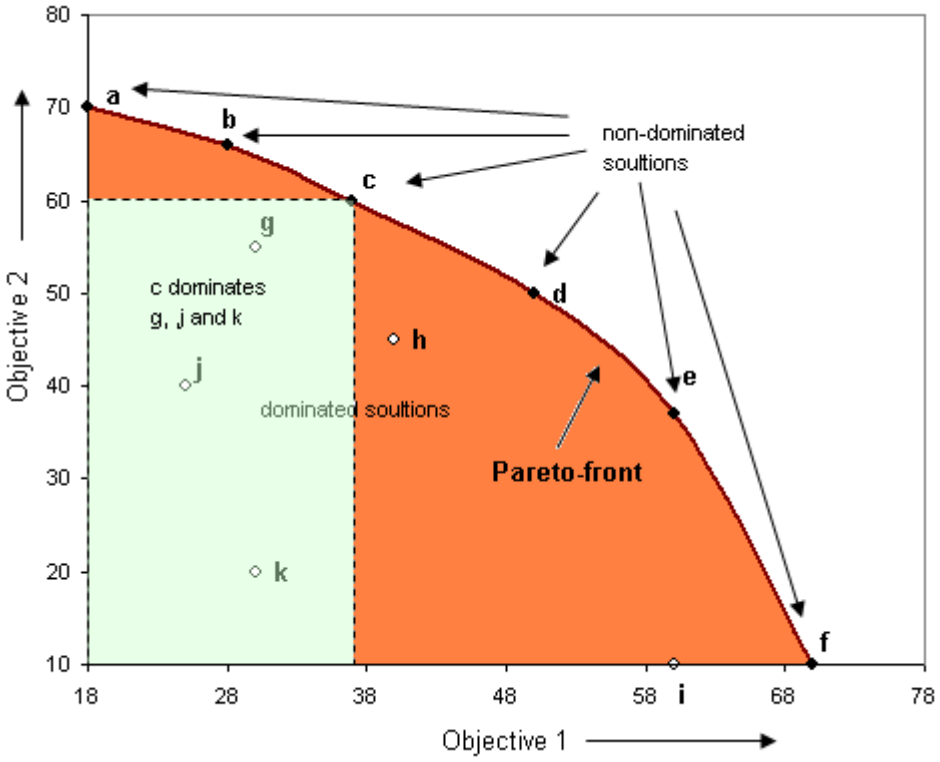

Figure 3.3. The Pareto-optimal front and dominance relationships

There are two main approaches that can be followed to establish the Pareto-optimal set (non-dominated set). The first approach is to aggregate the objectives to form a single objective function. By varying the preferences approximate solutions that lie on the Pareto-front can be found. The second approach is to generate a whole set of non-dominated solutions in one single optimisation run using a suitable modified GA (Multi-objective GA).

3.5. Aggregate Objective Based Multi-Objective Optimisation

The most simple approach to multi-objective optimisation is to aggregate all the various objectives to form a single scalar fitness function, which can then be handled by a suitable single objective optimisation technique, such as a standard GA, simulated annealing etc. This approach to multi-objective optimisation is commonly referred to as the aggregate approach. Different preference co-efficients can be used for the scalar function to find different solutions on the Pareto-front. This aggregate approach is the approach all previous modularity researchers have followed to obtain 'optimal' product modularisations. Two common methods to the aggregate approach are the weighted sum and goal attainment.

Weighted Sum

The most common aggregate approach is to use weighting coefficients to estimate the relative importance of each objective and combine them to form a scalar objective function. An 'optimal' solution can then be obtained based upon the maximisation or minimisation of the scalar objective function.

Goal Attainment

The fundamental concept of goal obtainment is to set target goals for each objective and to then minimise the deviations from the target goals. There are two common approaches to goal obtainment: weighted and Chebyshev (min-max) goal obtainment. Weighted or pre-emptive goal obtainment attempts to minimise all unwanted deviations from goals. Deviations from goals are multiplied by weights, reflecting their relative importance. Chebyshev or min-max goal obtainment seeks to minimise the maximum unwanted deviation, rather than the sum of deviations. This emphasises justice and balance rather than ruthless optimisation and is favoured by DMs interested in obtaining a balance between the competing objectives.

Problems with Aggregate Approach

Generally, these aggregate approaches are very simple and easy to implement. By varying the target goals or goal weights it is possible to generate a set of non-dominated solutions. A problem that arises however is how to normalise, prioritise and weight the contributions of the various objectives, especially if the problem is poorly defined and the DM does not have any clear priorities with regard to the various objectives. Indeed, it is sometimes difficult if not impossible to find a suitable approximation of the Pareto-front using an aggregate approach, especially when the Pareto-front is non-convex or discontinuous. Furthermore, the aggregate methods will only ever provide one single optimisation solution at a time. To explore trade-offs the algorithm may need to be rerun many times to obtain a suitable set of Pareto-optimal solutions. This can be tedious and time consuming.

3.6. Multi-Objective Genetic Algorithms (MOGAs)

GAs are population based, which makes them particularly well suited to solve multi-objective optimisation problems. The ability of GAs to simultaneously explore different regions of a solution space makes it possible to find a diverse set of solutions for a difficult to solve problem. A generic single objective GA can be readily modified to find a whole set of Pareto-optimal solutions in a single run.

Overview of Common MOGAs

A vast array of MOGA approaches have been developed over the years, and a detailed review of all of these algorithms is considered outside the scope of this thesis. The following sub-sections provide a condensed review, looking at some of the most popular and widely used MOGAs.

Vector Evaluated Genetic Algorithm (VEGA)

The earliest MOGAs used the concept of fitness sharing to find approximate sets of solutions in the Pareto-front. VEGA developed by Schaffer (1985) was the first GA used to approximate the Pareto-optimal set by finding a set of non-dominated solutions. In VEGA the approach is to perform proportional (roulette) wheel selection using each objective to select a number of sub-populations. That is to divide the population into k sub-groups based upon the performance of each individual to one of the k objectives. These populations are then shuffled together to form a new population. Crossover and mutation then proceeds on the new population in the same way as for a single objective GA. This is very simple and efficient, but solutions generated tend to be locally non-dominated and not necessarily globally non-dominated. The method also tends to produce solutions that excel only along one objective and the population therefore contains few compromise solutions. Schaffer suggested applying fitness penalties to locally dominated points and redistributing the deducted finesses to non-dominated ones. This caused a premature convergence because in populations with few non-dominated points these points were given large fitness values.

Weighted sum Based Genetic Algorithm (WBGA)

Hajela and Lin (1992) proposed the WBGA for multi-objective optimisation. Their goal was to be able to simultaneously generate, in a single run of the GA, a set of Pareto-optimal solutions corresponding to different weight vectors. The different weight vectors are embedded within the chromosomes of each solution in the population. The method also uses a vector evaluated approach based on ‘VEGA’ to perform selection. Consequently, the method evolves solutions and weight combinations simultaneously to help maintain diversity.

The Multiple Objective Genetic Algorithm (MOGA)

Fonseca and Fleming (1995) use a Pareto-based ranking procedure for their MOGA. The Pareto-ranking approach explicitly utilises the concept of Pareto dominance in evaluating fitness or assigning selection probability to solutions. Instead of ranking the population based upon a conventional objective function the population is ranked according to a dominance rule with each solution being assigned a fitness value based upon its dominance rank in the population. In Fonseca and Fleming's MOGA the rank of an individual is proportional to the number of solutions found in the population that it is dominated by. The fitness assignment is determined by interpolating the fitness value of the best individual (non-dominated) and the worst one (most dominated). The MOGA algorithm also uses a niche-formation method to distribute non-dominated members of the population over the Pareto-optimal region.

Non-Dominated Sorting Genetic Algorithms (NSGA and NSGA II)

Srinivas and Deb (1994) proposed a Pareto-ranking approach entitled the non-dominated sorting genetic algorithm (NSGA). NSGA uses a sorting procedure to sort the population into different Pareto-fronts based upon level of non-dominance. After sorting, tournament selection is used to choose individuals for mating. From the tournament, the two individuals with the lowest front numbers (highest fitness) are selected and genetic operators are then applied to create a new population. Using this procedure a quick convergence of the population toward non-dominated regions is achieved.

In a further extension to the NSGA, the NSGA II was developed by Deb et al (2002). Several new concepts were introduced to make the algorithm more effective. Firstly, the sorting mechanism was modified to increase its speed. Secondly, the concept of elitism was used in order to preserve the 'best' solutions from previous generations. And lastly a crowding distance was introduced in order to help maintain diversity within the population and encourage a uniform

distribution of solutions over the Pareto-front. The NSGA solutions that reside in the same Pareto-front are assigned the same fitness regardless of whether or not certain solutions are located within sparsely populated regions. The crowding distance assigns a higher fitness to individuals located within sparsely populated regions of the same Pareto-front to better promote diverse solutions. Using these improvements the new NSGA II is significantly better than its predecessor and to this date it remains one of the most widely used MOGAs, both within the research community and within industry.

Strength Pareto Evolutionary Algorithm (SPEA and SPEA2)

Ziztler and Thiele (1999) introduced an evolutionary approach to multi-criteria optimisation called the Strength Pareto Evolutionary Algorithm (SPEA). The method combines several features of previous multiple objective GAs in a unique manner that ensures that the population moves towards the Pareto-front and that population diversity is preserved. The fitness assignment scheme of SPEA is based upon each individual being assigned a fitness calculated by the number of non-dominated points (in the external population) that dominate it. Like other MOGAs, the method uses an elitist strategy to store non-dominated solutions in an external and continuously updated population. The external population update procedure proceeds as follows. First, all non-dominated population members (from the current population) are copied to the external archive population and any dominated individuals or duplicates are removed from the archive. If the size of the external archive exceeds a predefined limit the method performs a clustering procedure to reduce the number of individuals without destroying the distribution characteristics of the Pareto-front.

An improved version of the SPEA, the SPEA2 was developed by Ziztler et al (2001). SPEA2 has three main differences in respect to its predecessor: i) it incorporates an improved fitness assignment scheme, which for each individual takes into account how many individuals it dominates and how many it is dominated by; ii) it uses a nearest neighbour density estimation technique that

allows a more precise guidance of the search process and helps to maintain diversity in the population and iii) an improved external archive update method guarantees the preservation of boundary (extremal) solutions.

The Pareto Archived Evolution Strategy (PAES)

The Pareto Archived Evolution Strategy (PAES) was created by Knowles and Corne (2000). They argue that PAES may be the simplest algorithm capable of generating diverse solutions in the Pareto optimal set. What makes PAES unique is that it maintains only one population of solutions (the non-dominated solution archive) and does not use a mating operator, using only mutation to form new solutions. PAES comprises of three parts: the candidate solution generator, the candidate solution acceptance function and the non-dominated solutions archive. The candidate solution generator is akin to a local search procedure and uses a single current solution that is randomly mutated at each iteration to produce a single new candidate solution. Like other algorithms the candidate solution acceptance function utilises the Pareto-dominance concept and a solution diversity rank. If the new solution dominates the current solution (its parent) then it is accepted into the archived population. If however the new solution is non-dominated it is compared to the other solutions stored in the archived population and accepted into the archive only if it resides in a less crowded region.

3.7. Commonly Used MOGA Strategies

From the overview of MOGAs it has become apparent that the majority of the ‘state of the art’ MOGAs use the concepts of Pareto-dominance based ranking, some form of diversity preservation strategy and an elitist achieving strategy for storing the best found non-dominated solutions. In the next sub-sections these concepts will be examined in more detail.

Pareto Dominance Based ranking

The concept of Pareto-dominance based ranking is to assign a fitness to each member of the population based upon its level of dominance (or non-dominance) compared to all other solutions in the population. For example, if a solution x has no improvement over solution y in *any* of the objectives then it is said to be dominated by solution y. Obviously solution x may be dominated by more than one solution, the same goes for other solutions that may be dominated. For this reason it is appropriate to assign dominated solutions different dominance ranking (fitness), as inevitably some solutions will be dominated to a larger extent than others. An appropriate dominance measure must therefore distinguish between the levels of dominance present in the current population. This is important because the algorithm can then gradually replace largely dominated members of the population with less dominated (better) solutions when they are found, ensuring that the population creeps towards the Pareto-front in an efficient manner.

According to Zitzler et al (2003) there are three common ways of ranking dominated solutions:

a) Dominance depth - at which Pareto-front is an individual located.
b) Dominance rank - by how many individuals is an individual dominated.
c) Dominance count - how many individuals does an individual dominate.

The dominance depth method as used by NSGA-II employs the Pareto dominance ranking approach proposed by Goldberg (1989) to sort the population into different non-dominated fronts, as seen in figure 3.4. The first front can be seen as the current (best) Pareto-front and receives the lowest (best) rank of 1. Subsequent fronts receive a proportionally higher (worse) rank relative to the level they are dominated. As many of the population members will inevitably have the same rank, a further crowding distance based diversity rank is used to distinguish between solutions of the same dominance rank.

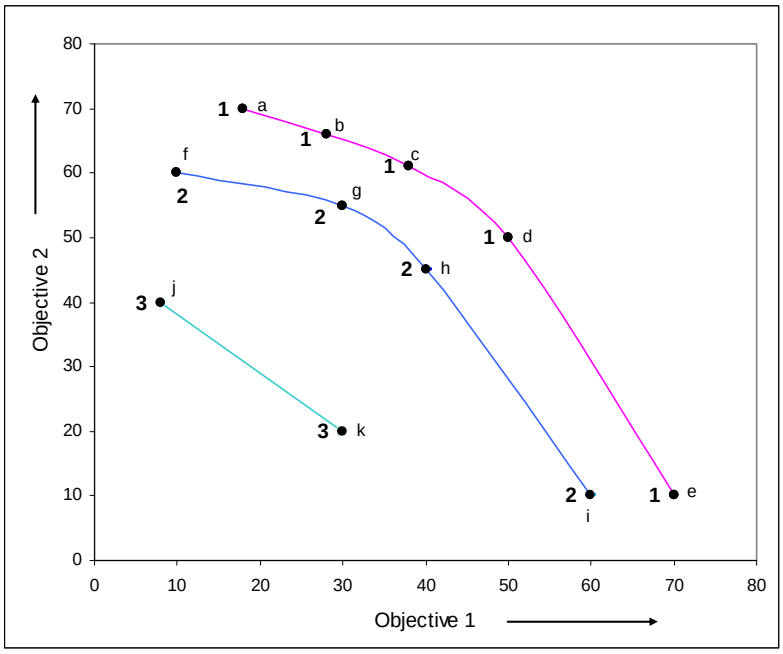

Figure 3.4 Dominance depth based Pareto-dominance ranking (NSGA-II)

Fonseca and Fleming (1995) use the dominance rank assignment method which is based upon the number of solutions a particular solution is dominated by. With this principle the level of dominance is calculated by counting the total number of solutions that a particular solution is dominated by. If a solution is only dominated by a small number of other solutions it will receive a better fitness than a solution that is dominated by a large number of solutions. In this way the ranking method penalises solutions located in densely populated regions of the solution space and provides a crude form of diversity preservation. For example, in figure 3.5 solution 'f' is dominated by solutions 'a, b and c'. Therefore, it is assigned a rank of 4 although it is in the same front with solutions 'g, h and i' according to the previous ranking method.

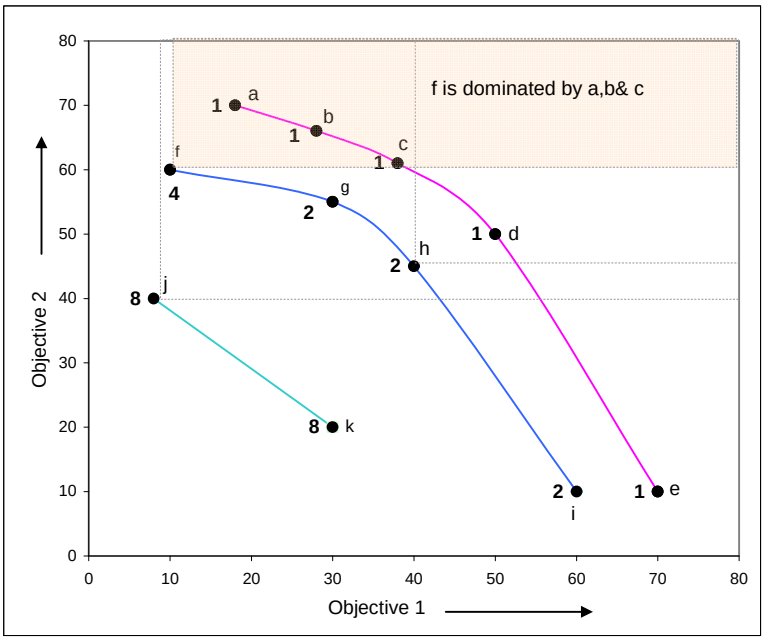

Figure 3.5 Dominance rank based Pareto-dominance ranking (MOGA)

In SPEA2 a combined dominance count and dominance ranking method is used. Dominance is calculated based upon the sum of the number of solutions a particular solution 'x' is dominated by and the number of solutions that the solution 'x' dominates. Firstly, all solutions are assigned a rank based upon the number of solutions that they dominate e.g. in figure 3.6 solution 'g' dominates two solutions so receives a rank of 2. All non-dominated solutions will receive an arbitrary rank of zero. Next, if a solution is dominated by another solution then the solution's dominating rank is added to the solution own rank. e.g in figure 3.6 solution 'k' is dominated by solution 'g' which has a dominance rank of 3 so the total dominance based rank for solution 'k' is 8.

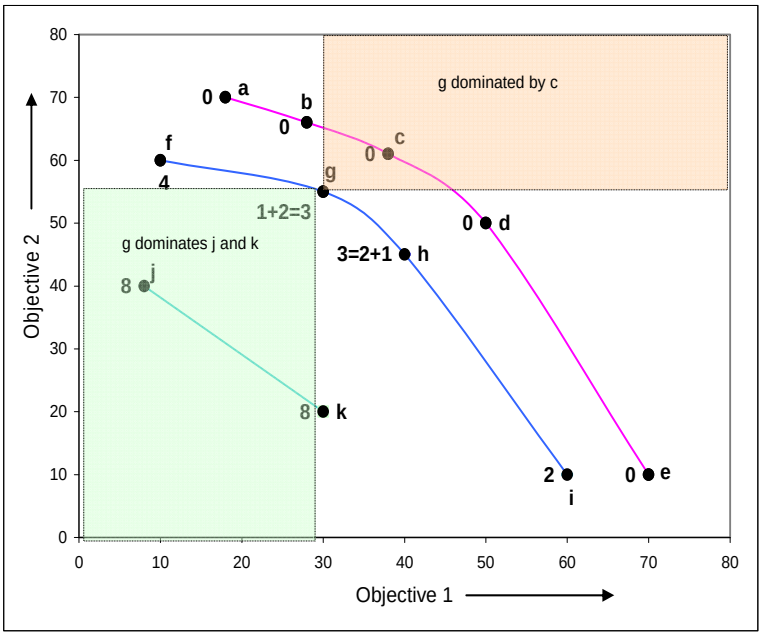

Figure 3.6 Dominance rank and count based Pareto-dominance ranking (SPEA2)

Diversity Ranking

As the MOGA search progresses, most, if not all of the solutions in the population are likely to become locally non-dominated, thus the MOGA needs a means of assessing the fitness of these non-dominated solutions during selection and population updating. This should be done in a manner that ensures that the algorithm is moving the population towards the Pareto-front as well as promoting the diversity of solutions along the Pareto-front, when all the solutions become non-dominated. A diverse population will ensure that the DM is presented with a wide range of different solutions so that compromises between objectives can be explored.

Most of the commonly used algorithms such as NSGA-II, PAES, SPEA-2, use an external 'elite' population or archive to store the 'best' non-dominated solutions found so far during the search. The archive is constantly updated with better solutions at each generation. When all of the solutions in the archive become non-dominated the better solutions are chosen as the solutions that lie in less crowded/ densely populated regions of the search space. To do this all of the algorithms

include a method that directly measures the level of crowding or density among solutions of the population. At present there are two main diversity measures, cell density based measures and Euclidean distance based crowding measures.

The NSGA-II uses a Euclidean distance based measure named the crowding distance. The crowding distance measure is a nearest neighbour density estimator used to ensure that solutions in less crowded regions of the solution space are favoured and maintained in the population. To obtain the crowding distance the distance between the two nearest points on either side of a particular solution is calculated for each objective. This is achieved by sorting the solutions into descending order according to the first objective and then for each solution calculating the distance between the neighbouring points on either side the objective space. Solutions that represent the extremes for each objective are given a crowding value of infinity to help ensure that they remain in the population. This process is repeated for all other objectives until all neighbouring objective distances have been calculated. The total crowding distance is then the sum of all of these distances. One of the main strengths of the approach is that no user-defined parameter is required. In figure 3.7 the crowding distance can be seen for solution 'c'.

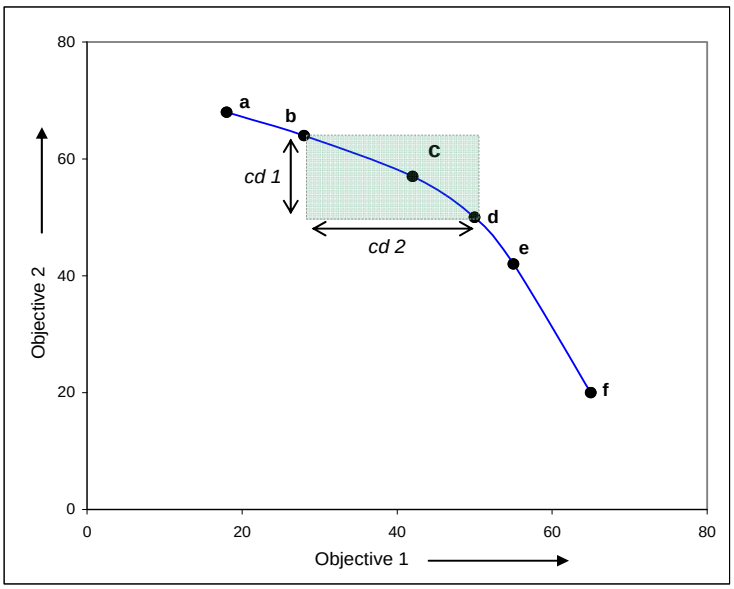

Figure 3.7. Crowding distance based diversity ranking

In a similar way to the NSGA-II the SPEA2 uses a crowding distance type approach for diversity ranking. The distances between K nearest neighbours (K-NN) for each solution are calculated and the solutions residing in the most crowded regions are given a proportionally lower fitness than solutions in less crowded regions. This can be seen in figure 3.8.

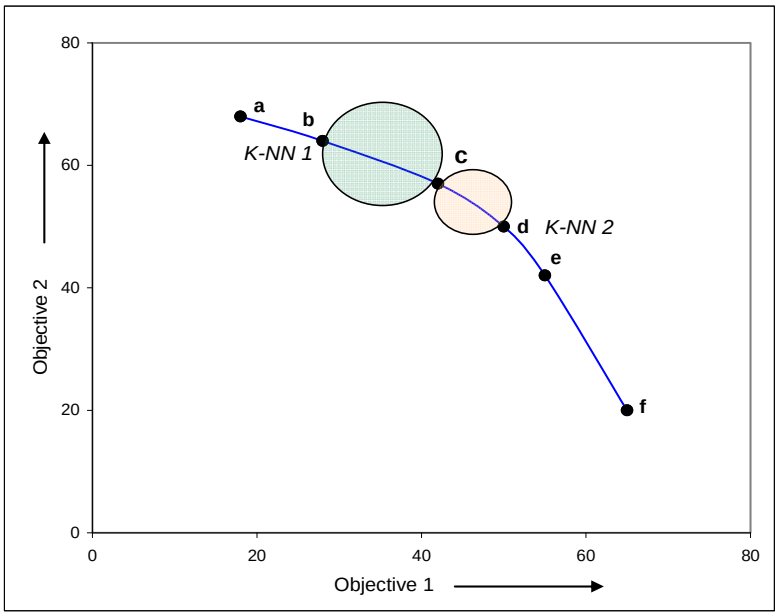

Figure 3.8. K-nearest neighbours (K-NN) based diversity ranking

For the cell density based measure, for example PAES, the solution space is divided into a number of cells or hypercubes, with diversity being measured in terms of how many solutions reside in the same hypercube. Solutions with many neighbouring solutions in the same hypercube will have a lower diversity ranking than solutions with fewer solutions within the same hypercube. For example, in figure 3.9 solution 'd' will have a worse diversity ranking than solution 'f' as it has more neighbouring solutions in the same hypercube.

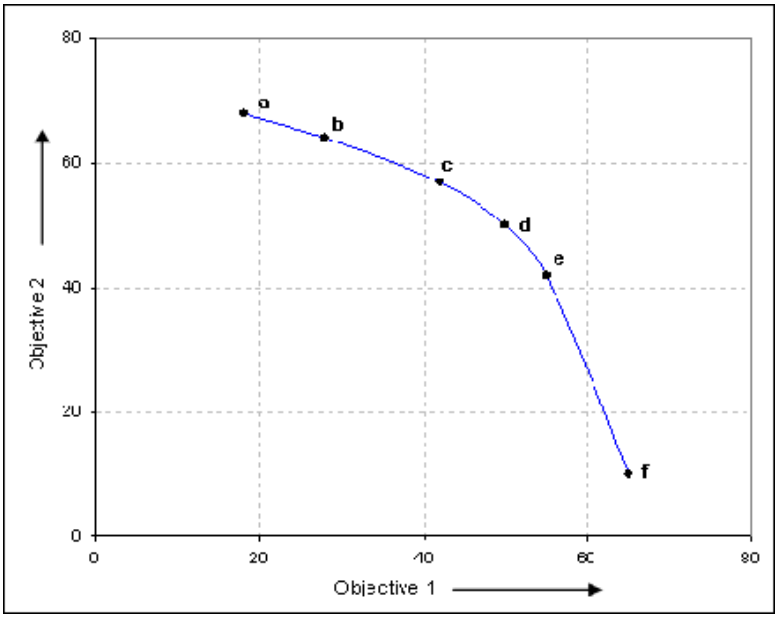

Figure 3.9. Cell density based diversity ranking

Population Archiving Strategies

Another important consideration when designing a MOGA is the way in which non-dominated solutions are archived. This must be done in a manner in which good non-dominated solutions are not lost or replaced with poorer solutions.

A common approach employed by algorithms such as the NSGA-II and SPEA2 is to use two populations, one as an archive population to store the best solutions found so far and another population that contains the current (temporary) population of newly generated offspring solutions. At each generation the two populations are combined and the best half of this combined population then forms the new archive population. Which solutions are considered 'the best' is based upon the dominance rank and the crowding distance. However there are known problems with this approach as the algorithm may in fact remove good Pareto-optimal solutions from the archive during population updating. Consider the problem encountered by the NSGA-II. Figure 3.10 shows the original archived population of solutions. Figure 3.11 then shows the combined current population and archived population, as shown, the solutions in the middle will be removed as

they are the solutions in the most crowded neighbourhood. The new elite archive is then seen in figure 3.12, from this it is clear that the new archive is in fact worse than the previously archive in terms of population diversity.

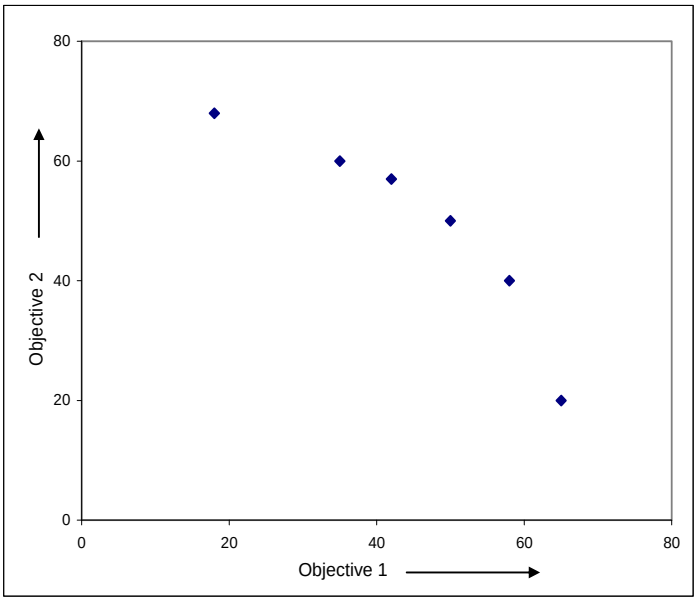

Figure 3.10. Population archiving strategy of NSGA-II- the original archive population

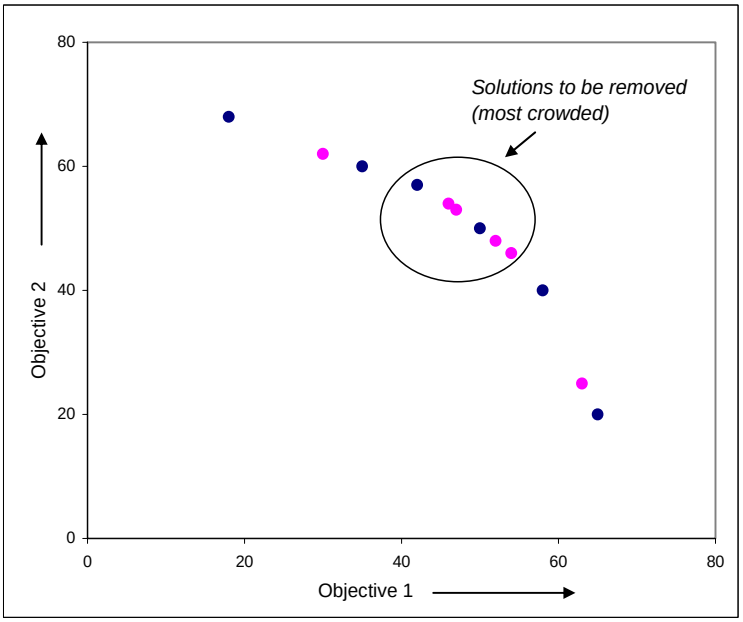

Figure 3.11. Population archiving strategy of NSGA-II- the combined population

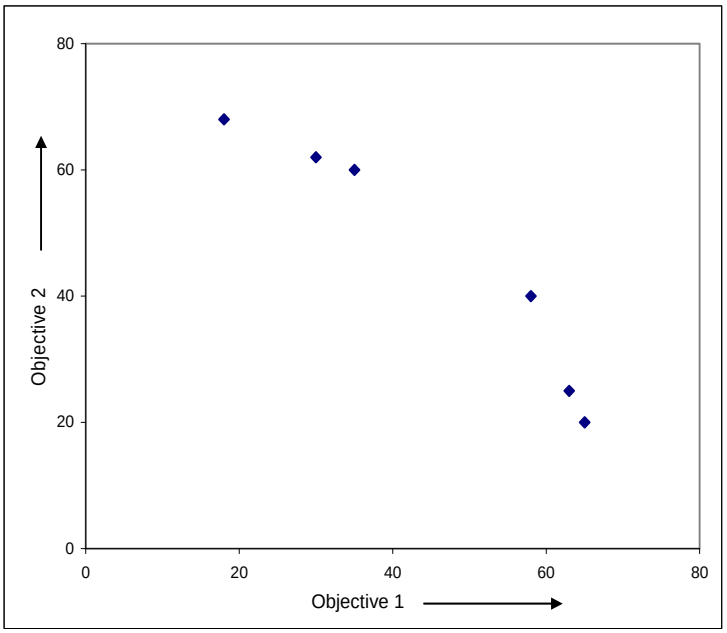

Figure 3.12. Population archiving strategy of NSGA-II- the new archive population

This problem has been addressed by Kukkonen and Deb (2006) who improved the NSGA-II by providing a 'pruning strategy' to iteratively reduce the combined population by selectively removing (pruning) the worst (most crowded) solutions until the current population reaches the desired elite population size. At each iteration the worst solution is removed from a crowded neighbourhood, and after removal the crowding distance measure is recalculated for all solutions in that neighbourhood. This pruning strategy ensures that the 'goodness' in terms of solution spread is improved (or at least not worsened) when the two populations are merged at each generation.

The pruning strategy can be compared to the population update method used in SPEA-2. In SPEA-2 the archive population is steadily updated by adding one new solution at a time. At each iteration the distances between K neighbours for each solution are calculated and the solution residing in the most crowded region is removed. In this way SPEA-2 is able to maintain a more diverse spread of solutions than the original NGSA-II. However greater computational load is needed for SPEA-2 than some other algorithms such as NGSA-II.

3.8. Hybrid Multi-objective Genetic Algorithms

In recent years, the development of hybrid GAs has been one of most significant trends in evolutionary computation. One such approach is to combine GA based genetic operators with a local search heuristic, aiming at improving the overall search efficiency of the evolutionary algorithm. This class of algorithms is often referred to as Multi-Objective Genetic Local Search (MOGLS) algorithms and in recent years they have proved to be a very effective class of methods for combinatory optimisation problems (Ishibuchi and Murata, 1996)

The local search element of these algorithms can in fact be seen as a special kind of mutation operator, where small changes are made to the individual's genes a number of times in an iterative manner to slowly move towards a better solution. This entails the solution fitness being recalculated at each iteration of the search.

Jaszkiewicz (1998) presented the random directions multiple objective genetic local search algorithm (RD-MOGLS). The method is based on the idea of simultaneous optimisation of a range of aggregated scalar functions. At each iteration of the search a weighted scalar function (weighted Tchebycheff function) is drawn at random. Then a sample of the best solutions (in regards to the weighted scalar) is selected from the population. The sample is treated as a temporary genetic population. A number of randomly selected pairs of solutions from the temporary population are then mated to form new offspring. Local search is then applied to improve the fitness of the new offspring solution. The search direction is also specified by the random weight vector.

Ishibuchi and Murata (1996) also proposed a multiple objective genetic local search algorithm (MOGLS). The MOGLS is a GA-based hybrid algorithm for finding a set of Pareto optimal solutions. Like Jaszkiewicz (1998) their method also uses a randomly generated weight vector (weighted sum of multiple objectives) at each iteration of the search. The weight vector is used as a fitness function with which to

select a pair of parent's solutions and is also employed in the local search procedure. They used an external reference population to store the non-dominated solutions, which is updated at each iteration of the search.

Ishibuchi et al (2003) extended their work to further improve the efficiency of their MOGLS. The algorithm was improved by using a tournament selection to choose only the most promising individuals as starting solutions for the application of local search. They also pointed out that the use of the same random weight vector for the local search direction and the genetic search direction (mating and selection) may not be appropriate, so to improve the algorithms efficiency they proposed assigning a weight vector based upon parents' objective achievements for the local search direction.

More recently, Ishibuchi et al (2008b) have also hybridised the NGSA-II with a local search operator and have reported significant improvements over the unmodified NGA-II when applied to many objective combinatory optimisation problems.

3.9. Choosing a Suitable Solution from the Pareto-set

Once the set of Pareto-optimal solutions has been generation using the MOGGA the DM still has to pick a final solution. This is not an arbitrary task especially when there are potentially a couple of hundred solutions to choose from. The chosen solution will therefore depend upon the DM's preference in regards to the considered importance of each objective and the inevitable compromises that will need to be made between these potentially conflicting objectives. The problem then falls into the field of multi-criteria decision making, which can be seen as a sister field to that of multi-criteria optimisation. There are numerous multi-criteria decision making approaches in the literature that could be followed to help the DM choose a suitable solution from a set of alternatives. One of the most popular and simple approaches is the Analytical Hierarchical Process (AHP).

The AHP was first developed by Saaty (1980) and provides a comprehensive framework for structuring a decision problem and evaluating alternative solutions. AHP is based upon the organisation of the decision problem into a logical hierarchy of sub-objectives. At each level of the hierarchy, the DM must evaluate the various objectives by quantitatively or qualitatively comparing them to one another. The AHP then converts these evaluations to numerical weights or priorities for each sub-objective of the hierarchy, allowing diverse and often incommensurable elements to be compared to one another in a rational and consistent way. Once the weighted hierarchy has been constructed the AHP calculates the overall priorities for each of the decision alternatives. The overall priorities for each of the decision alternatives are of course based upon each of the alternative's ability to solve the various sub-objectives along with the corresponding sub-objectives weights from the hierarchy. In figure 3.13 an example AHP can be seen for choosing the best city to live.

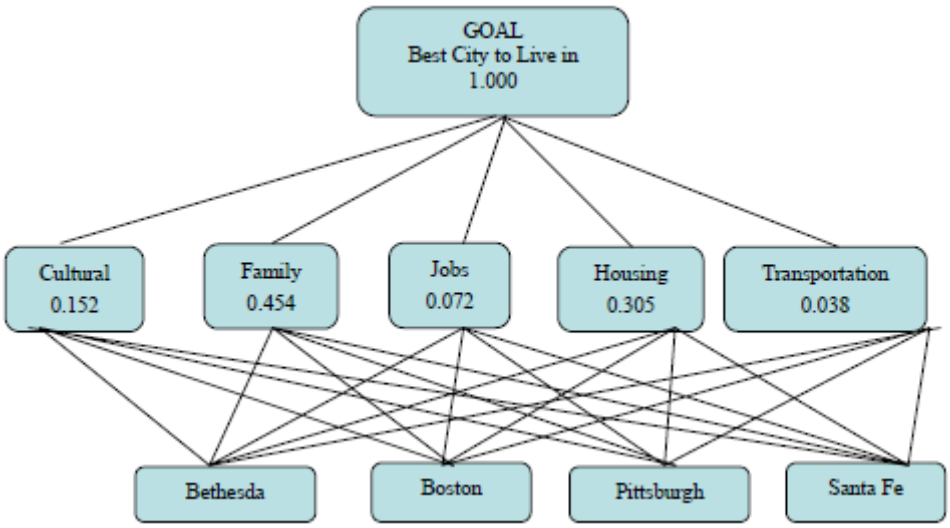

Figure 3.13 Example AHP for choosing best city to live (Saaty,2004)

Using the principles of the AHP it is possible to explore the Pareto-optimal solution set in a comprehensive yet simplistic way. By arranging the different optimisation objectives into a hierarchy and by then setting priorities at each level it is possible

to perform ranking of the various alternative solutions present in the Pareto-optimal set. The solution set can be explored by changing the objective priorities at the various levels of the hierarchy and presenting the corresponding best solutions to the DM for consideration. Solutions that correspond to different AHP preferences can then be compared until the most suitable solution is found.

3.10. Discussion

The most common and effective approach to multi-objective optimisation is to produce a (Pareto-optimal) set of solutions. These solution sets can then be used to examine the potential trade-offs that will exist between different solutions and allow the choice of a suitable compromise solution. One of the most effective means of producing a Pareto-optimal set is to use a suitably developed GA.

Perhaps the most simple way of handling the multi-objective problem is to use an aggregated objective based GA. In this way all the objectives are weighted and then aggregated to form a single vector that is maximised or minimised by the GA. However, only one solution is produced at a time, thus producing a whole set of (Pareto-optimal) solutions with this method can be tedious and time-consuming. Furthermore, it is sometimes difficult to find a suitable approximation of the Pareto-front using an aggregate approach, when, for example, the Pareto-front is non-convex or discontinuous.

The review has highlighted that, since the introduction of GAs three decades ago, much work has been done within the field of evolutionary computing to adapt these traditional single solution GAs to deal with complex multi-objective problems. The preferred method of generating a Pareto-optimal solution set is now to use a MOGA that will generate the whole set in one single optimisation run, without the need of user defined weights or goals. To accomplish this the state-of-the-art MOGAs employ, for the solution fitness assignment, the concept of Pareto-dominance and include some form of diversity preservation method to prevent solutions clumping around certain points on the Pareto-front.

It should be noted however that there are known problems with the current Pareto-dominance based algorithms when dealing with 'many' (more that 4) objective problems. When the number of objectives rises there is an increased chance of all solutions quickly becoming locally non-dominated during the search. This severely weakens the Pareto dominance-based ranking approach and deteriorates the convergence of the solution sets toward the (true) Pareto-optimal front. See figure 3.14.

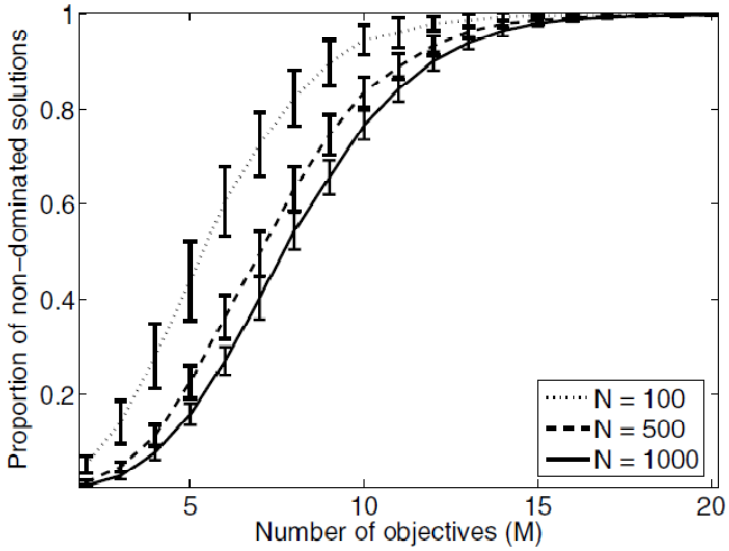

Figure 3.14 Proportion of non-dominated solutions with respect to the number of objectives (Kukkonen and Lampinen, 2007)

For many-objective problems the diversity ranking method thus becomes the primary means of differentiating the non-dominated solutions and measuring solution fitness. However these methods (crowding distance, cell density) are good at maintaining diversity yet can be poor at moving solutions towards the true Pareto-front. Alternative approaches to fitness assignment for non-dominated solutions can be followed, with various studies having been done by researchers such as Kukkonen and Lampinen (2007) and Hughes (2005). For example, for an 'average rank' ranking, the fitness is given by comparing all solutions against each other and noting their rank for each objective. Fitness is then given as the solution's average rank for all objectives. Another approach for elevating the convergence problems of many objective optimisation is to include some form of prior

preference information (if known) for the various objectives as seen in the work of Deb and Saxena (2006). A further method is to provide some form of global or local search vector during mating and mutation such as seen in Ishibuchi et al (2003) and Jaszkiewicz (1998). In this way solutions are pushed towards unexplored and potentially more optimal areas of the search space.

Another issue that grouping problems face is that the standard genetic operators are poorly designed to deal with the complexities of the grouping problem. It has been seen however that this part of the problem can be tackled by using a modified set of genetic operators such as those described by Falkenauer (1998). In addition it has been seen that by hybridising a MOGA with a local search heuristic better results are also possible for combinatory optimisation.

Finally, in terms of exploring and choosing solutions from the Pareto-set, the review has highlighted that the Analytical Hierarchical Process could present a suitable method. By constructing a modularisation objective hierarchy, it should be possible to change objective priorities at the various levels to explore the solution set and ultimately choose a suitable solution to the modularisation problem at hand.

3.11. Conclusion

This review has provided a general overview of the basic principles of evolutionary computing and given a snapshot of the main techniques that have been created to solve complex multi-objective optimisation problems. The field of evolutionary algorithms is extremely active and is continually changing with new research that is constantly pushing on the frontiers of knowledge. This review has in fact highlighted that there are no ‘off-the-shelf’ MOGAs available that are capable of efficiently dealing with complex grouping problems such as product modularisation.

It can be concluded however that the most effective approach to multi-objective optimisation is to produce a (Pareto-optimal) set of solutions that represents the approximate area of optimal solution space for the problem. These solution sets can

then be used to examine the potential trade-offs (between objectives) that will exist between different solutions and hence give the DM greater insight into the optimisation problem. The most effective means of producing a Pareto-optimal set is to use a suitably modified MOGA. However, for the development of a suitable MOGA for product modularisation there are a number of issues that must be addressed. Issues that need to be addressed for the algorithm development are:

- What are the most suitable GA encoding, selection, cross-over and mutation mechanisms for the product modularisation problem? The review has highlighted the Grouping GA (GGA) developed by Falkenauer (1998) as a possible candidate. However, how can this GGA be adapted for the modularisation problem?

- What is the most suitable means of creating a set of Pareto-optimal solutions for the modularisation problem? Can the principles of using a Pareto-dominance based approach be developed to produce a whole set of alternative solutions (in one run) for the modularisation problem.

- Can a local search heuristic be implemented to improve the performance of the MOGA? Local search has been seen to improve the performance of MOGAs when dealing with combinatory optimisation problems. Can a suitable local search heuristic be developed for multi-objective product modularisation?

CHAPTER 4

4. Development of Modularity Optimisation Framework

4.1. Introduction

This chapter will present the various aspects that have been considered in the development of a multi-objective optimisation framework. The chapter will aim to build upon previous research and addresses the research gaps identified in the review chapters to provide an improved multi-objective approach to modular product architecture optimisation.

4.2. Rationale for Framework.

There have been many product modularisation frameworks developed over the past decade. The vast majority of these methods pursue a 'bottom-up' approach in which low-level product elements (components) are grouped to form larger product elements (modules). These methods have been labelled configuration based approaches. The proposed framework presented in this chapter is also a configuration approach that aims to address a number of shortcomings in current works - as was discussed in detail at the end of chapter 2.

In summary, it is argued that current modularisation methods are highly ambiguous and do not adequately deal with the complex multi-objective nature of product architecture decisions. Current optimisation frameworks for product modularisation are simplistic (aggregated objective) approaches and thus cannot guarantee that optimal modular architectures can be found. Furthermore, in a decision making process (i.e product architecture) alternative solutions should be considered before arriving at a final decision. This implies that a good set of alternative solutions can in fact be found, in order to make comparisons. However finding a set of optimal solutions (for comparison) with current methods is problematic and time-

consuming. In addition most of the modularisation methods offer little in terms of how to represent and evaluate modularity according to the various different modularity viewpoints discussed in chapter 2.

To address these issues the development of an improved multiple-objective modularisation framework will be presented in this chapter. The framework draws together a number of principles from the literature (as depicted in figure 4.1) to provide an integrated method of multi-objective product modularisation.

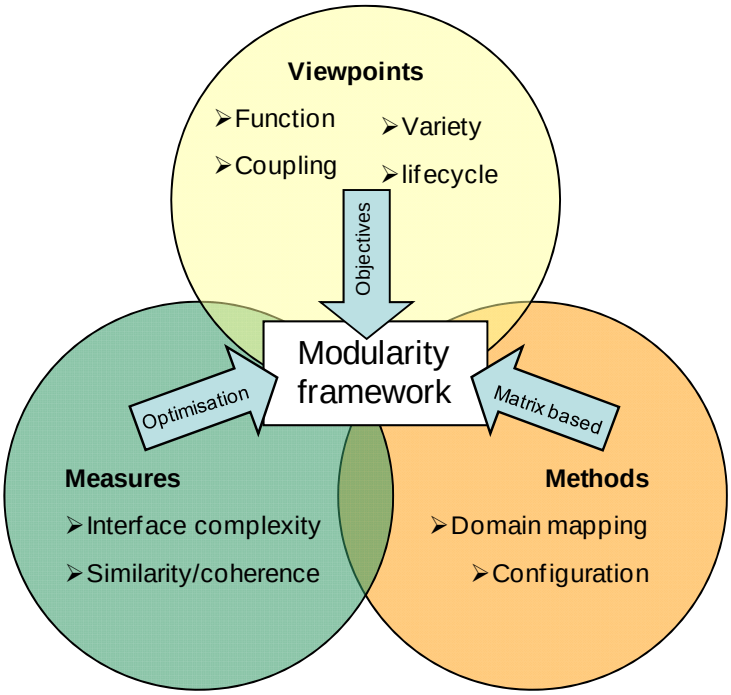

Figure 4.1. The multiple facets of modularity integrated into the developed framework

4.3. Overview of the Framework

The developed framework presents a matrix based approach to represent the various dependencies and strategic similarities that occur between a product's components. A multi-objective grouping genetic algorithm (MOGGA) is used to search the various matrices and produce a whole set of optimal modular product configurations which are then explored by the decision maker (DM) to find the best compromise solution.

As depicted in figure 4.2 the framework has four steps: 1) product decomposition 2) interaction analysis 3) formation of modular architectures and 4) scenario analysis. The aim of this chapter is to introduce the key principles that have been used for the development of the framework. The actual steps will be discussed further in chapters 7 and 8 where the software prototype will be presented and the application of the steps will be shown on example products.

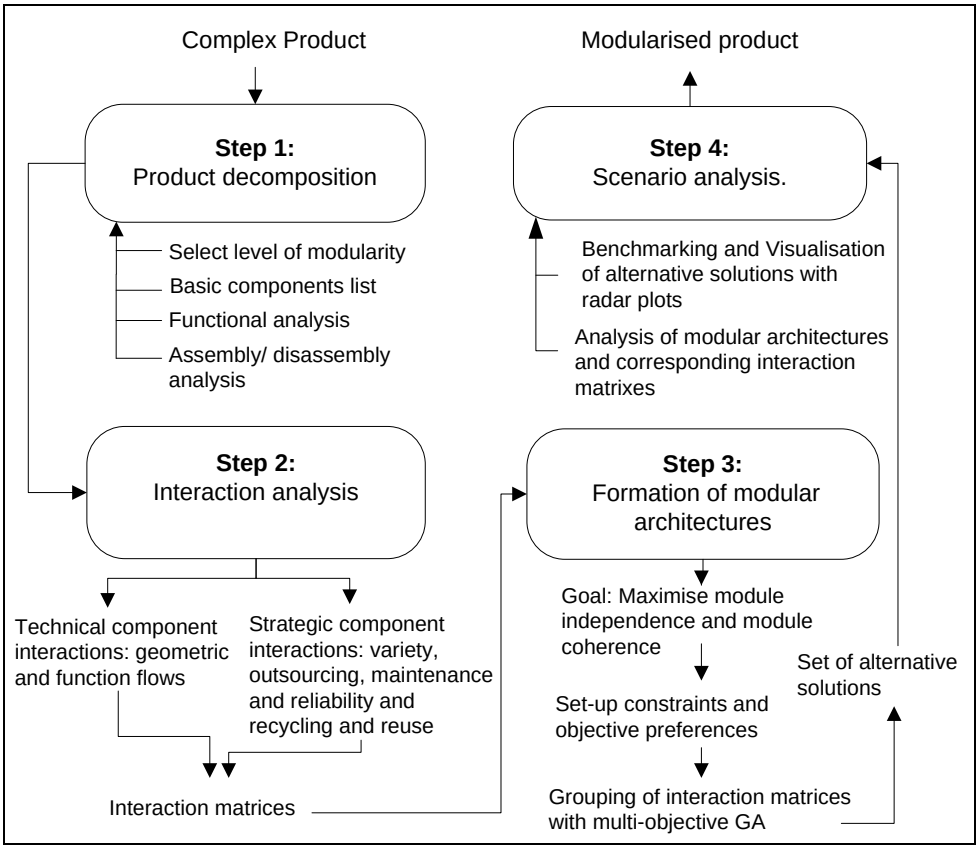

Figure 4.2. Overview of developed framework steps

4.4. Scope of the Framework

It is argued that product architecture decisions are best made early in the development process before costs are locked in and constraints are imposed. Therefore if possible product modularisation (with the framework) should be applied straight after the conceptual design stages once all of the technical concepts have been chosen yet before the detailed design phase has begun. At this stage in the product development cycle there should still be enough design freedom to make changes to the product structure if necessary.

The developed framework is however not meant to provide a radical new method of product development. It is intended that the framework can be integrated into existing product development approaches, either as part of a 'top-down' product development approach for new product designs, or as part of a 'bottom-up' product redesign and improvement approach. For example, the framework can be integrated and used in conjunction with DFX principles - such as design for assembly/ disassembly (Boothroyd et al, 1994), design for variety (Martin and Ishii, 2000) and design for the environment. Such approaches can be used after application of the framework to further improve the 'optimal' modular architecture though redesign of component attributes.

It must also be stated that the proposed framework will not be suitable for all products. Some types of products may gain little advantage from modularity. It is argued however that if the product is reasonably complex[1], then it is highly likely there will be some value in looking at optimisation of the product architecture. It has also been discussed by numerous researchers (Marshall, 1998; Pimmler and Eppinger, 1994) that for complex products there will be more than one level of modularity achievable. Thus before attempting to create a modular product architecture one must think carefully about the required level that one wishes to modularise the product at. For a highly complex product like a car, made of

[1] A complex product is defined here as a product that contains a mixture of technologies and/ or a large number of components.

thousands of parts, modularisation at the lowest (component) level may not be practical, both from a complexity and computational point of view. For these complex products one may wish to take advantage of the existing product hierarchy i.e. the sub-assembly level and modularise the product at this level. Alternatively one may wish to modularise each product sub-system separately. For example in the car industry the product structure is clearly decomposed into sub-systems, e.g. drive train, HVAC (heating ventilation and air conditioning), braking system etc. Each of these sub-systems could then be modularised.

In this research it is presumed that there is a need for product modularity. If the need for modularity is unclear then it is suggested that the reader refers to other works such as that of Marshall (1998), who's 'holonic product design workbook' provides a detailed means to assess the need for product modularity.

4.5. Representation of Modularity

In the framework interaction matrices (DSMs) are used for representing product modularity. These matrix representations are considered a somewhat natural way of representing the complex interactions that must be analysed across the various modularity viewpoints in order to form optimal modular structures - they are highly visual and can be readily manipulated with optimisation algorithms.

Essentially with this approach, like other 'configuration' based methods, we are still proposing that the product is decomposed into lower level product elements (components) which are then grouped into larger product sub-systems (modules)- However, in the framework the product is being represented in both the physical (component level) and functional domains. A cross-domain functional mapping approach allows the various modularity objectives to be analysed at two different levels of abstraction. For example the product variety mix can be analysed at the functional domain to identify common 'platform' based functions. These common platforms may not be obvious at lower levels of abstraction, such as the component level.

4.6. Modularity Metrics

Two modularity metrics have been developed for this research; these are the module independence ratio and the module coherence ratio, and these will be used as the criteria for module grouping during optimisation with the developed algorithm. These metrics are based upon two key modularity principles discussed in the literature. The objective of modularisation from the module independence perspective is to achieve loosely coupled, independent modules. This can be achieved by ensuring that component dependencies are kept *within* modules rather than *between* modules. Module coherence is concerned with ensuring that components *within* modules are similar in terms of the modularisation objective they are addressing. In the proposed framework the module independence metric is used as a goal to improve the more technical aspects of modularity (function binding and coupling) whereas the module coherence metric is associated with the strategically aspects of modularity (variety, maintenance, recycling etc.).

Ideally, an optimal modular architecture will have loosely coupled, functionally independent modules that are highly coherent in terms of their response to the various strategic modularisation objectives. However, in reality, the two modularity concepts are contradictory. Improving the independence of modules will often mean that the coherence of modules deteriorates and vice versa. To improve the module independence, fewer, larger modules are usually sought. Whereas, in order to achieve high module coherence a larger number of smaller modules will often be necessary. Hence an important part of the work in this thesis is to provide a framework that is able to generate and evaluate a number of alternative modular architectures based upon the trade-offs between module independence and module coherence.

Module Independence Ratio

This metric measures the ratio of the component dependencies within modules divided by the total dependencies between all components. A higher ratio means more interactions are kept within modules rather than across modules, and the modules are more independent. This can be seen in figure 4.3.

The module independence (MI) metric:

$$MI = \sum_{m=1}^{M} \frac{CI_n}{CI_{\max_n}} \quad (1)$$

Where:

M = module number
CI_n = the number of couplings within module m
$CI_{\max_n}$ = the maximum strength of all component couplings

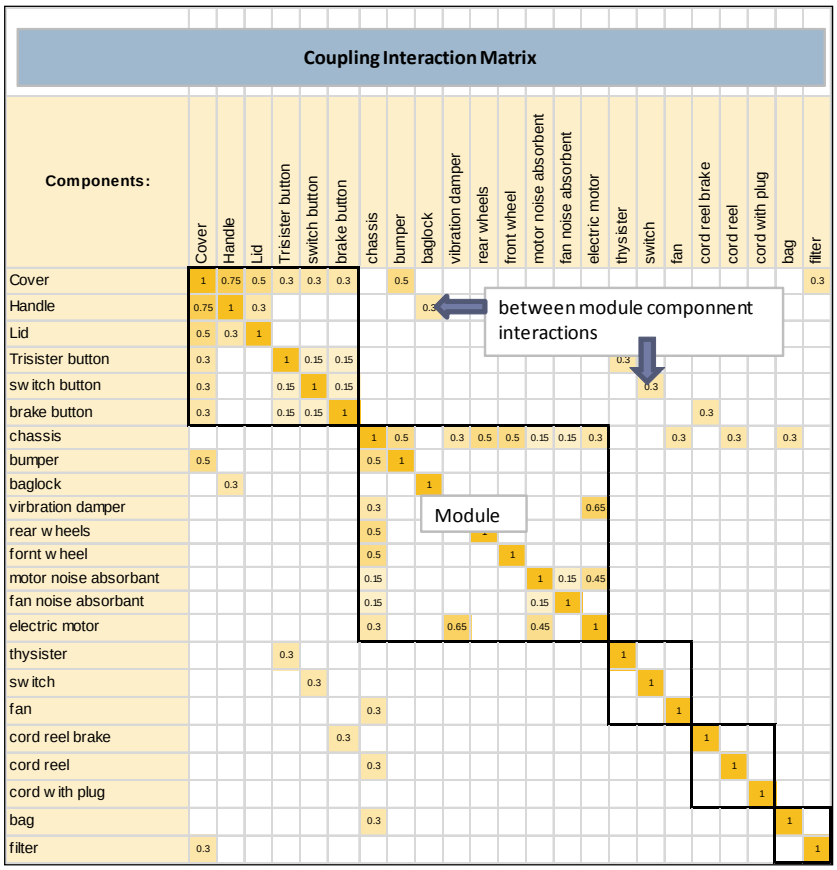

Figure 4.3. Coupling interaction matrix showing module independence

Module Coherence Ratio

The module coherence ratio measures the total number of component interactions within modules divided by the maximum potential number of component interactions within the modules. This principle can be seen in figure 4.4.

The module coherence (MC) metric:

$$MC = \sum_{m=1}^{M} \frac{SI_n}{SI_{\max_n}} \qquad (2)$$

Where:

M = module number

SI_n = total strategic component interactions within module m

$SI_{\max_n}$ = total possible strategic component interactions within module

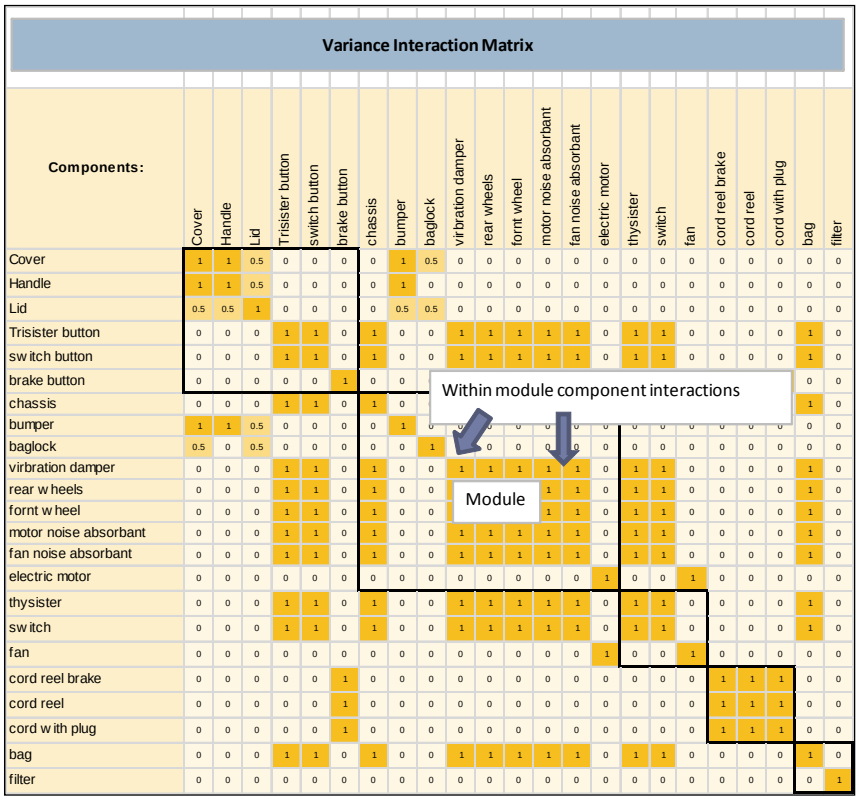

Figure 4.4. Variance interaction matrix showing module coherence

4.7. Reconciling Modularisation Objectives

The advantages of product modularity can be seen across all phases of the product lifecycle, as outlined in the literature survey. The problem is how can the various objectives be included in a multi-objective optimisation framework. Indeed, is it really necessary to include all of the modularisation objectives? Are there any similar objectives addressing the same fundamental issues? Similarly, are there any conflicting objectives? Thus can any of the objectives be eliminated or combined? These are important issues as they can dramatically reduce both the complexity of the problem and the level of information content needed. These issues are not new and have in fact been outlined (or at least hinted at) by numerous other researchers, including Gershenson et al (2004), Fixson (2005), Blackenfelt (2001) and Stake (2000).

Blackenfelt (2001) for example, pointed out that there are overlaps and conflicts between some of the modularisation objectives (or modular drivers as they are often referred to). Some objectives are polar opposites (like two sides of a coin), some complement each other (or are similar, pursuing the same goals) and some are alternatives to reach the same modularisation goal. For example, variety can be seen as the opposite to commonality, recycling is an alternative to remanufacturing or disposal. Blackenfelt (2001) ends up condensing the 12 strategic modular drivers presented by Ericsson and Erixon (1999) down to four sets of drivers focused towards product variety issues: 'variant versus common', 'reuse versus develop', 'make versus buy' and 'carry over versus change'. This is a step in the right direction, enabling a more focused evaluation of product modularity.

In this research, the first step towards addressing some of the previously discussed problems has, like Blackenfelt (2001), been to form a condensed set of key modularisation objectives. This has been achieved by establishing a list of all the primary modularisation objectives from the literature - as outlined in table 4.1.

Table 4.1. The various modularity objectives

Modularity Objective	Grouping components into modules because….	Source
Physical Coupling	they have strong physical interactions	Plimmer & Eppinger(1994) ;Gu & Sosale (1999); Newcomb (1996)
Functional flows	they have functional flows between them	Plimmer & Eppinger(1994) ;Gu & Sosale (1999);
Function binding	they contribute to the same sub-function	Stone (1999); Ulrich and Tung (1994); Suh (1995)
Planned product changes	they are planned to be redesigned in the near future	Ericsson & Erixon (1999)
Carry over	they will stay the same in the near future	Ericsson & Erixon (1999)
Upgrading	they will be upgraded when new components are developed	Ericsson & Erixon (1999);Gu & Sosale (1999); Ulrich & Eppinger (1995)
Technological evolution	they have a high level of technical evolution	Ericsson & Erixon (1999)
Variety	they will vary across the product range	Ulrich & Eppinger (1995); Ericsson & Erixon, (1999); Gu & Sosale, (1999)
Styling	they are subject to trends and fashion needs	Ericsson & Erixon (1999)
Commonality	they will become common across the product range	Ulrich & Eppinger (1995); Ericsson & Erixon (1999); Gu & Sosale (1999)
Out-sourcing (buy)	they should be bought from a supplier	Ericsson & Erixon (1999)
In-sourcing (make)	they should be developed/ and or made in house	Ericsson & Erixon (1999)
Separate testing	they should be tested together before final assembly	Ericsson & Erixon (1999)
Maintenance and service	they have similar maintenance and service needs	Gu & Sosale (1999); Ulrich & Eppinger(1995)
Wear-out life	they have similar reliability	Gu & Sosale (1999); Ulrich & Eppinger (1995)
Reuse	they can be reused or remanufactured	Newcomb (1996);Gu & Sosale (1999)
Recycling	they can be recycled together	Newcomb (1996);Gu & Sosale (1999)

By looking at the modularisation objectives that are either complementary to each other, opposite to each other, or which provide alternatives to one another, a somewhat condensed set of objectives has been developed. This can be seen in Figure 4.5, which provides an objective reconciliation matrix. Six objectives in total have been identified. An overview of the aims of each chosen objective is given in Table 4.2. The actual evaluation of the six modularisation objectives will be given in detail in chapter 7 where the software implementation of the framework will be discussed.

Modularity Objective	Physical Coupling	Functional flows	Function binding	planned product change	carry over	upgrading	styling	technological evolution	variety	commonality	out-sourcing (buy)	in-sourcing (make)	separate testing	maintenance	wear-out life	reuse	recycling	Reconciled Objective
Physical Coupling	1																	**Loose Coupling**
Functional interactions	C	1																
Function binding			1															**Function Binding**
planned product changes				1														**Variance** (Variety vs. Common platform)
carry over				O	1													
upgrading				A	O	1												
styling				C	O	C	1											
technological evolution				C	O	A	A	1										
variety				C	O	C	A	C	1									
commonality				O	A	O	O	O	O	1								
out-sourcing (buy)											1							**outsourcing** (Make vs. Buy)
in-sourcing (make)											O	1						
separate testing											A	C	1					
maintenance and service														1				**Maintenance and Reliability**
wear-out life														C	1			
reuse																1		**Reuse and Recycling**
recycling																A	1	

A=Addressing the same underlying objective
O= Opposite to
C = Complementary to

Figure 4.5. Set of reconciled strategic modularity objectives

Table 4.2. Overview of the modularity objectives

Objective	Description
Loose Coupling	To group together components that have strong physical and functional relationships to ensure that modules are as independent as possible. This will enable modules to be designed, manufactured, assembled and disassembled as concurrently as possible.
Function Binding	To achieve modules that perform discrete functions. That is to group components that are influenced by the same product sub-functions into the same module. This will help ensure that modules can be reused over future product generations, shared among different products and allow easier product reconfiguration.
Variance	To group components that respond to the same variance modes into the same modules. Non-variant components can then form common platform modules that can be shared across the product family.
Outsourcing	To group components that can be outsourced to various suppliers.
Maintenance and reliability	To group components that have similar maintenance and replacement needs into the same module.
Reuse and Recycling	To gather into the same module components that share the same end of life requirements - to enable module reuse and to make the recycling operations easier and more cost effective to perform.

4.8. Grouping to form Modules

For the modularisation framework a grouping algorithm is applied to find optimal modular architectures (solutions) through manipulation of the data in the various matrices. Each solution is found by varying the membership of components to modules, in each of the interaction matrixes, such that the developed modularity metrics are maximised for the different objectives. Of course it will often be impossible to simultaneously maximise every objective, so different trade-off solutions are produced. As discussed in chapter 3, an appropriate method is to produce a set of (Pareto-optimal) solutions allowing the decision maker (DM) to explore various alternative modular configurations and look at 'what if' scenarios before choosing the most suitable solution. In this way the DM is able to make a more informed decision on the most suitable modular structure for the product.

In the developed framework the modularisation objective achievements for the (Pareto-optimal) solutions are represented by radar plots as shown in the figure 4.6. These type of plots of are highly visual and give the user a suitable means for making comparisons between different solutions during the scenario analysis phase of the framework.

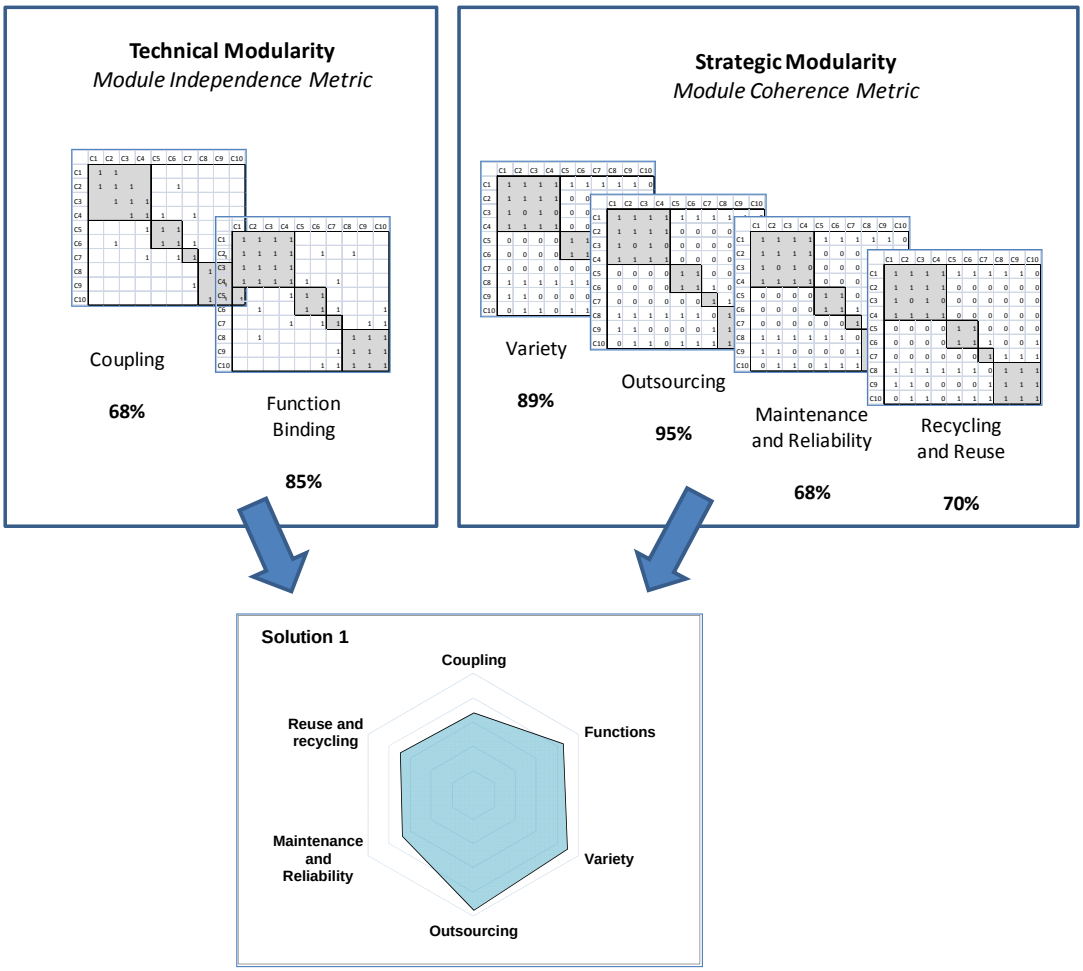

Figure 4.6. Example of an optimal modular solution using the proposed matrix grouping approach

4.9. Modularisation Objective Hierarchy

Another important issue that needs to be addressed for a multi-objective optimisation is how does one quantify the importance of the different modularisation objectives? The developed MOGGA actually creates a whole set of alternative (optimal) modular solutions without needing weights or priories to be set up beforehand. However there still needs to be a way of exploring the different solutions (scenario analysis) and ultimately choosing the most appropriate modular product architecture according to the DM preferences. To address this problem a

'modularisation objective hierarchy' has been developed (shown in figure 4.7) as a means of solution set ranking.

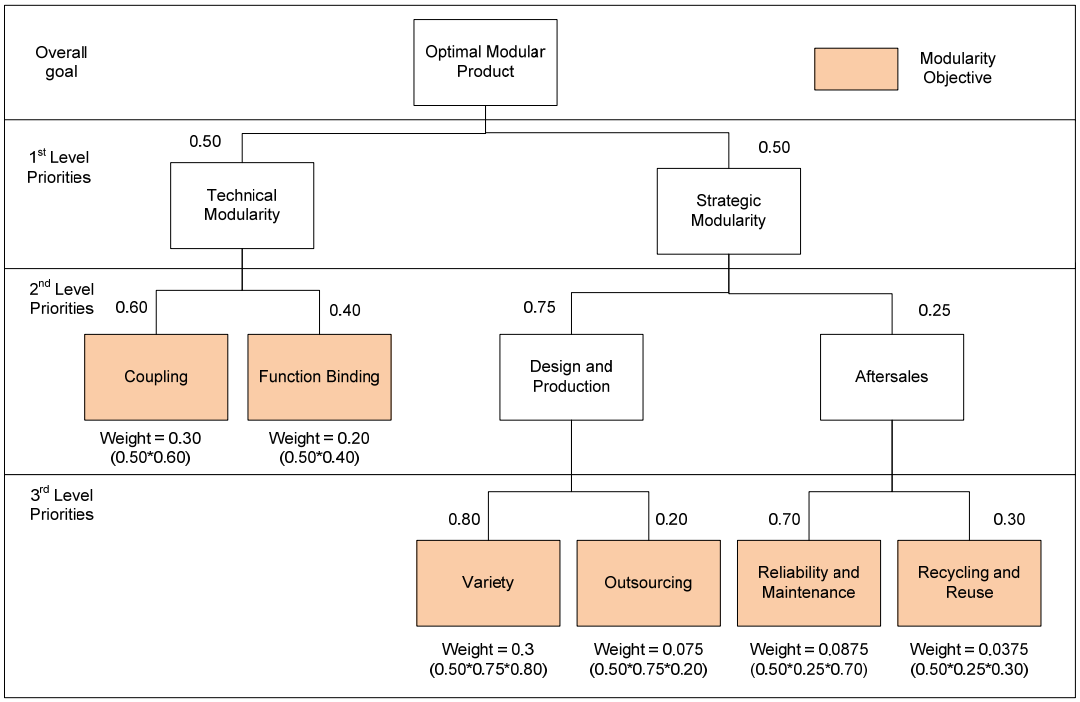

Figure 4.7. Modularisation objective hierarchy

The principle of this hierarchical ranking process is based upon the AHP approach used in multi-criteria decision making. In AHP pair-wise comparisons of the each criterion are made at each level of the hierarchy, ultimately generating a weight vector for each objective. The weight vectors are then used to provide ranking of the solutions generated by the algorithm. For example, starting at the top of the hierarchy the user may decide that the technical modularity objectives are more important than the strategic, this means that greater weights will then be given for 'function binding' and 'loose coupling' objectives. The generated solutions will thus be ranked according to these preference weights and the highest ranking solutions presented to the user for further analysis. By changing the preferences at each level of the hierarchy the solution set can thus be explored in a simple manner. This process will become more apparent during the course of the thesis when the software implementation is described and a case study example will be given

4.10. Conclusion

This chapter has presented the key aspects of the proposed modularisation framework. These aspects include: how modularity will be represented, modularity metrics, what modularity objectives will be used and how they can be ranked and prioritised, and how module grouping will take place using a multi-objective optimisation algorithm. The design and development of this algorithm will be presented in the next chapter.

CHAPTER 5

5. Multi-objective Grouping Genetic Algorithm for Product Modularisation

5.1. Introduction

The design and implementation of a suitable MOGA is a complex problem that can be tackled in many ways. The vast majority of 'state of the art' MOGAs use the concept of Pareto-dominance based ranking coupled with some form of diversity preservation strategy. However, many of the previously developed algorithms would not be suitable for the multi-objective product modularisation problem. This is because most of the developed MOGAs are not designed for combinatory optimisation problems (grouping problems).

This chapter therefore presents the design and development of a MOGA for product modularisation. The chapter begins with an overview of the developed algorithm and the remainder of the chapter then provides a detailed description of the various components that make up the algorithm.

5.2. Overview of Developed Multi-objective Grouping Genetic Algorithm

There are a number of core concepts that have been integrated to provide the overall functionality of the proposed multi-objective grouping genetic algorithm (MOGGA) for product modularisation. Falkenauer's group-based encoding and cross-over schemes are employed along with problem specific gene reallocation and repair heuristics. These are used as the means by which individual solutions are constructed, mated and repaired. Random vector based selection and search is carried out to ensure that new regions of the search space are explored in order to find new Pareto-optimal solutions. In addition dominance and diversity ranking are used for determining which individuals in the population are chosen for survival at each generation. By combining these concepts the algorithm is capable of evolving

a diverse set of Pareto-optimal solutions (alternative modular product architectures). The algorithm consists of four main steps:

Step 1) Initialisation: create an initial population of individuals using the initialisation procedure. Repair any infeasible solutions using the repair heuristic**.**

Start Main loop:

Repeat until the maximum number of generations (G_{max}) is reached.

Step 2) Selection

a) Randomly generate a set of weights to form a weighted scalar fitness function ($RW_{fitness}$)
b) Perform tournament selection to select two parent individuals from the current population. Tournament winners are the individuals that achieve the highest fitness according to the randomly generated $RW_{fitness}$.

Step 3) Recombination

a) Perform mating on the selected parents using the GGA crossover operator and the local search based reallocation heuristic. Use the $RW_{fitness}$ generated during selection to provide the creep direction during local search.
b) Use the repair heuristic to repair an infeasible solution.
c) Make the first offspring solution a temporary member of the current population and perform ranking. Repeat for the second offspring.

Step 4) Ranking

a) Evaluate the fitness of each solution in the population based upon the dominance and diversity fitness, $DD_{fitness}$
b) Replace the weakest solution with the offspring solution, if and only if the offspring solution has a higher fitness. If the offspring solution has the lowest fitness then remove it from the current population.

End main Loop

5.3. Encoding Scheme and Genetic Operators

Product modularisation can be defined as a grouping problem as the aim is to group a set of components (objects) into smaller sub-sets (modules). Grouping problems have been well studied in the literature and solved with appropriate algorithms. Examples of these grouping problems include: the machine part grouping problem, assembly line balancing, graph colouring and the bin packing problem.

The goal of any grouping problem is to partition a set of objects into smaller sub-sets, with each object belonging to exactly one group. For most grouping problems there are often a number of problem specific constraints that the groupings must adhere to. Under these constraints the objective of the problem is then to create groups that minimise a given cost function. For the product modularisation problem the overall goal (cost function) is to maximise module independence (to reduce coupling between modules) and maximise module coherence (maximise component modular driver similarities within modules). Module groupings must adhere to several constraints. Firstly the number of modules must not exceed a user defined maximum or drop below the allowed minimum. Secondly, the number of components in each module must not exceed the maximum defined number. Lastly, components should not be placed in the same module if there is a particular physical restriction or functional infeasibility.

It is possible to solve grouping problems with classical GAs. However as pointed out by Falkenauer (1998) grouping problems present numerous problems for classical GAs. Firstly, the encoding used in a standard GA is highly redundant, which means that solutions with identical groupings are often treated as completely different individuals, reducing the efficiency of the algorithm. Secondly, and more importantly, the standard genetic operators will recklessly break up good groupings as they work with the object part of the chromosome and not the group part. To

address these problems Falkenauer (1998) created the Grouping Genetic Algorithm (GGA) which uses a group based encoding scheme and suitably modified genetic operators.

Grouping Genetic Algorithm: Encoding

As the cost function of a grouping problem is to optimise the memberships of groups and not the properties of individual objects it follows that a GA adapted for a grouping problem should work with the group parts of the chromosome and not the individual objects.

In a standard GA each solution is encoded into a chromosome which is made up of a number of genes. For the grouping problem each gene would represent an item belonging to a group. For example, the chromosome ABBCCDD contains seven genes that represent a solution that places the second and third items into the same group, the fourth and fifth items in the same group, the sixth and seventh items in the same group and the first item in a group of its own. Exactly the same solution could be represented with the chromosome BCCDDAA. From this it is clear to see that there will be inefficiencies due to redundancy of the chromosome representation.

Falkenauer's approach is to shift the encoding of the genes to represent groups of items rather than individual items. In this way each group of items is encoded into a number of genes as seen in figure 5.1. This is significant as it allows the GA cross-over and mutation operators to work with the groups of items rather than individual items. In this way the integrity of the groups is better preserved

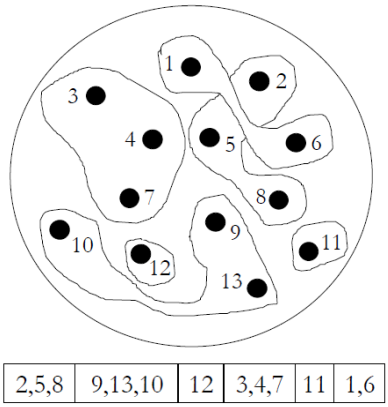

Figure 5.1 Group based encoding taken from Falkenauer (1998)

Grouping Genetic Algorithm: Cross-over

Cross-over is the primary means of exploring the search space to find the most promising solutions to the given problem. The main idea of cross-over is to recombine the genes of two parent solutions to produce offspring solutions that will inherit their 'good qualities' to provide further 'optimal' solutions. There are many cross-over operators that have been created but all work with the basic principle of recombining genes to create offspring.

To better understand the significance of the GGA encoding during cross-over let us consider the two parent solutions in figure 5.2 that have been chosen for recombination. The solutions are encoded using a standard GA encoding scheme (each gene representing one item) and are then recombined using a standard single point cross-over operation to produce two offspring solutions.

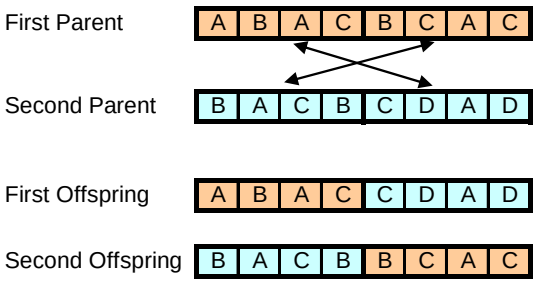

Figure 5.2. Problems with standard GA cross-over operation for grouping problem

It is clear to see that after recombination the offspring solutions that have been created are completely different to their parents and the offspring have not inherited any of the 'good' groups that were present in their parents. In fact the offsprings' new groupings may be completely infeasible due to the violation of constraints. Due to this inability to preserve the integrity of groups during cross-over the GA has lost the advantage of being able to breed good offspring solutions by combining the favourable qualities of two parents. Hence the GA will perform little better than that of a random search based algorithm.

To overcome the failures of the standard GA cross-over when applied to grouping problems, Falkenauer uses a five-step cross-over operator that is based upon the group based encoding scheme, which is the basic cross-over scheme employed in the algorithm. The steps are detailed below and in figure 5.3.

Step 1) Select at random two crossing sites, delimiting the crossing section, in each of the two parents.

Step 2) Inject the contents of the crossing section of the first parent at the first crossing site of the second parent. Recall that this means injecting some of the groups from the first into the second.

Step 3) Eliminate all objects now occurring twice from the groups they were members of in the second parent, so that the 'old' membership of these objects gives way to the membership specified by the 'new' injected groups. Consequently, some of the 'old' groups coming from the second parent are altered: they do not contain all the objects anymore, since some of those objects had to be eliminated.

Step 4) Replace any missing objects in the new group's (offspring), according to the hard constraints of the problem and the cost function to optimise. At this stage, local problem dependent heuristics can be applied.

Step 5) Apply the points 2 through 4 to the two parents with their roles reversed in order to generate the second child.

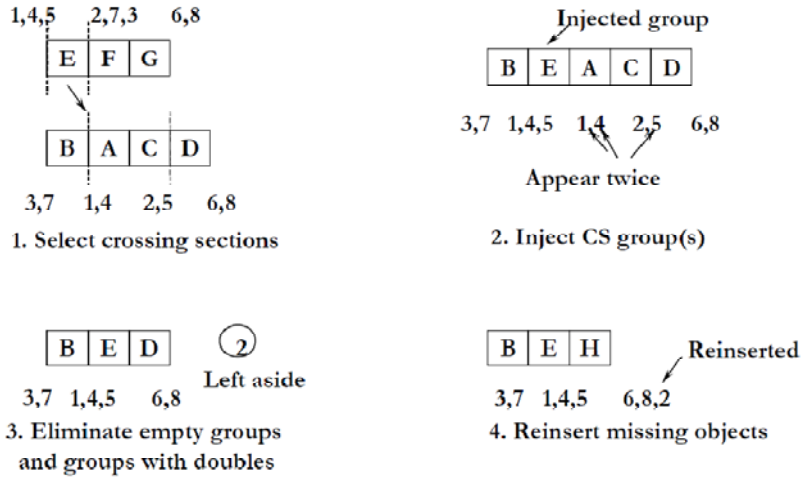

Figure 5.3 Grouping genetic algorithm cross-over operation (Falkenauer, 1998)

Reallocation Heuristic: Local Search

As part of the GGA cross-over a user defined reallocation step is needed (step 4) to reallocate missing objects (components) to groups (modules). The missing objects are the objects which were deleted during crossover as they appeared twice in the offspring's new groups.

As there is no generic replacement heuristic that can be used for all problems it is essential that the replacement heuristic is properly designed for the specific grouping problem at hand. One naive approach might be to randomly reallocate the missing components to modules. However this approach may end up destroying the integrity or 'goodness' of current modules and module groupings may end up being infeasible due to the violation of constraints. What is preferable is an intelligent reallocation heuristic. This will ensure that that a new offspring solution creeps towards a Pareto-optimal point and not away from one.

The significance of the reallocation step was studied by Brown and Sumichrast (2003) who conclude that the reallocation step can heavily affect the overall performance of the GGA. Their study found that if an intelligent reallocation

heuristic is used rather than random reallocation then the performance of the GA can be greatly improved. The GA will find better solutions and will take less time to converge to fitter solutions.

This 'intelligent' reallocation step can in fact be seen as a local search method as we are implementing small local changes (improvements) to module (group) memberships during component reallocation. Coupling local search with GAs has been seen as an efficient and effective means of improving the performance of the GA when solving combinatory optimisation problems, (Jaszkiewicz, 1998, 2002; Ishibuchi et al, 2003). For example, Ishibuchi et al (2008b) more recently hybridized a local search method with the NGSA II multi-objective algorithm and tested the algorithm on a number of combinatory optimisation problems. Ishibuchi et al reported improved convergence towards the Pareto-front as well as improved diversity of the obtained solutions of the non-dominated set.

For these discussed reasons an intelligent means of reallocation (reallocation heuristic) has been designed for the cross-over operation. The heuristic is specifically designed for the modularisation problem and proceeds as follows.

Step 1) Randomly select a missing $component_n$ and temporarily allocate to the first $module_{n=1}$ and evaluate the fitness of the module groupings according to the random weighted scalar fitness function $RW_{fitness}$

Step 2) Temporarily allocate the selected missing $component_n$ to the next $module_{n+1}$ and again evaluate using random weighted scalar fitness function $RW_{fitness}$.

Step 3) Repeat step 2 until $module_{n+i}$ reaches $module_{max}$

Step 4) Allocate the selected $component_n$ to the $module_n$ which showed the best fitness.

Step 5) Repeat steps 1 to 4 until each missing component is allocated to a module.

The five steps of the reallocation heuristic ensure that missing components are gradually reallocated to modules in a manner that ensures the best improvement of fitness according to the given randomly weighted scalar fitness function. This random vector weighting will be discussed later in this chapter.

Mutation

The mutation operator of the GGA also works with the group part of the chromosome to introduce small changes to the individual. This helps to ensure that the diversity of the population is maintained and that the population does not stagnate or converge too early. There are numerous ways in which this can be done for the GGA, such as described by Falkenauer (1998). However experiments conducted by Brown and Sumichrast (2003) have highlighted that for the GGA the mutation operator does not significantly improve the performance of the algorithm. This is primarily because the GGA cross-over procedure can introduce significant and random changes during cross-over i.e. steps 2-4 remove any duplicate objects belonging to groups and reallocates them using problem specific heuristics. This means that the offspring solutions can potentially be quite different from either of their parents, therefore ensuring new genetic material is introduced, preventing premature convergence and maintaining diversity. Furthermore the MOGGA also uses a diversity preservation strategy (will be discussed in detail later in the chapter) to maintain an archive of diverse solutions, ensuring that diverse genetic material remains in the population. For the modular design problem a number of tests have been done (in chapter 7) from which it has been concluded that mutation does not improve performance of the proposed MOGGA – it performs just as well without it. Thus the mutation operator is not used in the MOGGA.

5.4. Handling of Constraints: the Repair Heuristics

As the module grouping problem contains numerous constraints there needs to be an efficient method to deal with the issue. There are in fact numerous strategies that could be applied as was briefly discussed in chapter 3.

A two step approach has been adopted. This first step is to encourage the formation of feasible modules during mating and initialisation. This is done by applying a penalty to the fitness values of solutions that violate the constraints. In this way these 'infeasible' individuals would be less likely to be created in the first place.

However, there are situations when infeasible solutions will still be created. Hence an additional repair heuristic has been created that is used to return infeasible individuals to feasible ones after initialisation and mating. This is used instead of killing the infeasible solutions. The logic follows that the infeasible solutions may actually be quite good if it were not for their constraint violations. An infeasible solution may, for example, have mostly good module groupings with only one module containing constraint violations. This solution could be 'repaired' to produce a potentially good and feasible solution. In this way the diversity of the population is maintained and potentially good solutions are allowed to stay in the population and continue to reproduce.

If after initialisation or mating any of the constraints are violated then the module/s in which the violation has accurred is flagged and the repair heuristic is triggered. The repair heuristic will attempt to reallocate the offending component/s from the 'infeasible module' to other modules or place the offending component/s into new modules so that no constraints are violated.

Step 1) From the module in which there is a constraint violation select the first $component_n$ and temporarily reallocate it into a different $module_n$

Step 2) Evaluate fitness and if there are no longer any constraint violations then end the repair procedure.

Step 3) Temporarily reallocate the selected $component_n$ into the next $module_n$ + 1 and repeat step 2.

Step 4) Repeat step 3 by placing the selected component into different modules until $module_{max}$ is reached.

Step 5) Select the next component from the 'violated module' relocate it into a different module and repeat steps 2 to 4.

Step 6) Repeat the steps 1 to 5 for all modules in which there is a constraint violation.

5.5. Random Weighted Vector Based Search

Diversity measures are effective in ensuring that a diverse spread of solutions are maintained, but will not necessarily promote the search towards the true Pareto-front. This is particularly problematic when there are a large number of objectives (many objective problems) because all solutions are likely to become locally non-dominated. In these situations the measures will ensure that the solutions are remote from each other (less crowded) yet this does not necessarily mean that they will have good proximity to the true Pareto-front. This weakness tends to lead to a stagnation of the population and thus the random vector based search will be used.

In order to perform local search during reallocation (and to generate an initial population) one needs a search vector (temporary measure of fitness) for the module groupings to creep towards. That is, during the allocation of components to groups, each component should be placed into the group that creates the best improvement in fitness according to the weighted objective search vector.

If a static weighting system were to be used then the solutions in the population may end up losing diversity as they will constantly be creeping in the same direction. As has been discussed the overriding principle of a multi-objective GA approach is to create a set of solutions that as well as being 'fit', is also as diverse as possible. Thus a random objective weighting approach is used.

At each generation the algorithm generates a set of random objective weights to form a random weighted scalar fitness function $RW_{fitness}$ which is used to provide a search vector for local search based reallocation during the mating. The $RW_{fitness}$ is also used to select suitable parents. These ideas are based upon the works of Ishibuchi et al, (2003) and (Jaszkiewicz, 2002).

To find two suitable parents for mating the algorithm uses a tournament based selection procedure to choose two 'winning' parents that achieve the highest $RW_{fitness}$. As discussed by Ishibuchi et al (2003), choosing two parents that are close to one another in the objective space will help to ensure that the produced offspring will be more appropriate for the current search vector. If dissimilar parents are chosen then the produced 'raw' offspring (before reallocation based local search) has a high chance of being very different and potentially far away from the current search vector (see figure 5.4). When the quality of the raw offspring is poor with regard to the current search vector then local search based reallocation is less likely to yield improvement in the direction of the vector.

However, by always mating similar parents there is a possibility of the population stagnating. To avoid this an appropriate tournament size should be used so there is still a probability that dissimilar parents can be mated in order to produce potentially quite different offspring. This is important because the algorithm does not use a mutation operator.

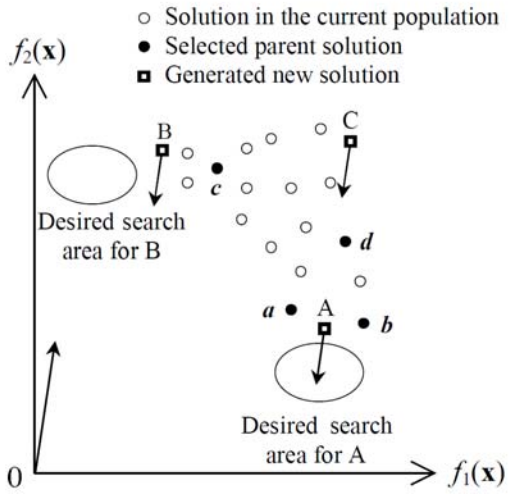

Figure 5.4. The importance of selecting appropriate parents for mating - Ishibuchi et al (2003),

5.6. Population Fitness Ranking Procedure

The MOGGA generates a whole set of alternative (optimal) modular architectures in one single run. This is done by maintaining a whole population of Pareto-optimal solutions by ranking solutions at each iteration (generation) of the search, thus ensuring that inferior solutions are replaced when better offspring solutions are found. With the MOGGA the fitness ranking of population members is done using the Pareto dominance based ranking method combined with a diversity based ranking scheme.

The dominance ranking procedure used in this research is the dominance count approach of Fonseca and Fleming (1995). The dominance count approach has been chosen as it is simple to implement and can provide a suitable means to ensure that solutions will converge towards the Pareto-front. To reiterate the concepts, Fonseca and Fleming use a dominance rank assignment which is based upon the number of solutions a particular solution is dominated by.

The diversity ranking procedure is considered important for many objective problems and will have a significant impact on the quality of modular design solutions produced by the MOGGA. Thus in this thesis a number of diversity ranking procedures have been explored and the K-nearest neighbour approach of SPEA2 has been adopted (this will be explained in chapter 7).

The MOGGA calculates the solution dominance and diversity in the following way. Firstly, each solution in the population is compared with one another and the Euclidian distance between each solution is calculated i.e. for each pair of solutions the differences in each objective achievement are found then added together, each objective distance being normalised using the maximum and minimum objective achievements in the current population.

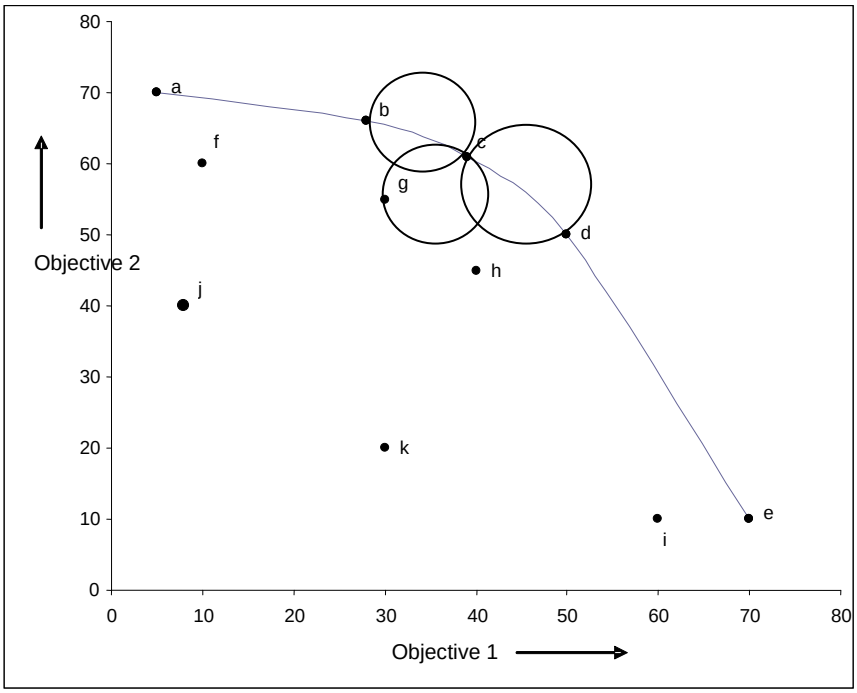

Figure 5.5. Combined population ranking of two-objective problem

The dominance count is then calculated based upon these pair-wise objective comparisons. For example, if solution x is better than solution y in all objectives (contains only positive objective distances for each objective) then solution y is dominated by solution x. For the diversity rank the three closest solutions (smallest

Euclidian distances) are flagged This can be seen in figure 5.5 where the closest three solutions to ‘c’ are ‘b’, ‘g’ and ‘d’. A proportionally lower diversity rank is then given to solutions that have closer neighbours.

Once all the dominance counts and diversity ranks have been calculated the fitness rank is then given by the normalised dominance based fitness plus the normalised diversity based fitness. All non-dominated solutions obtain a +10 to their final scores to ensure that they are not replaced by dominated solutions. After normalisation, the higher the fitness value, the higher ranking the solution is.

5.7. Preference Based Fitness Ranking

Finding a suitable Pareto-set can be computationally demanding and the sheer number of possible non-dominated solutions can be extremely high. Thus, presenting the DM with this huge number of solutions can sometimes be overwhelming. In some cases the DM may already have rough preferences on the importance of the various objectives. This information can be used to generate a more focused set of solutions. A number of researchers have advocated this approach (Deb and Saxena, 2006; Ishibuchi et al 2006).

In the MOGGA preference information can be included before the search. The preferences-based fitness assignment will be based upon the current preference values set in the objective hierarchy.

The fitness of the population members during ranking will still be dominance based as using the procedure outlined previously, i.e. non-dominated solutions, will always be favoured. However, when using the preference weighting feature of the algorithm, the non-dominated solutions will now be ranked using a combination of diversity-based fitness and the preference-based fitness. The ratio can be changed to place more or less emphasis on weighted preference and be used to create a more refined Pareto-set. These effects will be explored in chapter 7.

5.8. Dealing with Duplicate Solutions

Another problem that has not been mentioned yet is how to deal with duplicate solutions. That is solutions within the population which have exactly the same achievement for all of the objectives. The crowding distance ranking of the NSGA-II does not deal with duplicate solutions in an appropriate manner. Duplicate solutions will receive a zero crowding distance because they have zero Euclidian distance in each objective direction to the nearest neighbours. A zero crowding distance may lead to the total replacement of all duplicate solutions. This can be a problem as by replacing all duplicates a number of good solutions may be lost. It would be more desirable to keep one of the duplicates (the original) and remove the rest. In the algorithm developed all duplicate solutions actually receive a zero fitness during ranking. This is not a problem due to the incremental (one-solution-in, one-solution-out) population update procedure. For example if two duplicate solutions exist then they will both receive a zero fitness during ranking, however just one of them will be replaced during population updating. The remaining solution will no longer be a duplicate and will receive the appropriate fitness during the next iteration of the search when all solutions are ranked again.

5.9. Population Update Procedure

As already discussed the majority of previous MOGAs such as the NSGA-II and SPEA2 use a dual population approach. One population is used for storing elite solutions and one population is used for the current offspring solutions. At each generation the populations are combined and then ranked to remove the worst 50%. In this algorithm only one population is used and this is constantly updated after each mating operation.

At each generation the parent solutions are selected from the population and mated using the previously described procedure. Each produced offspring is then temporarily added to the population so that the population number becomes pop_{max}

+ 1. The offspring is always added to the pop_{max} +1 position - which is a temporary population slot only used during ranking. For example, if the population size is 50 then the new population size during ranking will be 51, with the 51st population member being the new offspring solution. After the new offspring is added to the population the ranking procedure is invoked to assess whether or not the offspring is fitter than any other member of the population. If the offspring is fitter then it will replace the weaker member and become a new member of the population. The overall fitness of solutions during ranking is based upon the combination of the Pareto-optimal based dominance and diversity fitness as described in the previous sub-section.

5.10. Generation of Initial Population: the Initialisation Heuristic

Much like the previously outlined heuristics, generating a good set of initial population members (solutions) should also be done in an intelligent manner to ensure that each solution is feasible and will have a good starting fitness. This is important as it can drastically improve the convergence speed of the algorithm. Hence an intelligent initialisation heuristic has been created for the generation of the initial population and proceeds as follows:

Start - Repeat for each initial solution until the $population_{max.}$

Step 1) Randomly generate a weighted scalar fitness function.

Step 2) Randomly generate the number of modules for $solution_n$ (between $module_{min}$ and $module_{max}$).

Step 3) Create a 'seed' point for each module by randomly allocating a component to each 'empty' module.

Step 4) Randomly select a remaining $component_n$ and temporarily allocate to the first $module_{n=1}$ and evaluate the fitness of the module groupings according to the randomly generated scalar fitness.

Step 5) Temporarily allocate the selected $component_n$ to the next $module_{n+1}$ and again evaluate the fitness using the scalar fitness function.

Step 6) Repeat step 5 until $module_{n+i}$ reaches the randomly chosen module number.

Step 7) Allocate the selected $component_n$ to the $module_{n=i}$ which showed the best fitness.

Step 8) Repeat steps 1 to 4 until every component is allocated to a module.

Step 9) Repair infeasible solutions using the repair heuristic.

End

A larger population will yield a higher level of granularity in regards to the trade-off region covered, but will present the DM with an increased number of solutions to choose from and can dramatically decrease the speed of the algorithm, and thus the population size should be chosen wisely. In this thesis the population size has been kept between 50 and 100, which has been found to produce adequate results. Increasing population size in excess of 100 has not been found to offer any significant performance advantages.

5.11. Conclusions

This chapter has presented the development of a novel MOGGA for the modular design grouping problem. During the course of the chapter is has been seen that for the development of the MOGGA a number of associated problems and limitations have had to be overcome.

The first problem is that the standard GA encoding schemes and genetic operators are not suitable for grouping problems. To overcome this problem the group-based encoding and cross-over schemes proposed by Falkenauer (1998) have been employed and some problem-specific local search heuristics have been developed.

The next issue that has been explored is how to design the algorithm to evolve a whole set of solutions rather than just one solution at a time. This has been done in a number of ways. Firstly, to enable the algorithm to explore different regions of

the search space and evolve new solutions, a random vector based selection and mating method has been employed. Secondly, to ensure that the population converges towards the Pareto-front and covers a diverse number of points along the front novel dominance and diversity ranking procedures have been developed. These procedures improve upon those employed by previous 'state-of the art' algorithms to produce a MOGA that is better designed to deal with complex many-objective grouping problems - this will be verified in the chapter 7.

CHAPTER 6

6. Software Implementation of Framework

6.1. Introduction

This chapter discusses the software implementation of the computer aided modularity optimisation framework (CAMO). Firstly an overview of the software implementation is given and then the various steps are discussed in terms of how they have been implemented with the prototype software. An important aspect of this chapter is the guidance on how each of the six modularisation objectives (indentified in chapter 4) are evaluated using the developed software.

6.2. Overview of Software

A prototype software has been created in an excel environment using VB coded macros to create a genetic algorithm (GA) based optimiser and a VB programmed user interface. The software prototype and the corresponding coding can be examined in appendix 1. In this appendix a number of excel files are included – the annotated VBA code can be seen in these files.

As highlighted in figure 6.1 the software has three main modules in which the various steps of the framework are undertaken. To reiterate these steps are: 1) product decomposition 2) interaction analysis 3) formation of modular architectures 4) scenario analysis. Throughout this chapter a vacuum cleaner based case study from the work of Ericsson and Erixon (1998) will be used to illustrate the various steps of the framework.

In the input module of the software the product is decomposed at the physical and functional levels and component interactions are entered into a number of matrices using VBA based evaluation forms to define the interaction strengths for each

modularisation objective. Once all the data has been input the optimisation module produces a set of Pareto-optimal solutions using the VB programmed MOGGA. The analysis module is then used to explore the solution set and choose the most suitable modular architecture. Again a number of VBA macros have been produced for this stage.

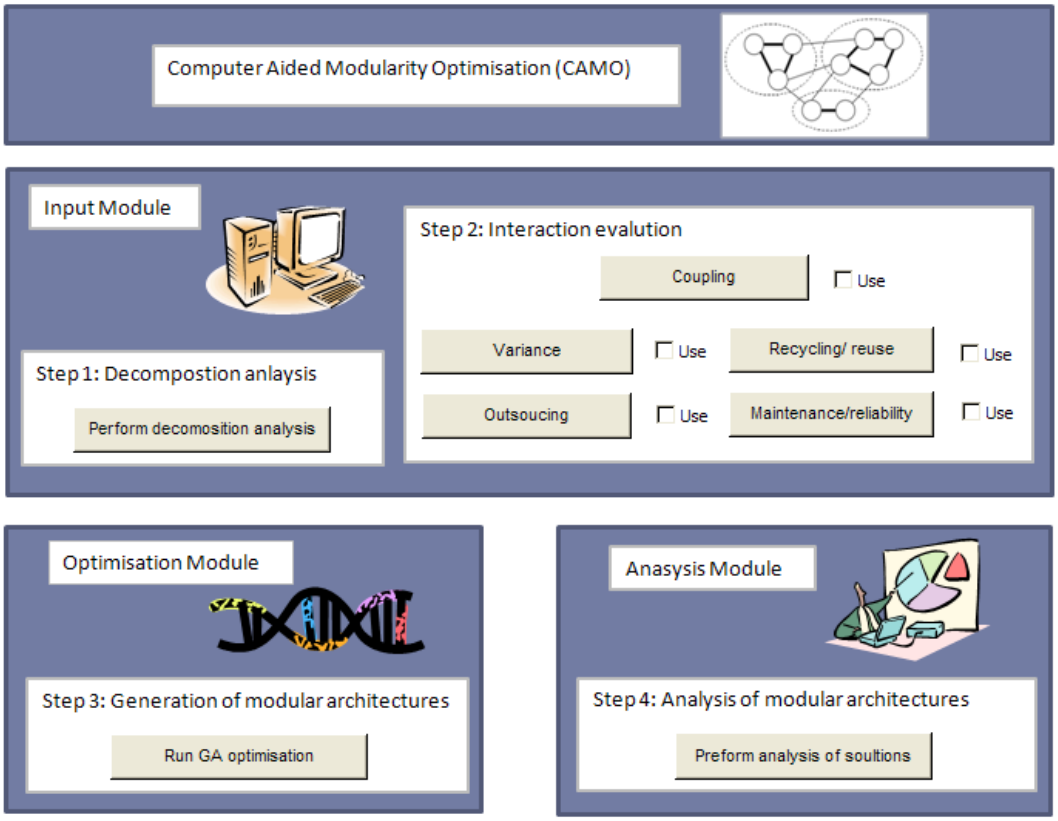

Figure 6.1. Main user menu screen of CAMO software

6.3. Step 1: Product Decomposition

Essentially, within the framework basic product components are grouped into larger product sub-systems (modules). This of course means that the basic product components must first be found by using some form of product decomposition logic. Pimmler and Eppinger (1994) suggest that the basic product components should be identified at one level of resolution lower that what one wants to achieve modules at. This is also what is advocated in the CAMO framework.

However, at the same time as indentifying basic physical components the product functions are also indentified at this step of the framework. The granularity of the functional mapping should be closely related to the customer requirement for the product. In particular the level should match the required variety mix needed in the product family.

There are several formal methods that have been created to provide a functional decomposition of the product - as discussed in the literature review. Examples are the top down approach of axiomatic design (Suh, 2001), in which the overall product function is broken down into sub-functions in a hierarchical manner, and the bottom-up approach of Stone et al (1997) in which the functional flows of the product are followed to generate a number of sub-functions. What is important is that functions should be described in 'verb-action' format given that they perform a specific product operation. They should also be solution neutral. For example, following the vacuum cleaner example, the function 'provide suction power' is a solution neutral statement and it is possible that a number of design concepts can be used to fulfil it. A cyclonic based suction engine could fulfil the function or a simpler motor and fan arrangement could also provide the function.

It is suggested that the level of functional analysis should be related to the level of granularity that most closely matches the required (or expected) customer needs for product variety. In fact there are different types of functions (common, variant and optimal) that should be identified in order to provide the required product variety mix - which will be discussed later in this chapter.

To explain how the functions are mapped within the software, consider the vacuum cleaner case study by Ericsson and Erixon (1999). In this work the system has been broken down into 24 components which can be considered functional elements in themselves. However the point here is to group the components into higher level modules more closely related to customer variety needs. So a higher level of the functional hierarchy must be looked at. In this example this is achieved using a top

down functional decomposition approach. At the top level the overall function of the vacuum cleaner is to 'remove dirt'. This has been broken down into a number of sub-functions and mapped to the associated physical components in the decomposition matrix (as in figure 6.2). When mapping the functions to components the user must enter '1' into the corresponding position and during the mapping phase it is important that all components in the matrix correspond to a function.

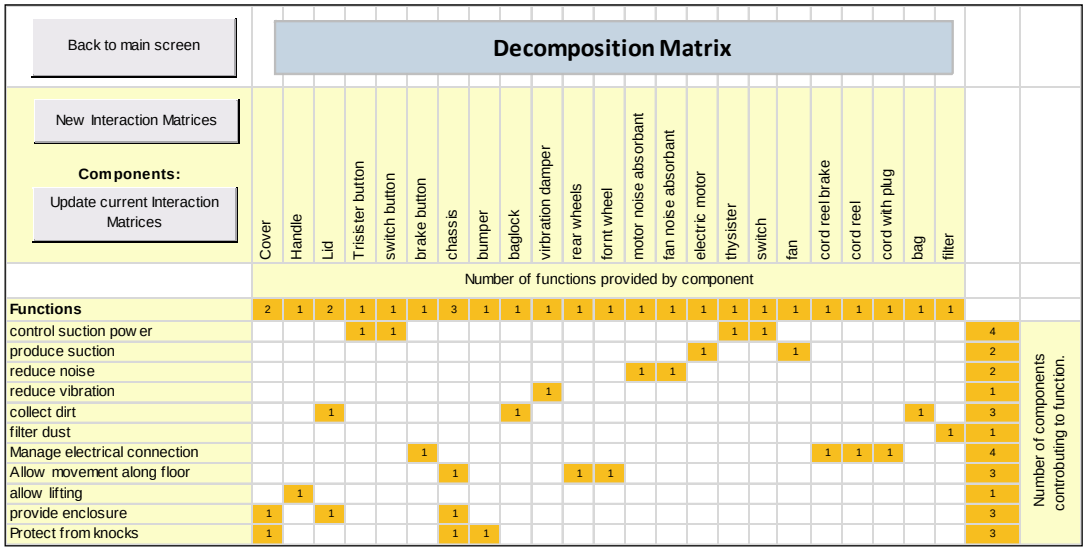

Figure 6.2. Decomposition matrix for the vacuum cleaner example.

Once the functions have been mapped within the decomposition matrix, the user must press the 'new interaction Matrices' button. The software then automatically generates a function interaction matrix as seen in figure 6.3. The software calculates the interactions between components by evaluating if the components are contributing to the same function. If they are, then a '1' is placed in the corresponding position of the interaction matrix. For example, the *electric motor* and the *fan* both contribute to the function *produce suction* so they have a '1' in the corresponding position of the interaction matrix.

Function Interaction Matrix

Components:	Cover	Handle	Lid	Trisister button	switch button	brake button	chassis	bumper	baglock	virbration damper	rear wheels	fornt wheel	motor noise absorbant	fan noise absorbant	electric motor	thysister	switch	fan	cord reel brake	cord reel	cord with plug	bag	filter
Cover	2		1				2	1															
Handle		1																					
Lid	1		2				1		1													1	
Trisister button				1	1											1	1						
sw itch button				1	1											1	1						
brake button						1													1	1	1		
chassis	2		1				3	1			1	1											
bumper	1						1	1															
baglock			1						1													1	
virbration damper										1													
rear w heels							1				1	1											
fornt w heel							1				1	1											
motor noise absorbant													1	1									
fan noise absorbant													1	1									
electric motor															1			1					
thysister				1	1											1	1						
sw itch				1	1											1	1						
fan															1			1					
cord reel brake						1													1	1	1		
cord reel						1													1	1	1		
cord w ith plug						1													1	1	1		
bag			1						1													1	
filter																							1
Functions																							
control suction pow er				1	1											1	1						
produce suction															1			1					
reduce noise													1	1									
reduce vibration										1													
collect dirt			1						1													1	
filter dust																							1
Manage electrical connection						1													1	1	1		
Allow movement along floor							1				1	1											
allow lifting		1																					
provide enclosure	1		1				1																
Protect from knocks	1						1	1															

Figure 6.3. Function interaction matrix for vacuum cleaner example

As well as setting up the function interaction matrix the software also sets up ‘blank’ matrices for each modularisation objective that has been selected for analysis. A copy of the function map is also placed at the bottom of each interaction matrix as the function map can be used to assist the user in evaluation of the various objectives - this will become apparent in the next sub-section. If at any point the user wishes to change the physical and functional map (i.e add, delete or merge physical components), then the ‘update current interaction matrices’ button should be used – the software will then automatically update all of the interaction matrices.

6.4. Step 2: Interaction Analysis

Once the basic physical components and product functions have been identified, the dependencies and similarities between components are analysed for each modularity objective and entered into a number of interaction matrices. To assist the interaction analysis evaluation forms have been developed (using VBA for excel) for each modularisation objective.

Furthermore, in the CAMO software prototype interaction analysis is made less information intensive using a number of macros to semi-automate the process. Rather than the user having to evaluate each and every interaction between components, the software is able to make certain evaluations automatically, presenting the user with only the interactions which need further human judgement.

There are two types of component interactions that must be entered into the matrices: technical interactions and strategic interactions. The technical interactions are based upon the modularisation objective of: 'loose coupling'. The strategic interactions are based upon the objectives 'variance', 'outsourcing', 'maintenance and reliability' and 'recycling and reuse'. The evaluation of each of these modularisation objectives will be discussed in the next sub-sections.

Loose Coupling

The approach advocated in this research is to use one of two component coupling evaluations to produce the interaction matrix for the *loose coupling* objective. This will depend upon the information at hand. If the product is in the conceptual stages where the joining methods have not yet been chosen or the mating complexity between components is unknown it is suggested that a simpler measure (basic coupling interaction in figure 6.4) is used to quantify component coupling. For this evaluation the user must enter the level of interaction due to three types of functional flows as well as the estimated level of interaction due to physical

coupling. For the functional flow the transfer of force is seen to provide the strongest coupling, closely followed by the transfer of materials or energy, with the transfer of information likely to cause far weaker coupling. If there is more than one type of functional flow between the components then the strongest flow should be entered.

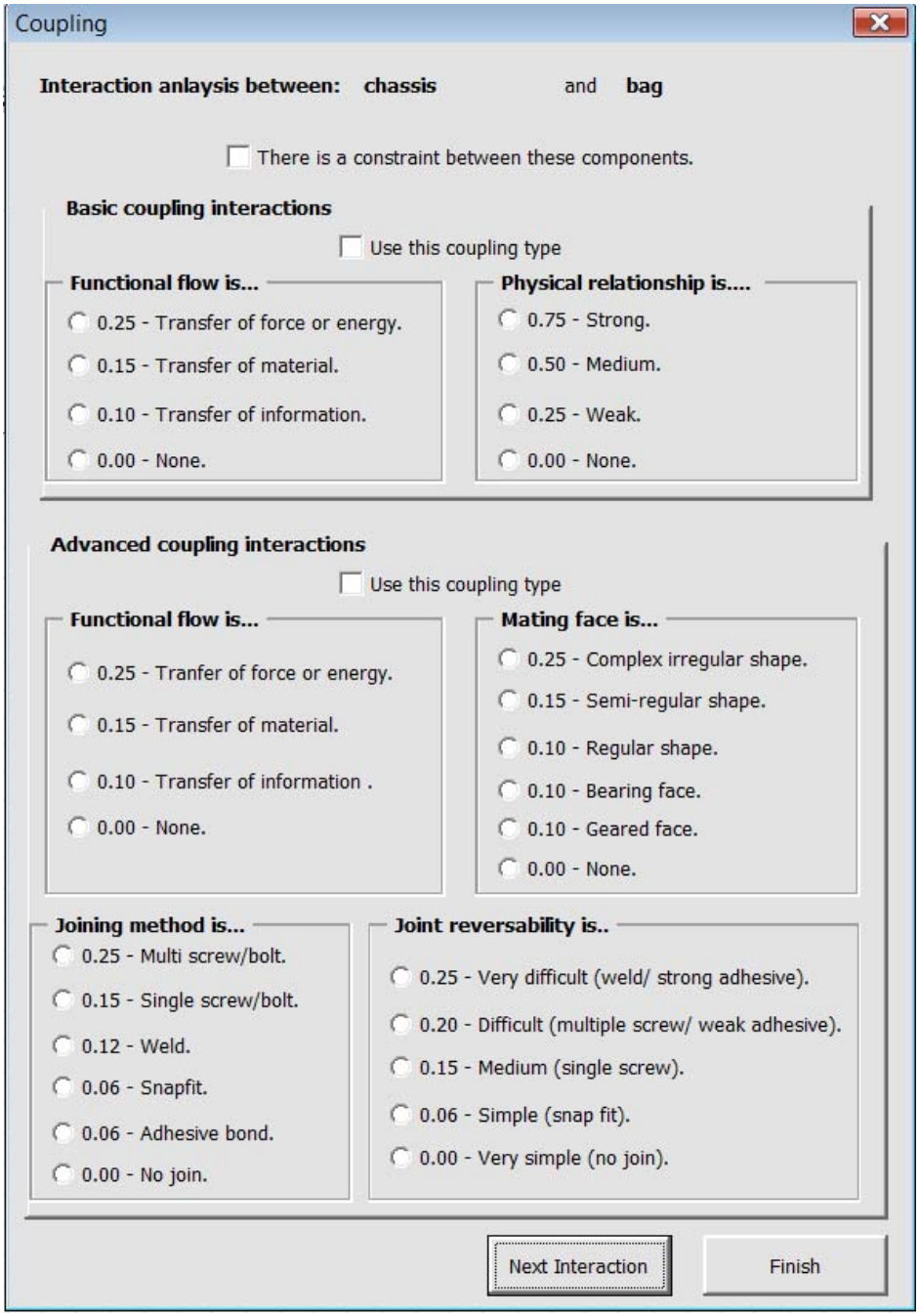

Figure 6.4. Interaction evaluation form for loose coupling objective

If sufficient knowledge is known about the possible joining methods and the level of mating complexity, then the advanced coupling evaluation can be used, which can be seen in figure 6.4. This advanced coupling evaluation is based upon four sub factors. Using the evaluation form the user must evaluate the joining method and the mating face complexity[2], the functional flows and the interface reversibility between components. The combined interaction value is then entered into the corresponding position of the module coupling matrix. For example, if two components are joined with a multi-screw (two or more screws) and the mating faces are complex (will be difficult to align with each other), interface reversibility is difficult and if there is a flow of material between them then the maximum interaction score of 1 would be entered for the component pair.

An example interaction matrix using basic coupling evaluation can be seen in figure 6.5. For example, for the handle and the cover of the vacuum cleaner there is a transfer of energy and it is estimated that there will be a strong physical coupling between the components and so the maximum interaction score of 1 is given.

[2] The evaluation factors (and guidelines) in respect to the mating faces and joining methods are based upon the work of Lai and Gershenson (2008).

Coupling Interaction Matrix

Components:	Cover	Handle	Lid	Trisister button	switch button	brake button	chassis	bumper	baglock	virbration damper	rear wheels	fornt wheel	motor noise absorbant	fan noise absorbant	electric motor	thysister	switch	fan	cord reel brake	cord reel	cord with plug	bag	filter
Cover	1	0.8	0.5	0.3	0.3	0.3		0.5															0.3
Handle	0.8	1	0.3						0.3														
Lid	0.5	0.3	1																				
Trisister button	0.3			1	0.2	0.2										0.3							
switch button	0.3			0.2	1	0.2											0.3						
brake button	0.3			0.2	0.2	1													0.3				
chassis							1	0.5		0.3	0.5	0.5	0.2	0.2	0.3			0.3		0.3		0.3	
bumper	0.5						0.5	1															
baglock		0.3							1														
virbration damper							0.3			1					0.7								
rear wheels							0.5				1												
fornt wheel							0.5					1											
motor noise absorbant							0.2						1	0.2	0.5								
fan noise absorbant							0.2						0.2	1									
electric motor							0.3			0.7			0.5		1								
thysister				0.3												1							
switch					0.3												1						
fan							0.3											1					
cord reel brake						0.3													1				
cord reel							0.3													1			
cord with plug																					1		
bag							0.3															1	
filter	0.3																						1

Figure 6.5. Loose coupling interaction matrix for vacuum cleaner example

During the coupling interaction analysis there may be components that the DM wishes to prevent grouping into the same module. For example, there may be a geometric infeasibility of joining two components as they must be kept in different physical locations of the product. Alternatively, the DM may wish to prevent certain component groupings in order to limit the search space and provide a more focused or desirable set of module groupings. To constrain components the DM must use the 'there is a constraint between these components' option in the evaluation form. Caution should be taken however as these constraints are hard – the algorithm is designed to prevent infeasible module groupings during the evolutionary search process.

Variance

Using the developed software the evaluation of the *variance* objective, and subsequent production of the corresponding interaction matrix is done by looking at all the possible product variance modes and mapping them to the corresponding components. This is done in a similar manner to the product function mapping. An example variance evaluation can be seen for the vacuum cleaner in figure 6.6.

Variance Interaction Matrix

	Components:	Cover	Handle	Lid	Trisister button	switch button	brake button	chassis	bumper	baglock	virbration damper	rear wheels	fornt wheel	motor noise absorbant	fan noise absorbant	electric motor	thysister	switch	fan	cord reel brake	cord reel	cord with plug	bag	filter
	Cover	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Handle	1	1	0.5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Lid	1	0.5	1	0	0	0	0	0.5	0.5	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0
	Trisister button	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	switch button	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	brake button	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	chassis	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	bumper	1	1	0.5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	baglock	0	0	0.5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	virbration damper	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	rear wheels	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	fornt wheel	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	motor noise absorbant	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	fan noise absorbant	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	electric motor	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
	thysister	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	switch	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	fan	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
	cord reel brake	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	cord reel	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	cord with plug	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	bag	0	0	0	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0
	filter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Type	**Variance Mode**	V	V	V	C	C	V	C	V	V	C	C	C	C	C	V	C	C	V	V	V	V	V	V
FA	suction power (produce suction)															1			1					
FA	dirt dollection capacity (collect dirt)			1						1													1	
FS	Manage electrical connection						1													1	1	1		
T	Fashion shape	1	1	1					1															
CA	Filter type																							1

Figure 6.6. Variance interaction matrix for vacuum cleaner example

Like the function mapping, variance mode mapping is done in a binary manner; a '1' is entered if the component corresponds to the variance mode. However, unlike the functional mapping, the software does not automatically produce an interaction matrix. After mapping variance modes to components the DM must then use the module evaluation form in figure 6.7 to establish the interaction matrix. During this process the DM must think carefully about whether or not certain variant modes may or may not be integrated into the same module. This will very much depend upon the required level of module mix-and-match needed to create the required product family range. Moreover, components that are not affected by any variance modes can be grouped and integrated to form common platform modules, which can be shared across the product family.

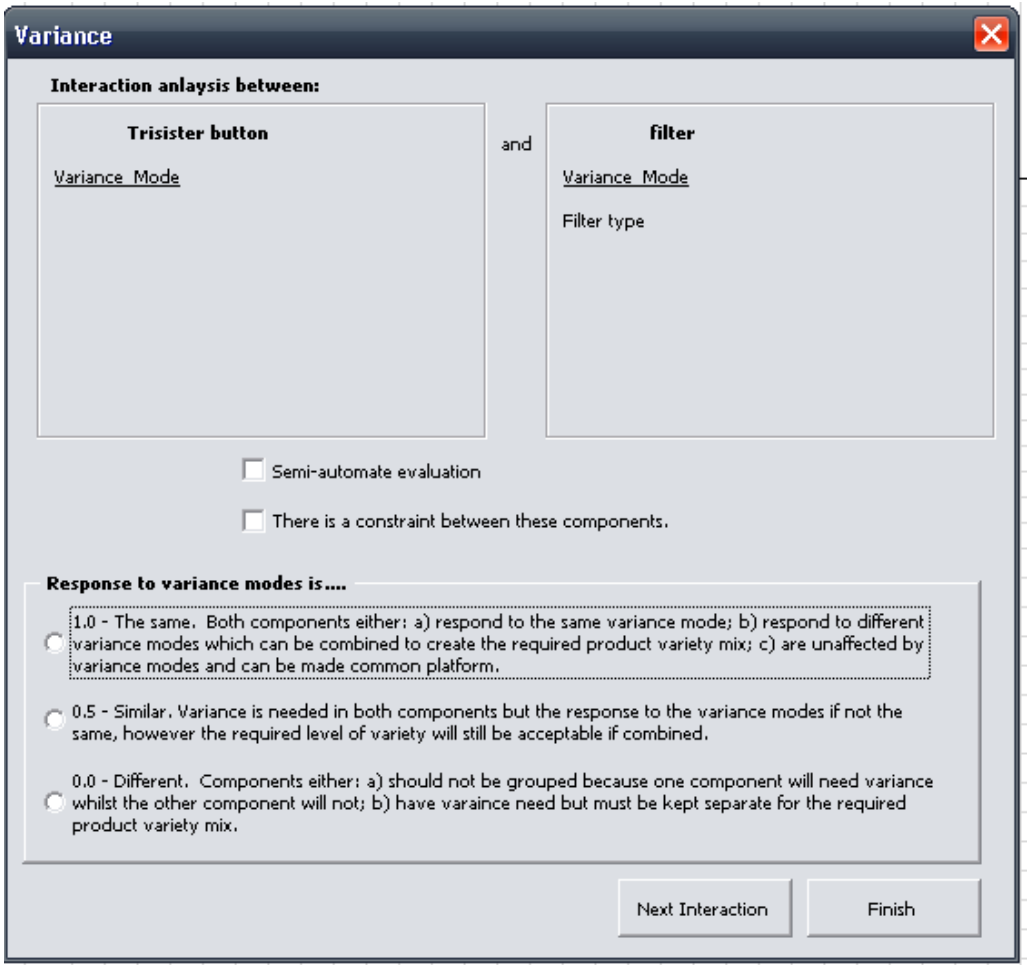

Figure 6.7. Interaction evaluation form for variance objective

During interaction analysis between each component pair, the variance modes affecting each component are displayed in the evaluation form as can be seen in figure 6.7. This will help the user determine whether or not the components should be combined based upon their response to the variance modes.

For example in figure 6.6, for the vacuum cleaner, the interaction between the 'motor' and the 'fan' is given as '1' as they both correspond to the same variance mode (i.e. both components provide the suction power variance mode). Another example of a high interaction is between the 'front wheels' and the 'chassis' - they can be integrated into a common module. The DM may also wish to prevent the combinations of certain variance modes so they can not be clustered into the same module by using the *constraint exists between these components* option in the evaluation form.

If the user wishes semi-automated interaction analysis of product variance can also be done (once all variance modes have been mapped). If this option is used the software will automatically allocate interaction scores. Components that respond to exactly the same variance modes will be allocated a high interaction score and components that are distinctly different (i.e. one component is definitely common and one is definitely a variant) will receive a low interaction score. However it is argued that the evaluations between two components that are likely to be variant yet respond to different variance modes will require expert judgement and so for these interactions the software will prompt the user for the interaction score.

As discussed by Martin and Ishii (2000) product variety can take two forms: generational variety (future variety based on future customer needs) and spatial variety (current variety based on current customer needs). Hence one must think carefully about the two types when identifying possible product variance modes. In addition it is argued that product variety can be identified across various product domains - the functional, physical and technological. These principles are important when generating the variance mode map and will thus be given some further discussion next.

Functional Domain Variance

One of the main advantages of carrying out a functional decomposition of the product/ product family is that it allows the product variety mix to be analysed at a higher level of abstraction. For example, after functional decomposition of all products in the product family the product functions can be analysed to identity common 'platform' based functions. These common platforms may not be detectable at lower levels of abstraction, such as at the physical component level.

According to Pahl and Beitz (1984) there are three different types of functions that can be used to provide the product variety mix. These are variant functions, optional functions and common functions. In this thesis these principles are used and expanded upon.

The first type of functions that need to be considered are variant functions. Further to Pahl and Beitz's work, a classification of variant functions has been made into: *variant solution functions* and *variant performance attribute functions. Variant solution functions* are functions that have a number of different technical solutions that can be used to provide the function. In figure 6.6 the *manage electrical connection* function is an example of a *variant solution function* i.e. there is one electrical winding mechanism and one mechanical winding mechanism used to provide the function. *Variant performance attribute functions* are functions that have the same technical solutions across the product family. However for these functions different performance attributes are needed to address different customer requirements. In fact *Variant performance attribute functions* are somewhat similar to the engineering metrics described by Martin and Ishii (2000) in their design for variety approach. The *provide suction power* function in figure 6.6 is an example of an identified *variant performance attribute function.* Ultimately different specification motors and fans will be needed to address the different performance requirements of the function.

Optional functions are also described by Pahl and Beitz. These are functions that are not needed in every product. That is functions of the product range that are not needed by every customer. For example, the function *indicate dirt collection level* refers to a sensor and LED, used to alert the user to when the vacuum has low suction due to the dirt collection bag being full. This is an optional extra for the standard vacuum cleaner model. As these functions are optional they are best isolated into separate modules. However if they are required in most products then it may be possible to integrate the function with common product platform functions.

Lastly, in the design of product families there will of course be common functions. For the vacuum cleaner, common functions include 'provide enclosure', 'manage movement' and 'protect from knocks'. It is possible to integrate these common functions to form a core 'common product platforms'.

Technological Domain Variance

Products often become obsolete due to fast changing technologies or fashion needs. These factors should be considered when producing the product variance mode plan. If there are any fast moving technologies present in the product, or the product is subjected to high fashion needs, then the corresponding components should be isolated into a variance module, away from common platform modules. For the vacuum cleaner for example, the cover, bumper and handle, will correspond to the 'fashionable shape' variance mode thus separation from other components should be considered.

Physical Domain Variance

During variance mode identification the variance needs of the product may already be defined at the physical component level. An example of a physical domain level product variance mode would be the different size attributes of the dust bag or different colour requirements of certain parts.

Outsourcing

The evaluation of the 'outsourcing' objective is based upon the outsourcing or in-sourcing potential of components if placed into the same candidate module. This should be done by looking at the capabilities of the suppliers that are currently being used, by considering potential new suppliers expertise and the capability of the firm's own resources For some products there may by high intellectual property (IP) content, so certain components may need to be grouped into the same module and designed/made in house. The evaluation of the objective must start with the user listing all potential suppliers (including the firm's own capability) and mapping each of them to components (as seen in figure 6.8).

Outsourcing Interaction Matrix

Components:	Cover	Handle	Lid	Trisister button	switch button	brake button	chassis	bumper	baglock	virbration damper	rear wheels	fornt wheel	motor noise absorbant	fan noise absorbant	electric motor	thysister	switch	fan	cord reel brake	cord reel	cord with plug	bag	filter
Cover	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
Handle	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
Lid	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
Trisister button	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
switch button	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
brake button	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
chassis	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
bumper	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
baglock	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
virbration damper	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
rear wheels	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
fornt wheel	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
motor noise absorbant	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
fan noise absorbant	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
electric motor	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
thysister	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
switch	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
fan	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
cord reel brake	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
cord reel	1	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
cord with plug	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
bag	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
filter	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Supplier																							
In house	1	1	1			1	1	1	1	1	1	1	1	1				1	1	1			
Supplier B				1	1										1	1	1						
Supplier C																					1		
Supplier D																						1	1

Figure 6.8. Outsourcing interaction matrix for vacuum cleaner example

Figure 6.9. Interaction evaluation form for outsourcing objective

The evaluation form should then be used (as shown in figure 6.9) to populate the interaction matrix. To add the user during evaluation the corresponding supplier information for each component is given in the form.

In addition semi-automated evaluation can be done. If this option is selected the software will allocate the highest interaction score to two components that have at least one common supplier. If the components have distinctly different outsourcability (e.g. one should be outsourced whilst the other should be in-sourced) then the software will automatically allocate the lowest interaction score.

Maintenance and Reliability

Evaluation of the maintenance and reliability modularity objective is done in a similar way to the variance objective. But instead of mapping variance modes the user must map all maintenance and failure modes. The goal of modularity then becomes to group components affected by the same maintenance and reliability modes. In is argued that two types of maintenance and failure modes can be identified: a) planned and b) potential.

Planned maintenance and failure modes are the known maintenance or replacement/upgrade operations that take place at least once during the product's lifecycle. For the vacuum cleaner these include: the replacement of dust bags and filters.

Potential failure modes are those that have been identified by means of failure modes and effects analysis (FMEA) or design failure modes and effects analysis (DFMEA[3]). If a FMEA or DFMEA has not been done, the functional analysis may provide an alternative method for potential failure mode analysis. In this way the function map can be used to identify possible product failure modes. For example for the vacuum cleaner the function 'provide suction power' may be the primary cause for a potential 'no suction' failure mode. A number of example maintenance and failure modes and their mapping to components is shown in figure 6.10 for the vacuum cleaner.

[3] DFMEA is a more simplified method that can be used at the early conceptual design stage (Kapur, 1988).

Maintenance and reliability Interaction Matrix

	Components:	Cover	Handle	Lid	Trisister button	switch button	brake button	chassis	bumper	baglock	virbration damper	rear wheels	fornt wheel	motor noise absorbant	fan noise absorbant	electric motor	thysister	switch	fan	cord reel brake	cord reel	cord with plug	bag	filter
	Cover	0	0	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	Handle	0	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	Lid	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5
	Trisister button	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	switch button	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	brake button	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	chassis	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	bumper	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	baglock	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5
	virbration damper	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	rear wheels	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	fornt wheel	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	motor noise absorbant	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	fan noise absorbant	1	1	0	0	0	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	electric motor	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	thysister	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	switch	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	fan	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
	cord reel brake	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	cord reel	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	cord with plug	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
	bag	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0.5
	filter	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.5	1
Type	**Failure Mode**																							
M	Replace Bag			1						1													1	
M	Replace filters			1																				1
F	No suction				1	1										1	1	1	1					
F	Not winding cord						1													1	1	1		
F	Filter blocked																							1

Figure 6.10. Maintenance and reliability interaction matrix for vacuum cleaner example

Once all maintenance and failure modes have been mapped to components the evaluation form (as shown in figure 6.11) must be used to populate the interaction matrix. The maintenance and failure modes affecting each of the two components is presented to the user during evaluation to make the process more straightforward.

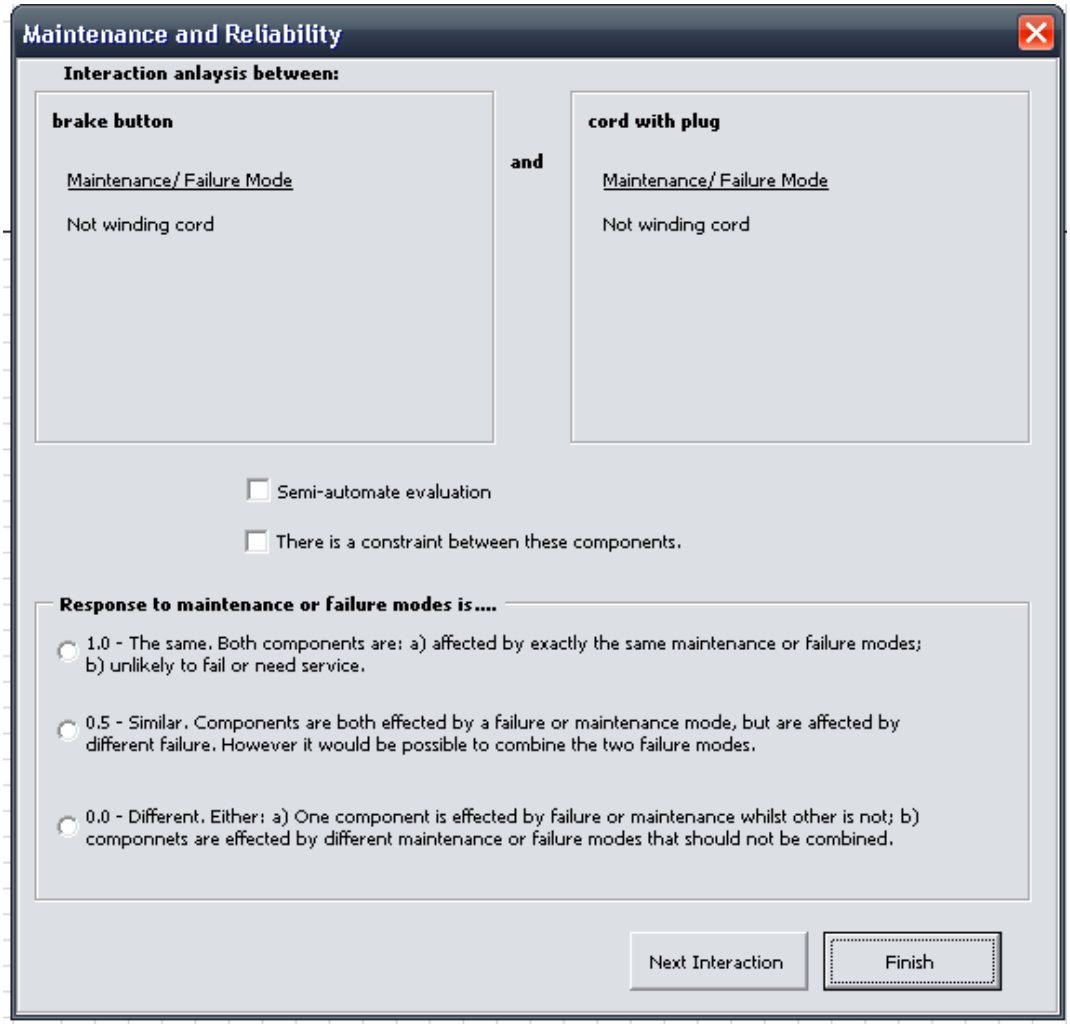

Figure 6.11. Interaction evaluation form for 'maintenance and reliability' objective

Semi-automated evaluation can also be done if required. If this option is selected the software will allocate the highest interaction score if the two components have exactly the same response to maintenance and failure modes. If the components have distinctly different responses (e.g. one component is likely to need replacement whilst other will not) then the software will automatically allocate the lowest interaction score. If two components respond to a maintenance or failure mode, but respond to different ones then the user will be promoted to evaluate these interactions as expert judgment is deemed necessary.

There is another important consideration for the maintenance and reliability objective that has to do with the number of modules. If one produces a modular structure that contains a large number of small modules then the costs of replacing

worn-out/ failed modules would obviously be lower than in the case of a module architecture with a small number of large modules. However with a larger number of modules the disassembly costs may be higher. So if maintenance and reliability are considered important for the product then the user may end up having to trade-off the two factors (coupling versus reliability and maintenance) when choosing a suitable modular architecture.

Recycling and Reuse

It can be argued that to better improve the end-of-life management of products both recycling and reuse should be used when developing a modular product structure. Hence for this research a combined 'recycling and reuse' modularisation objective has been developed. The evaluation form can be seen in figure 6.12.

Before evaluation begins however, one must think hard about the overall end-of-life scenario for the product and what is the overall benefit that modularisation of the product can provide. Questions that should be asked are what type of recycling and reuse will the product undergo? Does the product contain valuable material making manual disassembly to module level worthwhile? Must certain components be removed prior to recycling to conform to environmental legislation? Or are there certain high value components/modules that can be removed and reused?

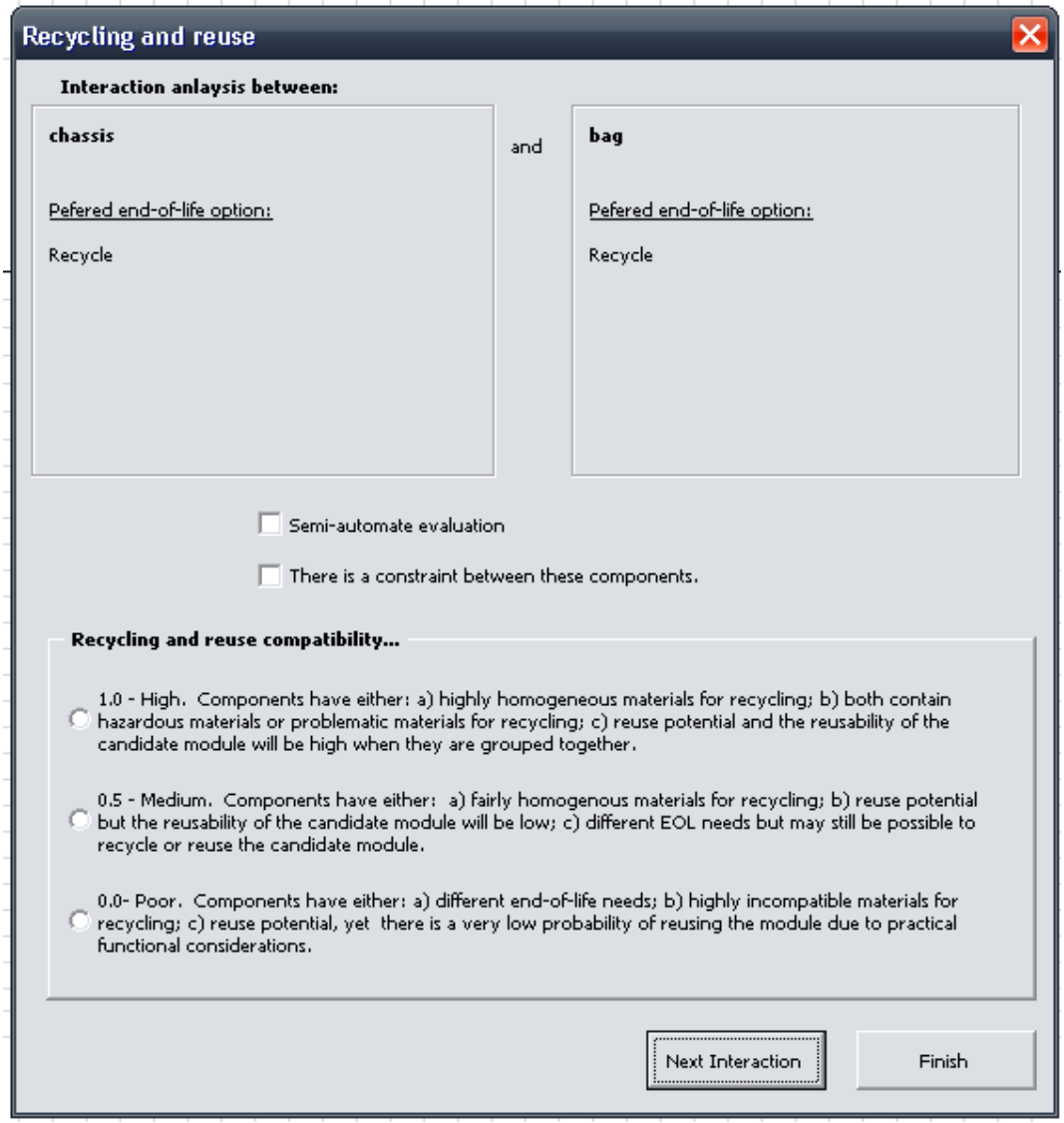

Figure 6.12. Interaction evaluation form for recycling and reuse objective

To start the evaluation the user should create an end-of-life component map by entering whether the components are: a) reusable; b) recyclable; or c) contains hazardous/hard to recycle materials). This mapping is done in a binary manner like the previous objectives (example shown in figure 6.13). Note this approach allows for the possibility that a component can be both reusable and recyclable.

Once the end-of-life map has been created the interaction evaluation form should then be used to populate the interaction matrix. During the interaction analysis of component pairs the software will highlight the preferred end-of-life option for each component pair to make the evaluation easier.

Recycling and reuse Interaction Matrix

Components:	Cover	Handle	Lid	Trisister button	switch button	brake button	chassis	bumper	baglock	virbration damper	rear wheels	fornt wheel	motor noise absorbant	fan noise absorbant	electric motor	thysister	switch	fan	cord reel brake	cord reel	cord with plug	bag	filter
Cover	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	1	0
Handle	0	1	1	0	0	1	0.5	0.5	0.5	0.5	0.5	1	0	0	0	0	0	0	1	1	1	1	1
Lid	0	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
Trisister button	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
switch button	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
brake button	0	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
chassis	0	0.5	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
bumper	0	0.5	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
baglock	0	0.5	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
virbration damper	0	0.5	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
rear wheels	1	0.5	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
fornt wheel	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1
motor noise absorbant	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
fan noise absorbant	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
electric motor	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
thysister	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
switch	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
fan	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
cord reel brake	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	1	1	1	0
cord reel	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0
cord with plug	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0
bag	1	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0
filter	0	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1
EOL Option																							
Reuse	1	1		1	1										1	1	1	1					
Recycle	1	1	1			1	1	1	1		1	1							1	1	1	1	1
Dispose										1			1	1									

Figure 6.13. Recycling and reuse interaction matrix for vacuum cleaner example

Like the previous objectives a semi-automated evaluation can also be done if required. However semi-automated evaluation is somewhat more limited for this objective. It is argued that expect judgment is needed to evaluate whether or not two components can be recycled or reused together in the same module – which may depend upon a number of factors such material homogeneity and suitability for reuse. Therefore the software will only automatically evaluate interactions between components that have distinctly different end-of-life needs. For example, one component needs disposal whilst the other can be recycled.

6.5. Step 3: Formation of Modular Architectures

In this step of the framework the specially designed multi-objective grouping genetic algorithm (MOGGA) is applied to find a whole set of optimal (alternative) modular architectures through manipulation of the interaction data in the various matrices. In a single optimisation run the MOGGA is able to produce a whole set of Pareto-optimal solutions that represent a good coverage of the trade-off surface i.e. a range of solutions, each with different combinations of objective achievements. The user can set up a number of parameters before running the algorithm as can be seen in figure 6.14. An important point that must be noted is the preference rate setting. If set above zero then the generated Pareto-optimal set will be directed towards the preferences set in the 'modularisation objective hierarchy' as was discussed in previous chapters. This may or may not be desirable for the user.

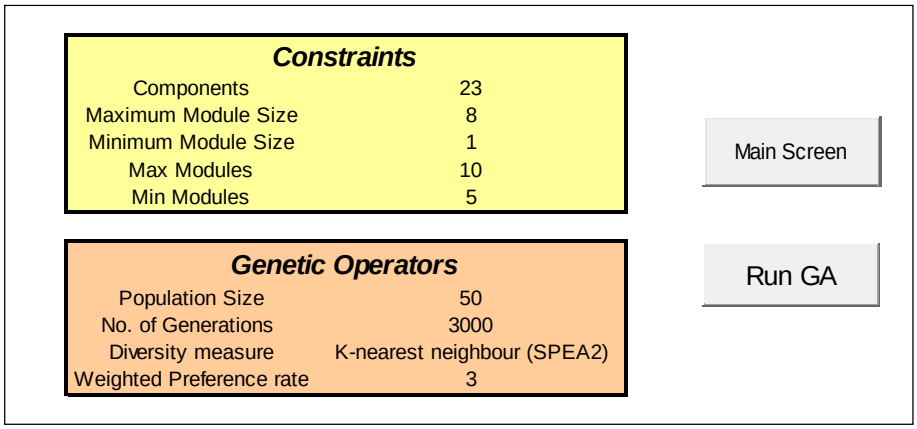

Figure 6.14. The constraints and parameter settings for the MOGGA

6.6. Step 4: Analysis of modular Architectures

To support the solution exploration stage a product modularisation objective hierarchy has been developed (as seen in figure 6.15). As was discussed in chapter 5 the hierarchy is based upon the principles of the analytical hierarchical process (AHP).

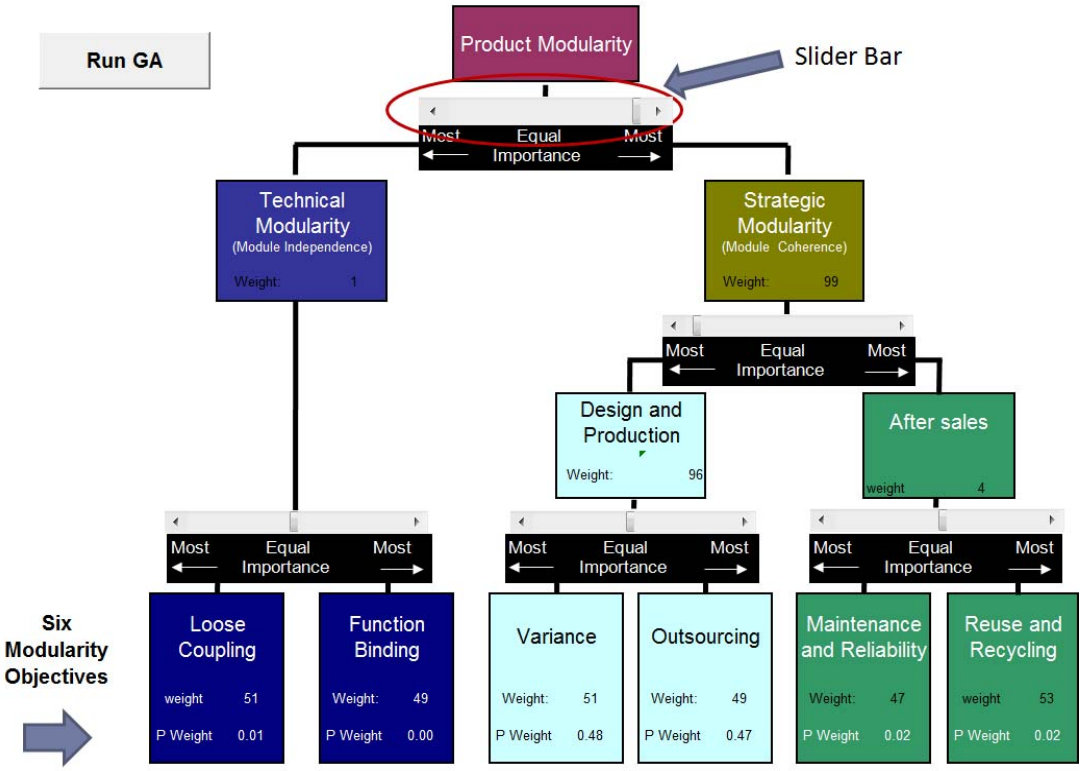

Figure 6.15. The modularisation objective hierarchy

By changing the preferences at the various levels of the hierarchy corresponding solutions can be visualised in real time. These corresponding 'best' solutions are visualised using radar plots. By exploring the solutions in this manner the decision maker is then able to make a more informed decision on the most suitable modular structure for the product.

For example, figure 6.16 shows the four best solutions corresponding to the current preferences in the hierarchy. In this example, the DM has given a greater preference to strategic modularity and in particular the design and production phase.

Figure 6.16. Corresponding radar plots of 'best' solutions for design and production phase preferences

During any stage of this exploration process the DM can choose to make one of the solutions the 'benchmark' solution which can then be compared to other solutions to explore the trade-offs. The benchmark solution is displayed as the red solution in the radar plots.

Several macros have also been written to provide a visualisation of the modular structure in matrix format, such that analysis and redesign/ improvement can be made easier. In figure 6.17 an optimal modular structure for the vacuum cleaner is shown. Combinations of the module independence and module coherence based objectives can be displayed at the same time.

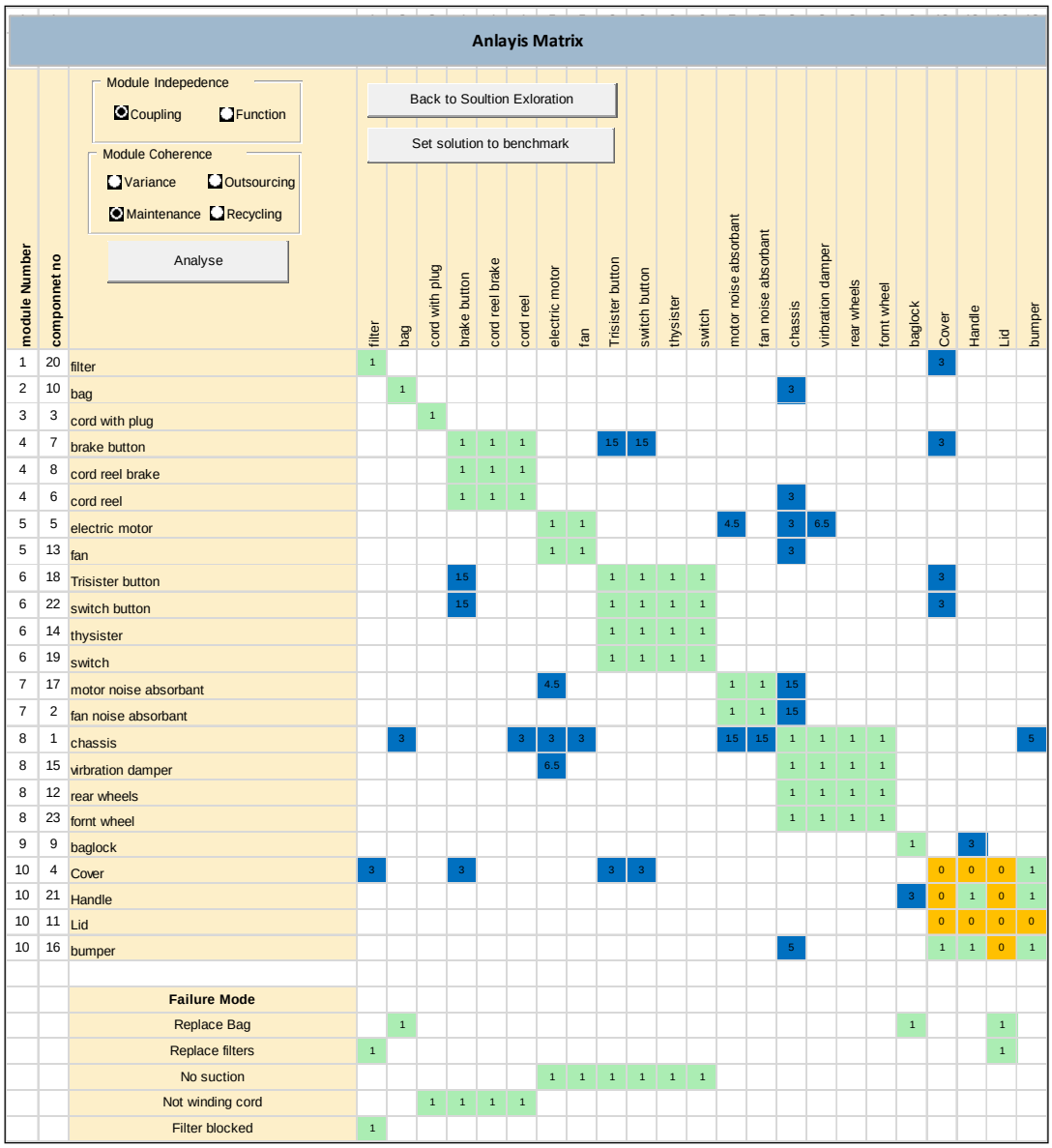

Figure 6.17. Analysis of the chosen modular solution for the vacuum cleaner.

In figure 6.17 the loose coupling (module independence) and maintenance and reliability objectives (module coherence) are shown. Module independence (between module) interactions can be seen as the various shades of blue (the stronger the interaction the darker). The module coherence (within module) interactions are seen as the tones between orange and green, with the orange colours denoting problem interactions (that should be improved) and green colour showing less problematic interactions. In addition to the main interaction matrix the rows underneath the matrix (the maintenance and failure modes here), may also highlight modules that may need further consideration/ redesign i.e the modules

that may need to be removed for maintenance and replacement. These modules should be placed first in the disassembly sequence for ease of removal, and/or the coupling between these modules should be reduced if possible. For example, modules 6 and 7 should be made easy for disassembly and re-assembly as they are likely to need maintenance/ replacement during the product's life.

6.7. Conclusions

This chapter has presented the software implementation of the developed CAMO framework. The major novelty of the framework is that it presents a true multi-objective approach to product modularisation. This is achieved by the coding of a state-of-the art MOGGA and through the production of evaluation guidance and software evaluation forms for each of the six indentified modularisation objectives. This ultimately provides a more holistic modularity optimisation framework.

CHAPTER 7

7. Algorithm Testing

7.1. Introduction

In this chapter the MOGGA's performance will be tested to verify that it is capable of finding optimal solution sets for the modularisation problem. Firstly, a number of diversity ranking schemes will be tested. Secondly, the chapter will look at performance compared to an aggregated objective approach. Thirdly, the influence of the mutation operator will be looked at. Lastly, the effects of including user defined objective preferences will be evaluated.

For the various tests the optimisation of a vacuum cleaner example will be used. It must be stated that the purpose of this chapter is to highlight the effectiveness of the algorithm in dealing with a multi-objective combinatory optimisation problem and not to assess the actual modularisations of the vacuum cleaner.

7.2. Metrics for Measuring the Performance of the MOGGA

To test the performance of diversity ranking schemes and the influence of the mutation operator, the solution sets which they generate can be compared and contrasted using some of the commonly used multi-objective algorithm performance metrics. There are generally two properties that are measured:

a) the level of convergence towards the Pareto-front.
b) the level of diversity or spread along the Pareto-front.

There are numerous performance measures that can be used. Knowles et al (2006) performed a comparison of which measures are the most suitable. They recommend the use of the hypervolume measure, the R-metric and the Epsilon indication. However that later two of these metrics require a reference set of established

Pareto-optimal points and in the case of the test problem these points are unknown. Thus only the hypervolume measure will be used in the analysis of the algorithm.

The Hypervolume Measure

Perhaps the single most popular and useful means of assessing the quality of solution sets is to calculate the hypervolumes of the solution space that each set covers. The hypervolume is the area of the solution space that is covered by points of the Pareto-optimal set, as illustrated in figure 7.1. The solution set with the largest hypervolume coverage indicates the best performing algorithm. The hypervolume measure has in fact become an almost standard performance indicator within the field of MOGAs. This is due to the fact that the hypervolume provides an effective means of measuring both the level of convergence and diversity (spread) of solutions.

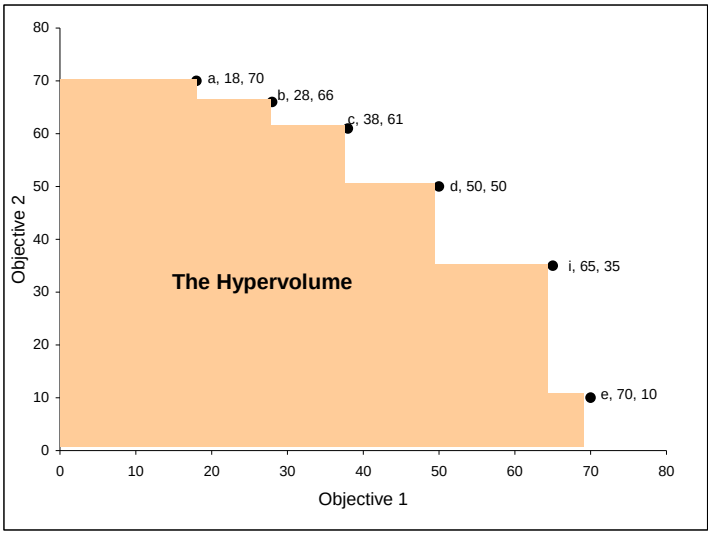

Figure 7.1. The hypervolume measure for example two-objective problem

For calculation of the hypervolume, the coding has been taken from the Pisa GA optimisation suite from Zilter's research group web site (Pisa source code, 2010). With this code the lower the measure the better the performance of the algorithm i.e. the greater the hypervolume (solution space) the solution set covers.

7.3. Comparison of Different Diversity Ranking Methods

The algorithm has been uniquely designed for the multi-objective product modularisation problem and therefore it is difficult to make a direct comparison to well known MOGAs such as NSGA-II or SPEA2. These algorithms will of course perform poorly for the product modularisation problem as they are not designed for combinatory optimisation problems i.e. they do not have suitable genetic operators and encoding schemes (they are not group based). Hence for practical reasons the raw NGSA-II or SPEA2 will not be directly compared to the algorithm developed in this research. However comparisons will be made with the diversity ranking procedures of these algorithms. To ensure a good coverage of the search space the diversity ranking procedure is a vital component of a MOGA. Especially for a 'many objective' problem, as most solutions in the Pareto-set will become locally non-dominated as the number of objectives increases (Corne and Knowles, 2007; Ishibuchi et al, 2008a).

To compare the various diversity measures the developed MOGGA has been modified to use different diversity ranking schemes. In total four different diversity ranking procedures have been coded and are given in appendix 1. The first is the 'original crowding distance' diversity ranking of the NSGA-II. The second is the 'modified crowding distance' ranking. The third is the 'k nearest neighbour' approach of the SPEA2 and lastly the 'Average Ranking' method outlined by (Corne and Knowles, 2007).

With the crowding distance measure extreme solutions are assigned a higher fitness (infinity value) so they remain in the population during ranking. However, this means that some solutions that are not globally optimal may end up remaining in the population, even when better solutions (better in the majority of objectives) are found. Thus, a modified version of the crowding distance, that assigns a suitably lower fitness value (not infinity) to extreme solutions, has also be coded.

As can be seen in table 7.1 the results of the hypervolume test indicate there is no significant difference in the different diversity methods, apart from the average rank approach which shows a slightly poorer performance. These results differ from the results reported by (Corne and Knowles, 2007), who reported that the average rank produced better results. This is because the diversity measure in their work was the primary mechanism for pushing the search space towards more optimal points along the Pareto-front.

However, with the MOGGA the diversity measure merely serves to ensure that only better solutions are kept during the population update procedure. The push towards more new optimal points is done with the local search and the random vector search implemented in the MOGGA meaning that new areas of the search space are constantly being explored. In fact, these results are in agreement with the recent works of Ishibuchi et al (2008b) who integrated a local search heuristic with the NSGA-II and reported an improvement of the performance of the algorithm when dealing with many objective problems.

Table 7.1. Hypervolume results for optimisation runs with different diversity measures at population size of 50 and generation size of 5000

Run number	Crowding distance (original NSGA-II)	Crowding distance (modifiedl)	K-nearest neighbour (SPEA-2)	Average rank
1	-0.71	-0.71	-0.71	-0.67
2	-0.70	-0.71	-0.70	-0.67
3	-0.71	-0.71	-0.71	-0.67
4	-0.72	-0.70	-0.71	-0.69
5	-0.70	-0.70	-0.70	-0.68

For the developed MOGGA it can be concluded that the K-nearest neighbour method of SPEA2 is the best approach as it is slightly simpler to code and fits better with the existing coding used for the calculation of the dominance counts.

7.4. Influence of Mutation Operator

The mutation operator is used to introduce an element of randomness to the search process. There are numerous ways in which the mutation operator could be designed as outlined by Falkenauer (1998). To ensure it is consistent with the local search reallocation heuristic that has been advocated in this research, the mutation operator works in the following way and is applied straight after mating. It works by eliminating a small percentage of module groupings and then re-allocating the missing components to new/existing modules.

Step 6) Randomly select a number of modules. The number of modules selected must not exceed one quarter of $module_{max..}$

Step 7) Delete all components occurring in the randomly selected groups.

Step 8) Randomly select a missing $component_n$ and temporarily allocate to first $module_{n=1}$ and evaluate the fitness of the module groupings according to the random weighted scalar fitness function $RW_{fitness}$ used during mating.

Step 9) Temporarily allocate the selected missing $component_n$ to the next $module_{n+1}$ and again evaluate using random weighted scalar fitness function $RW_{fitness}$.

Step 10) Repeat step 2 until $module_{n+i}$ reaches $module_{max.}$

Step 11) Allocate the selected $component_n$ to the $module_n$ which showed the best fitness.

Step 12) Repeat steps 1 to 4 until each missing component is allocated to a module.

As can be seen by the results in table 7.2 the mutation operator does not make any difference to the performance of the MOGGA according to the hypervolume metric. Thus it can be concluded that the mutation operator is not necessary for the MOGGA. In fact, it is quite logical that this is the case. For single objective algorithms the operator is of course important to introduce a level of randomness to the search and prevent it converging on an optimum too soon. However with the

multi-objective approach there are several mechanisms in place to ensure that diversity is maintained and the whole point of the multi-objective approach is to maintain a whole set of different solutions. Thus there is plenty of different genetic material to choose from during the mating operation and thus new solutions will continue to be produced without the need for a mutation operator.

Table 7.2. Hypervolume results for optimisation runs with mutation and no mutation at population size of 50 and generation size of 5000

Run number	Mutation operator	No mutation operator
1	-0.71	-0.71
2	-0.70	-0.70
3	-0.71	-0.71
4	-0.71	-0.72
5	-0.71	-0.71

7.5. Comparisons to Aggregated Objective Approach

In this section the MOGGA will be compared to an aggregated objective optimisation (objectives are weighted and aggregated to form a single optimisation goal) method using dominance rank and hypervolume measures. Because the search space is large with the 'many' objective modularisation problem there is a risk that some of the solutions produced by the MOGGA may not be quite as optimal as the single solutions produced with a traditional aggregated objective algorithm. This is because the aggregated objective algorithm is focussing on only one search region at a time as opposed to exploring multiple search regions simultaneously. To provide the user with optimal product architectures it is important that the MOGGA is able to find the same or at least similar Pareto-optimal points as an aggregated objective approach is able to find.

What is of uppermost importance is that the reference solutions produced by the aggregated objective algorithm do not dominate (better) the solutions produced by

the MOGGA as this will indicate the MOGGA being a weaker optimisation algorithm.

To make this test, a number of modularity optimisations have been performed using an aggregated objective algorithm to produce a set of reference solutions by applying different weight combinations for the objectives and re-running the algorithm (population size of 50 and generation size of 5000). The reference set will then been compared to the solution set produced by the MOGGA. If the comparison solution is worse than a reference solution in all objectives then it is said to be dominated. Likewise if any of the MOGGA generated solutions are clearly dominating solutions in the reference set then the MOGGA can be seen to be outperforming the aggregated algorithm.

The aggregated objective algorithm used in the test has the same group based encoding scheme and generic operators as described in the previous chapter and uses the mutation operator described in the previous section. The fundamental difference between this algorithm and the MOGGA is that only one single solution is produced in each run, rather than a Pareto-set.

Figure 7.2 shows the first 10 (out of 50) reference solutions that have been generated using the aggregated objective algorithm and the first 10 of the solutions produced by the MOGGA. A macro has been written to do the dominance comparisons between the two sets. The total number of MOGGA generated solutions that are dominated by the reference set is shown along with the total number of reference solutions that are dominated by MOGGA solutions. In the case of figure 7.2 no solution in the reference set clearly dominates any of the solutions produced by the MOGGA. Neither do any of the MOGGA solutions dominate the reference solutions.

20	**Reference Solutions**	1	2	3	4	5	6	7	8	9	10
Objective	Coupling	0.88	0.90	0.77	0.82	0.80	0.78	0.81	0.82	0.79	0.82
	Functions	0.90	0.75	1.00	0.82	0.85	0.82	0.80	0.92	0.92	0.85
	Variety	0.68	0.55	0.62	0.82	0.92	0.97	0.60	0.62	0.67	0.60
	Outsourcing	0.77	0.80	0.78	0.86	0.93	0.91	0.98	0.93	0.93	0.85
	Maintenance and Reliability	0.66	0.61	0.76	0.84	0.84	0.90	0.74	0.88	0.92	0.77
	Reuse and recycling	0.77	0.80	0.95	0.93	0.80	0.86	0.84	0.96	0.90	0.99
	Solution Dominated by:	_	_	_	_	_	_	_	_	_	_
	Total number of solutions reference set dominated by:	0									
	Closest Euclidean distance	0.4863	1.38	0.45	0.33	0.49	0.4	1.25	0.39	0.45	0.74
	Closest comparison Solution	16	16	3	14	11	11	2	10	2	18
Objective	**Comparison set**	1	2	3	4	5	6	7	8	9	10
	Coupling	0.77	0.81	0.77	0.73	0.68	0.71	0.79	0.86	0.76	0.84
	Functions	0.90	0.92	1.00	0.72	0.77	0.75	0.95	0.82	0.95	0.92
	Variety	0.79	0.63	0.58	0.85	0.87	0.98	0.74	0.82	0.76	0.60
	Outsourcing	0.80	0.89	0.80	1.00	0.98	0.97	0.75	0.85	0.64	0.90
	Maintenance and Reliability	0.81	0.87	0.72	0.92	0.85	0.94	0.74	0.75	0.72	0.83
	Reuse and recycling	0.96	0.88	0.88	1.00	0.78	0.98	0.82	0.75	0.79	0.95
	Solution Dominated by:	_	_	_	_	_	_	_	_	_	_
	Total number of solutions comparison set dominated by:	0									

Figure 7.2. Comparison of aggregated algorithm generated reference set versus a Pareto-optimal set produced by MOGGA

A further 10 dominance comparisons have been carried out with different MOGGA sets (results seen in table 7.3) and in only one test (out of 10) did any of the reference set solutions dominate the solutions produced by the MOGGA. This shows that the aggregated objective does not outright outperform the MOGGA.

Table 7.3. Comparison of aggregated algorithm generated reference set versus a Pareto-optimal set produced by MOGGA

MOGGA set number	Number of solutions Reference set dominates	Number of solutions MOGGA set dominates
1	1	0
2	0	1
3	0	0
4	0	0
5	0	0
6	0	1
7	0	0
8	0	0
9	0	0
10	0	0

However, it may still be argued that the reference set generated by the aggregated objective algorithm may still provide a better Pareto front (optimal solution space) coverage. Thus the hypervolumes of each of the previously generated MOGGA solution sets has also been calculated and compared to the reference set (see table 7.4). These results show that in fact the solutions produced by the MOGGA are comparable to the reference set generated by the aggregated objective.

Table 7.4. Hypervolume results for MOGGA sets (population size of 50 and generation size of 5000) and Reference set.

Algorithm	Hypervolume
Reference set	-0.72
MOGGA set 1	-0.71
MOGGA set 2	-0.73
MOGGA set 3	-0.70
MOGGA set 4	-0.71
MOGGA set 5	-0.70
MOGGA set 6	-0.72
MOGGA set 7	-0.69
MOGGA set 8	-0.71
MOGGA set 9	-0.70
MOGGA set 10	-0.72

These two tests provide verification of the effectiveness of the MOGGA at finding a good set of Pareto-optimal solutions. Furthermore, it has also confirmed that the production of the reference set (i.e. coverage of optimal solution space) with the aggregated objective algorithm was time consuming and tedious. In contrast the MOGGA algorithm is simply left to run once and will produce a suitable set of Pareto-optimal solutions that can be further explored by the user to conduct a 'what-if' scenario analysis.

7.6. Effects of Preference Weighting

Lastly this chapter will also look at the preference weighting effects on the production of Pareto-optimal sets. The point here is for the DM to be able to specify some loose preferences before the search, in order to narrow down the search area and present the DM with a more focussed Pareto-optimal set. The DM may wish to use this approach only after an initial exploration of the solution set (without any preferences set). Then once better informed about the compromises that will be needed, the preferences can be set to produce a more refined set for scenario analysis.

The point of this section is thus to explore the ability of the algorithm to produce Pareto-optimal solution sets that match rough user preferences. This has been done by making visual comparisons via box plots. Box plots have been advocated by numerous researchers (Zitzler and Thiele, 1999, Deb, 2001) as a suitable means to visualise the objective ranges of solutions in order to make rough comparisons between the Pareto-optimal sets produced by MOGAs.

The results of a number of tests, where different preferences have been used can be seen in figures 7.3 - 7.6. In these figures the preference weights (taken from the objective hierarchy) are shown in the bottom right corner of each box plot graph. As can be seen the range of objective values and mean values of the solution set shift to reflect the objective preferences given in each case. For example, in figure

7.3 the preferences have been placed on the technical modularity so it can be seen that the box plot ranges are higher for the loose coupling and functional binding objectives, although at the expense of lower range values of the other strategic modularity objectives.

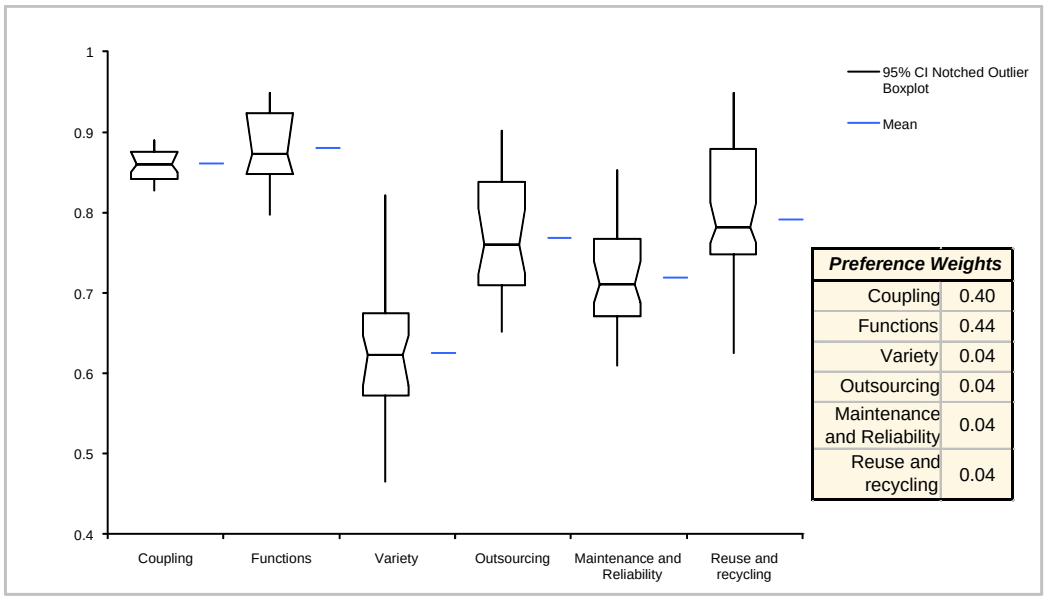

Figure 7.3. Box plots showing a preference placed on technical modularity

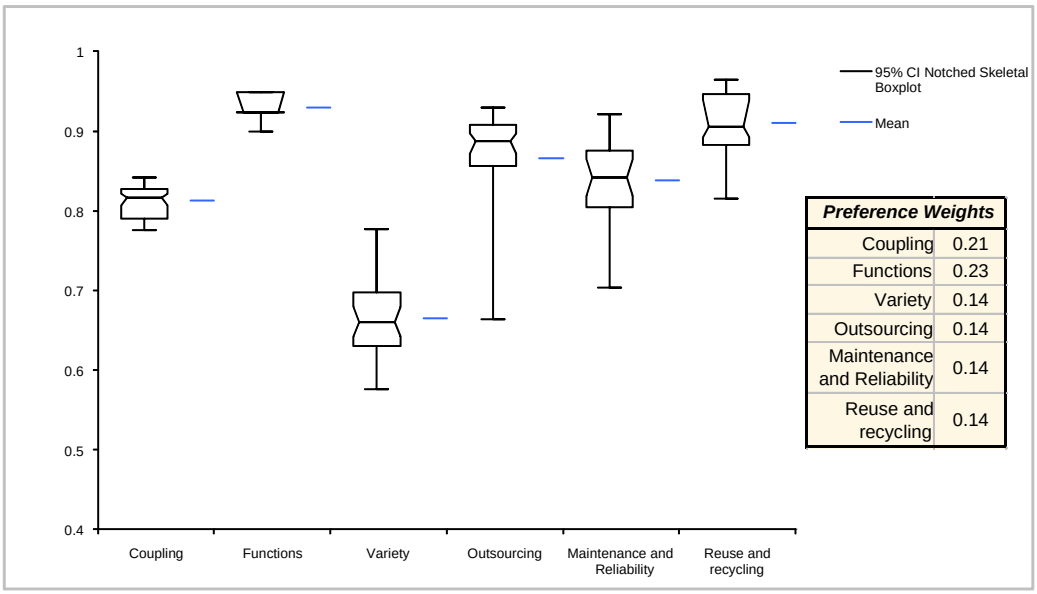

Figure 7.4. Box plots showing equal preference placed on technical and strategic modularity

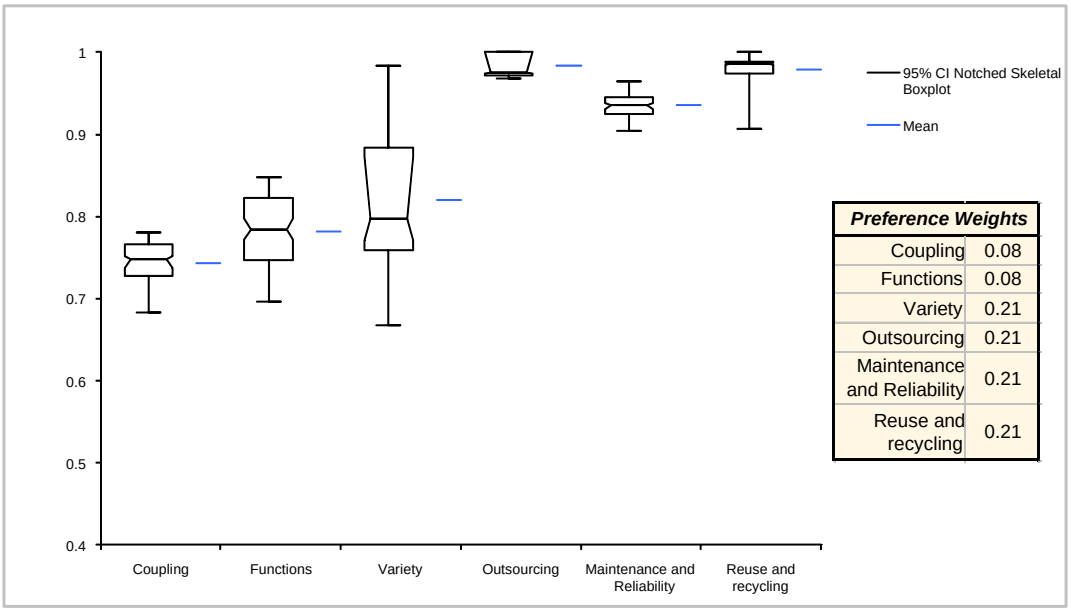

Figure 7.5. Box plots showing a preference placed on strategic modularity

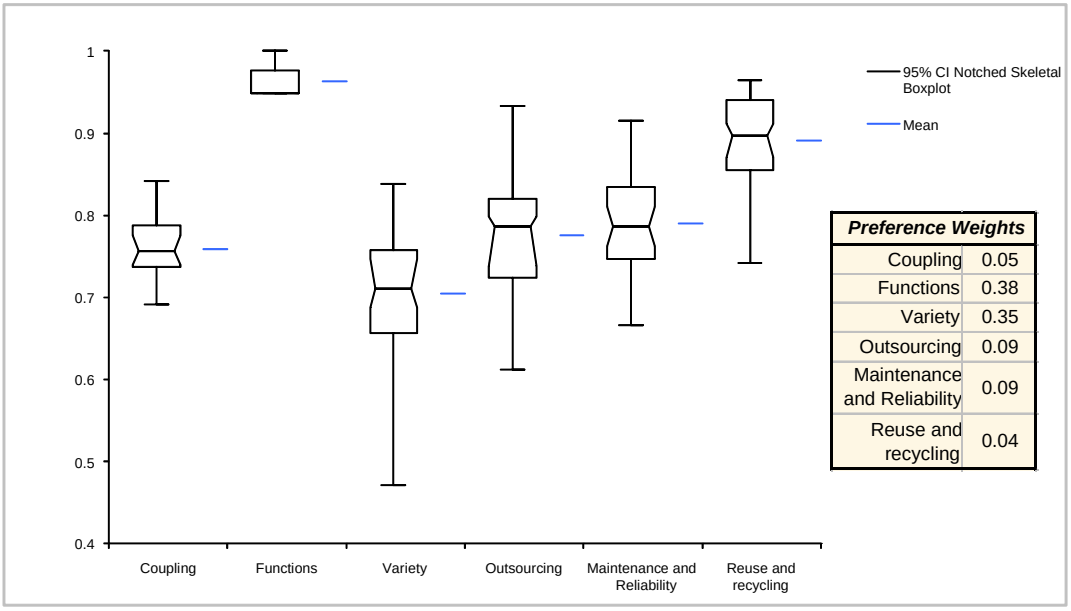

Figure 7.6. Box plots showing a preference placed on functional binding and variance objectives

What is clear from these box plot visualisation tests is that the preference weighted scheme is clearly working and in each case a more focussed set of Pareto-optimal solutions are being generated that reflect the user based preferences given.

7.7. Conclusion

In order to be confident that the modular architecture solutions generated by the algorithm are as optimal as possible the performance of the MOGGA has been tested in this chapter.

As the diversity measure is considered important for ‘many’ objective problems a number of different measures have been tested and the results confirm that there is no significant difference between them. The K-nearest neighbour approach of SPEA2 will be adopted as it fits best with the code of the developed MOGGA.

The influence of the mutation operator has also been tested and the results show that there is no difference in performance of the MOGGA when using the mutation procedure highlighted in this chapter. It is thus concluded that the mutation operator will not be used in the MOGGA, as it will add extra time to the optimisation run, with no benefit gained.

The MOGGA has also been tested against a traditional aggregated (weighted) objective algorithm to verify that the MOGGA is able to find Pareto-optimal solutions that are as good (or similar) to the single solutions generated by the algorithm. Results confirm that the MOGGA is indeed capable of finding solutions that are very close to the reference solutions produced by the aggregated algorithm and most importantly the tests confirm that the reference solutions are not dominating (better in all objectives) the solutions generated by the MOGGA.

Lastly it has been seen that the preference based ranking can be successfully integrated into the MOGGA, which can be used to produce a more focused set of Pareto-optimal solutions should the user wish. This may be particularly useful to

narrow the search space when performing modularisation of a large complex product.

CHAPTER 8

8. Case Study

8.1. Introduction

The chapter will present a case study of an automotive climate control system. The various steps of the CAMO framework will be applied to the climate control system to demonstrate the application of the approach and to show that alternative modular architectures can be found by looking at modularity from a more holistic lifecycle viewpoint.

8.2. Overview of Car Climate Control System

The car climate control system is a fairly complex system that is comprised of various technologies that must be split across numerous geometric locations within the car. This makes the climate control system an ideal case example to assess the potential of the developed modularity framework. The case study is in fact based upon the works of Pimmler and Eppinger (1994) who look at the clustering of highly interactive components to improve product development.

The data used for evaluation of the six modularisation objectives comes from a number of sources. The information on the functional and physical interactions between components has been taken from Pimmler and Eppinger's case study and used as a basis for the coupling objective. The product variety information has been taken, in part, from a case study performed by Hata et al (2001). Outsourcing information has been gathered by online research. Recycling and reuse objective scores have been based upon vehicle teardown information provided by Jaguar. Lastly the functional decomposition and the maintenance and reliability information has been generated by the author using best judgement.

8.3. Framework Steps Applied to Automotive Climate Control System

Step 1:Decomposition Analysis

The physical decomposition of the automotive climate control system follows Pimmler and Eppinger's work, who decompose the system into 16 main functional components. This breakdown considers the main functional components only. These components can be thought of as the chosen technical solutions that have arisen from the conceptual design phase. The different components are listed in figure 8.1, the front end of the systems can be seen in figure 8.2 and an example of the overall system within a car can be seen figure 8.3.

1	Air Controls	9	Compressor
2	Refrigeration Controls	10	Accumulator
3	Sensors	11	Evaporator Core
4	Heater Hoses	12	Heater Core
5	Command Distribution	13	Blower Motor
6	Radiator	14	Blower Controller
7	Engine fan	15	Evaporator Case
8	Condenser	16	Actuators

Figure 8.1. Main components of the automotive climate control system

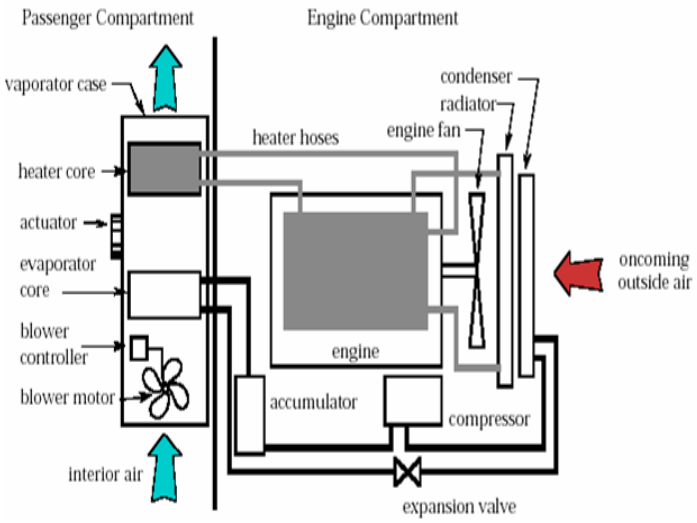

Figure 8.2. Diagram of automotive climate control system (Pimmler and Eppinger, 1994)

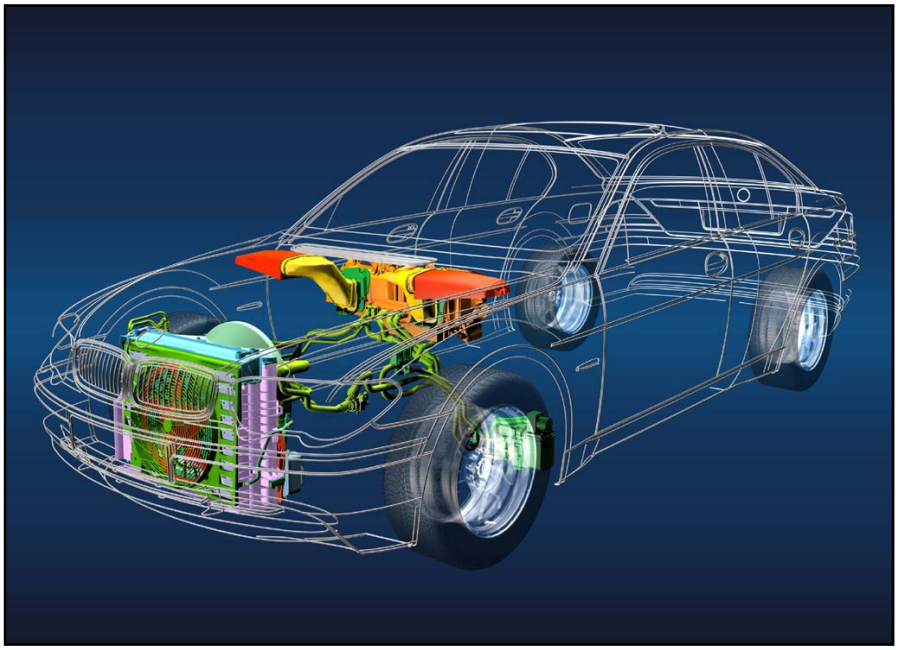

Figure 8.3. Diagram of car climate control system (taken from Behr, 2010)

The overall function of the system is to control the interior temperature of the car and consists of a combined air-conditioning (AC) and heater system. The AC side consists of a compressor, accumulator, condenser and evaporator core and uses the basic principles of the refrigeration cycle: i.e. the compressor pressurises a refrigerant gas, causing the gas to heat up. The compressed gas is then passed to the condenser and cooled by heat exchange with the exterior air, and the gas condenses to a liquid. The liquid is then pumped to the evaporator core, where the pressure drops and the fluid evaporates. The evaporation of the gas absorbs heat from the surrounding air, and the surrounding air cools off. The gas is then returned to the compressor.

To provide heat, there are two main components: a heater core and heater hoses. Heater hoses simply take heated liquid from the engine and pass it to the heater core which heats up the surrounding air adding heat to the system.

As seen in figure 8.2 the evaporator core, heater core and blower motor are housed in the evaporator case which is positioned in the front end of the engine bay. Air is drawn from the car using the blower motor and passed into the evaporator case.

Actuator controlled flaps then direct air across the evaporator core and heater core, adding or removing heat from the system. The heated/ cooled air is then blown back into the car.

Temperature is controlled by the refrigeration controls, air controls and sensors. These components, via the command distribution, send control signals to the compressor, actuators and blower controller. This, in turn, will increase/ decrease refrigeration by changing the compressor speed, will change the amount of air directed across the evaporator and heater core by adjusting the actuator position and will increase/ decrease air flow via the blower motor speed.

As well as definition and identification of the physical components, in this step of the framework the main product functions are also found. The overall function of the automotive climate control system is to 'control temperature of car interior' which has been broken down into further sub-functions and mapped to physical components as seen in figure 8.4. Following the ideas presented in chapter 5, the idea is to identify functions at a higher level than that of the physical components.

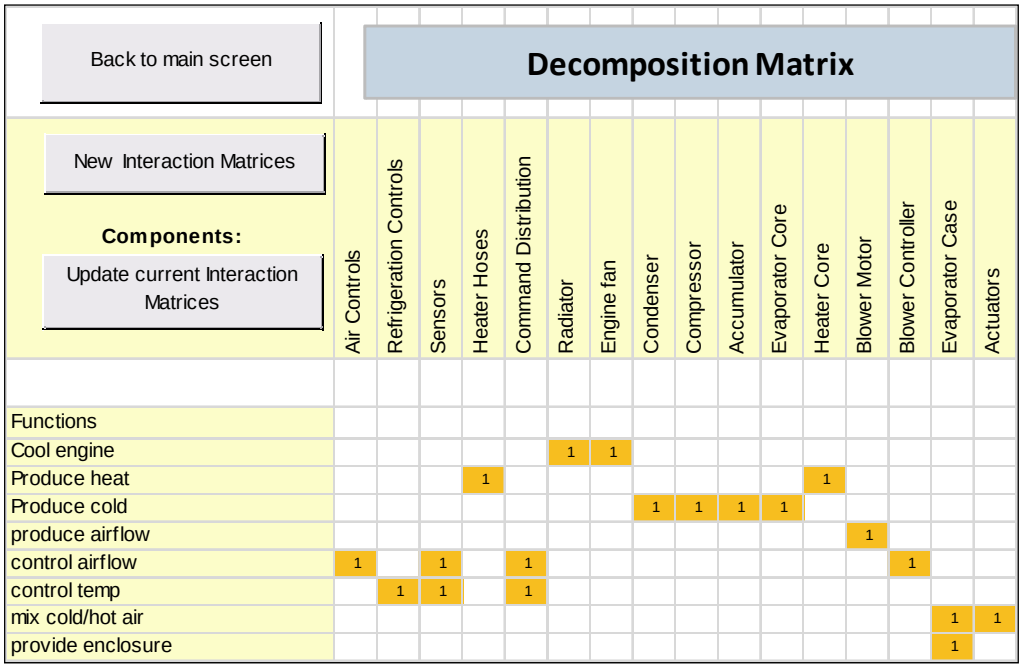

Figure 8.4. Main product decomposition matrix for the automotive climate control system showing functions mapped to components

The resulting functional interaction matrix can be seen in figure 8.5. If two components contribute to the same function a value of 1 is deduced by the software and input into the corresponding position in the matrix. For example the heater hose and the heater core are both highly affected by the function 'provide heat' so receive a '1' interaction score.

Function Interaction Matrix

Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
Air Controls	1		1		1									1		
Refrigeration Controls		1	1		1											
Sensors	1	1	2		2									1		
Heater Hoses				1								1				
Command Distribution	1	1	2		2									1		
Radiator						1	1									
Engine fan						1	1									
Condenser								1	1	1	1					
Compressor								1	1	1	1					
Accumulator								1	1	1	1					
Evaporator Core								1	1	1	1					
Heater Core				1								1				
Blow er Motor													1			
Blow er Controller	1		1		1									1		
Evaporator Case															2	1
Actuators															1	1
Functions																
Cool engine						1	1									
Produce heat				1								1				
Produce cold								1	1	1	1					
produce airflow													1			
control airflow	1		1		1									1		
control temp		1	1		1											
mix cold/hot air															1	1
provide enclosure															1	

Figure 8.5. Functional interaction matrix for the automotive climate control system

Step 2: Interaction Analysis

The second step of the case study is to populate the interaction matrix by evaluating all the relevant modularity objectives.

Loose Coupling

Because it is assumed that the design will be in the conceptual stages the basic coupling interactions are used. The coupling relationships between the components of the automotive climate control system can be seen in figure 8.6 and are based upon the physical and functional interactions of Pimmler and Eppinger's work.

Coupling Interaction Matrix

Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
Air Controls	1		0.4		0.4				0.1					0.1		0.1
Refrigeration Controls		1			0.3				0.1	0.3						
Sensors	0.4		1		0.3											
Heater Hoses				1								0.3				
Command Distribution	0.4	0.3	0.3		1		0.3		0.3				0.3	0.3		0.3
Radiator						1	0.7	0.7								
Engine fan					0.3	0.7	1	0.7								
Condenser						0.7	0.7	1	0.3		0.5					
Compressor	0.1	0.1			0.3			0.3	1	0.4	0.3					
Accumulator		0.3							0.4	1	0.4					
Evaporator Core								0.5	0.3	0.4	1		0.2		0.5	
Heater Core				0.3								1			0.5	
Blow er Motor					0.3						0.2		1	0.7	0.4	
Blow er Controller	0.1				0.3								0.7	1	0.2	
Evaporator Case											0.5	0.5	0.4	0.2	1	0.2
Actuators	0.1				0.3										0.2	1

Figure 8.6. *Coupling interaction matrix for the automotive climate control system*

For example, according to Pimmler and Eppinger, there is a strong physical relationship between the blower motor and the evaporator case as well as a

functional interaction – material flow (air flow) between the components, and so based on the interaction form the interaction score of 0.7 is entered into the corresponding position in the matrix.

There are a number of geometric constraints that have been defined as can be seen in figure 8.7. These hard constraints are defined because the various components of the automotive climate control system have to be split across different geometric locations. In this interaction matrix if a '1' exists between components then a constraint exists between them and they will not be grouped into the same module.

Constraints Interaction Matrix

Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
Air Controls						1	1	1	1							
Refrigeration Controls						1	1	1	1							
Sensors						1	1	1	1							
Heater Hoses																
Command Distribution						1	1	1	1							
Radiator	1	1	1		1				1		1	1	1	1	1	1
Engine fan	1	1	1		1				1		1	1	1	1	1	1
Condenser	1	1	1		1				1		1	1	1	1	1	1
Compressor	1	1	1		1	1	1	1			1	1	1	1	1	1
Accumulator																
Evaporator Core						1	1	1	1							
Heater Core						1	1	1	1							
Blow er Motor						1	1	1	1							
Blow er Controller						1	1	1	1							
Evaporator Case						1	1	1	1							
Actuators						1	1	1	1							

Figure 8.7. Constraint interaction matrix for the automotive climate control system

Variance

A typical producer of automotive climate control systems will have contracts with various car manufacturers and so will have to produce many different variants for different car models - inevitably there will be a large number of different component variants needed. However, there will be certain components that can be standardised and used across multiple car models. The common components should be grouped and integrated into common product platforms and variant components kept as more flexible variant modules to make the design, manufacture and assembly process more efficient.

Firstly the variant modes must be analysed. The various variance modes and their mapping to the components can be seen in figure 8.8. Some of the variety modes have been taken from the case study done by Hata et al (2001) – a case study on part of an automotive climate control system front end unit produced by Denso. According to Hata a number of components can be defined as variants. These variants are needed to enable the product to 'fit' numerous car models, and are the heater hoses and connectors (command distribution) and the evaporator case. According to Hata et al (2001) the heater core and evaporator core are most likely to become common components probably due to the relative high cost of manufacture. Hata's work also discusses the need for manual and automatic versions of the product. That is, there will be two different product platforms for the climate control system. To provide the different functionality required for each product platform there will be different functional components (technical solutions) needed. However, there will be certain functions (and associated physical components) that will be common to both of the product platforms and should be kept separate from the variants so they can be shared. To identify these common functions it is of course necessary to then define the variant functions that will provide the different models' functional requirements. These variant functions have been identified as: 'control airflow', 'control temp' and 'mix hot/cold air', which can be seen in the interaction matrix in figure 8.8.

In the real world application of the CAMO framework a modularisation of both the manual and automated version of the product will be needed. In this case study it is assumed that the product is the automated version.

Variance Interaction Matrix

	Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
	Air Controls	1	1	1	0	0.5	0	0	0	0	0	0	0	0	1	0	1
	Refrigeration Controls	1	1	1	0	0.5	0	0	0	0	0	0	0	0	1	0	1
	Sensors	1	1	1	0	0.5	0	0	0	0	0	0	0	0	1	0	1
	Heater Hoses	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	Command Distribution	0.5	0.5	0.5	0	1	0	0	0	0	0	0	0	0	0.5	0	0.5
	Radiator	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Engine fan	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Condenser	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Compressor	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Accumulator	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Evaporator Core	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Heater Core	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Blow er Motor	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
	Blow er Controller	1	1	1	0	0.5	0	0	0	0	0	0	0	0	1	0.5	1
	Evaporator Case	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5	1	0.5
	Actuators	1	1	1	0	0.5	0	0	0	0	0	0	0	0	1	0.5	1
Type	**Variance mode**	V	V	V	V	V	C	C	C	C	C	C	C	C	V	V	V
Component	Hose length				1												
Component	Command distribution Length					1											
Component	Evaporator case size															1	
Function	control airflow (manual/ auto models)	1		1		1									1		
Function	control temp (manual/ auto models)		1	1		1											
Function	mix cold/hot air (manual/ auto models)															1	1

Figure 8.8. Variance interaction matrix for the automotive climate control system

Outsourcing

For the outsourcing objective the evaluations have been deduced based upon research into current automotive climate control system design and production. It has been found that current practice in the automobile industry is to outsource the complete automotive climate control system to a tier 1 supplier, who will design and manufacture it according to the car manufacturer's specifications. Tier one suppliers that provide automotive climate control system systems include Denso, Visetion and Behr. These suppliers then outsource the design and production of certain automotive climate control system components to their own partner suppliers. Although the information on which components are outsourced to which supplier is not available, some sensible assumptions have been made in order to score each component for the outsourcing objective. For example the design and production of the electronic control system, (the air and refrigeration controls, the command distribution and the sensors and blower controller) is highly likely to be outsourced to a company that specialises in control systems. Hence during the design of a new automotive climate control system all control based components should be in the same module, which will then be designed and produced by the control system company. This can be verified by looking at Behr's automotive climate control product development partnerships. Behr use Behr-Hella Thermocontrol *(*BHTC*)* to provide their control systems. The mapped outsourcing needs can be seen in figure 8.9.

Outsourcing Interaction Matrix

Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
Air Controls	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0
Refrigeration Controls	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0
Sensors	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0
Heater Hoses	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Command Distribution	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0
Radiator	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Engine fan	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Condenser	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Compressor	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Accumulator	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Evaporator Core	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Heater Core	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Blow er Motor	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Blow er Controller	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0
Evaporator Case	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Actuators	0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	1
Supplier																
supplier 1	1	1	1		1									1		
Make in-house				1		1	1	1		1	1	1	1		1	1
supplier 2									1							

Figure 8.9. Outsourcing interaction matrix for the automotive climate control system

Maintenance and Reliability

For this objective the author's best judgements have been made. The potential maintenance and failure modes were listed (can be seen at the bottom of the main interaction matrix in figure 8.10). After this information was entered, a full evaluation (with the interaction evaluation form) was then carried out based upon the response of each component to these modes.

Maintenance and reliability Interaction Matrix																
Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
Air Controls	1	0.5	0.5	0.5	1	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5
Refrigeration Controls	0.5	1	1	0	1	0	0	0	0.5	0.5	0	0	0	0	0	0
Sensors	0.5	1	1	0	1	0	0	0	0	0	0	0	0	0.5	0	0
Heater Hoses	0.5	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
Command Distribution	1	1	1	0	1	0	0	0	0	0	0	0	0	0.5	0	0.5
Radiator	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Engine fan	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Condenser	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Compressor	0	0.5	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Accumulator	0	0.5	0	0	0	0	0	1	1	1	1	0	0	0	0	0
Evaporator Core	0	0	0	0	0	0	0	1	1	1	1	0.5	0	0	0.5	0
Heater Core	0	0	0	1	0	0	0	0	0	0	0.5	1	0	0	0.5	0
Blow er Motor	0.5	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Blow er Controller	0.5	0	0.5	0	0.5	0	0	0	0	0	0	0	1	1	1	1
Evaporator Case	0.5	0	0	0	0	0	0	0	0	0	0.5	0.5	1	1	1	1
Actuators	0.5	0	0	0	0.5	0	0	0	0	0	0	0	1	1	1	1
Failure Mode																
Not cooling engine properply						1	1									
not enough supply of heat	1	1		1	1							1				
not enough supply of cold		1	1		1			1	1	1	1					
not enough airflow	1		1		1								1	1	1	1

Figure 8.10. Maintenance and reliability interaction matrix for the automotive climate system

Recycling and Reuse

End of life vehicles are recycled using automated recycling processes. Hence, at the end of the automotive climate control system's life, the unit is likely to be put into a shredder and put through a number of separation processes to reclaim useful materials. However these processes are not perfect and reclaimed materials are often left contaminated, reducing their usefulness as reusable engineering materials. For example copper will contaminate the recovered metals, reducing the ability to

use the reclaimed metal for engineering applications. Modularity can help overcome these problems by grouping potential problem materials into modules that can be removed before shredding and separation. Hence the recycling objective for the automotive climate control system is to separate into modules, components that contain problem materials from components that do not.

To analyse which components are likely to contain problem materials and those that do not, the likely material types are needed for each of the automotive climate control system components. A car disassembly teardown provided by an automobile manufacturer has been used to gather the material type information for the main functional components of the car climate control system.

High value components of an automobile are often removed for spares by car dismantlers before the car is shredded. According to the Jaguar teardown, components that have high reuse potential are the blower motor and its controller and the compressor. Figure 8.11 shows how the recycling and reuse objective is mapped to components. The score system for the recycling and reuse objective is outlined in chapter 6.

Recycling and reuse Interaction Matrix

Components:	Air Controls	Refrigeration Controls	Sensors	Heater Hoses	Command Distribution	Radiator	Engine fan	Condenser	Compressor	Accumulator	Evaporator Core	Heater Core	Blower Motor	Blower Controller	Evaporator Case	Actuators
Air Controls	1	1	1	1	1	0	0	0	0	0	0	0	0	0.5	0	0.5
Refrigeration Controls	1	1	1	1	1	0	0	0	0	0	0	0	0	0.5	0	0.5
Sensors	1	1	1	1	1	0	0	0	0	0	0	0	0	0.5	0	0.5
Heater Hoses	1	1	1	1	1	0	0	0	0	0	0	0	0	0.5	0	0
Command Distribution	1	1	1	1	1	0	0	0	0	0	0	0	0	0.5	0	0.5
Radiator	0	0	0	0	0	1	0.5	1	0	1	1	1	0	0	1	0.5
Engine fan	0	0	0	0	0	0.5	1	0.5	0	0.5	0.5	0.5	0	0	0.5	0.5
Condenser	0	0	0	0	0	1	0.5	1	0	0.5	1	1	0	0	1	0.5
Compressor	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
Accumulator	0	0	0	0	0	1	0.5	0.5	0	1	0.5	0.5	0	0	0.5	0.5
Evaporator Core	0	0	0	0	0	1	0.5	1	0	0.5	1	1	0	0	1	0.5
Heater Core	0	0	0	0	0	1	0.5	1	0	0.5	1	1	0	0	1	0.5
Blow er Motor	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Blow er Controller	0.5	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0	1	1	0	0.5
Evaporator Case	0	0	0	0	0	1	0.5	1	0	0.5	1	1	0	0	1	0.5
Actuators	0.5	0.5	0.5	0	0.5	0.5	0.5	0.5	0	0.5	0.5	0.5	0	0.5	0.5	1
EOL Option																
Reuse									1				1	1		
recycle (shedder)						1	1	1		1	1	1			1	1
dispose (remove)	1	1	1	1	1				1					1		1

Figure 8.11. Recycling and reuse interaction matrix for the automotive climate system

Step 3: Formation of Modular Architectures

The goal of the GA optimisation is to create a set of non-dominated Pareto-optimal solutions according to the information entered into the six interaction matrices, so that the DM can explore trade-offs. For the automotive climate control system a set of 50 non-dominated, unique solutions was generated in approximately 2 minutes on a 2.5 GHz dual core processor. The configuration settings of the GA can be seen in figure 8.12.

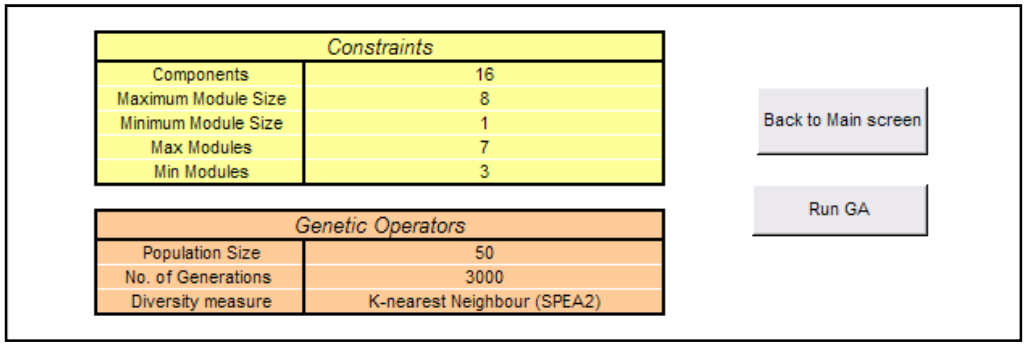

Figure 8.12. MOGGA settings for the automotive climate control system example

Step 4: Analysis of Solution Set

In this section the results of the GA optimisation will be explored to consider how a trade-off analysis can be performed using the solution set. From the non-dominated set, trade-off analysis has been carried out by searching the set for the closest matching solutions according to the DM preferences. The advantage of the method is that trade-off analysis can be carried out in real time by adjusting the considered importance of the various modularity objectives using the 'objective hierarchy' as shown in figure 8.13.

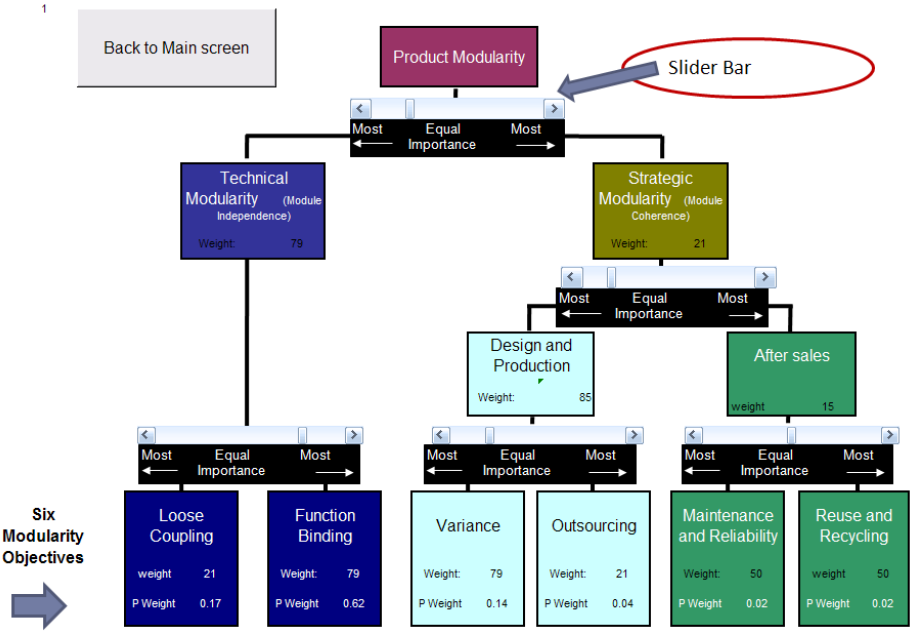

Figure 8.13 Modularity objective hierarchy with preference placed on technical modularity

Scenario analysis 1: Focus on module independence

Presume that the DM considers 'technical modularity; is of high importance and hence moves the 'slider' bar as seen in figure 8.13. The corresponding 'best' solutions can be visualised using radar plots as seen in figure 8.14. Presume that upon analysis, these solutions are not suitable. They show that to achieve such high module independence (loose coupling and functional binding) the compromises that have to be made for the other strategically based modularity objectives are too high.

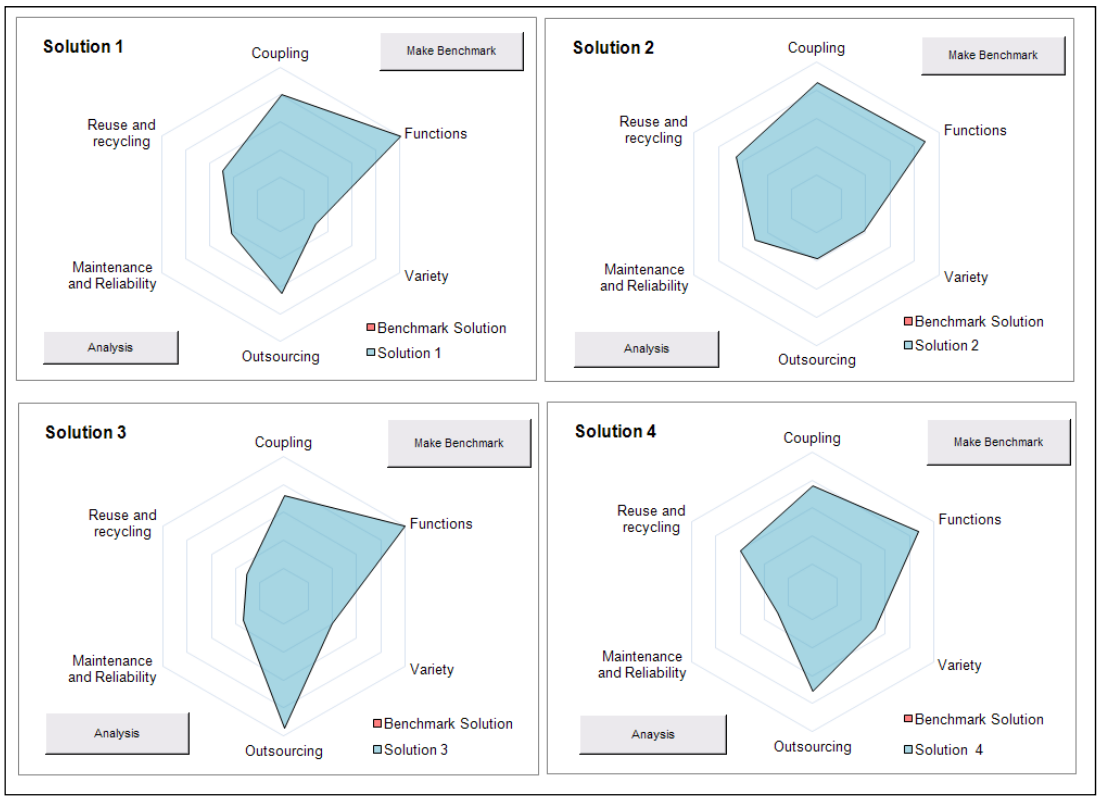

Figure 8.14. Optimal solutions for the automotive climate control system with the focus on technical modularity

Scenario analysis 2: Focus on module coherence

Now presume that the DM considers 'strategic modularity' to have more importance and moves the 'slider' bar accordingly. The corresponding 'preferred' solutions are shown in figure 8.15. These solutions now show much improvement for the strategic based objectives. The DM is much happier with these solutions and thus decides to explore each solution further by clicking on the analysis button to examine the module structure - an example can be seen in figure 8.16. The DM then decides that solution 1 is a promising solution and sets this as the benchmark solution. However, before deciding upon this solution the DM decides to carry on exploring the set by examining more scenarios.

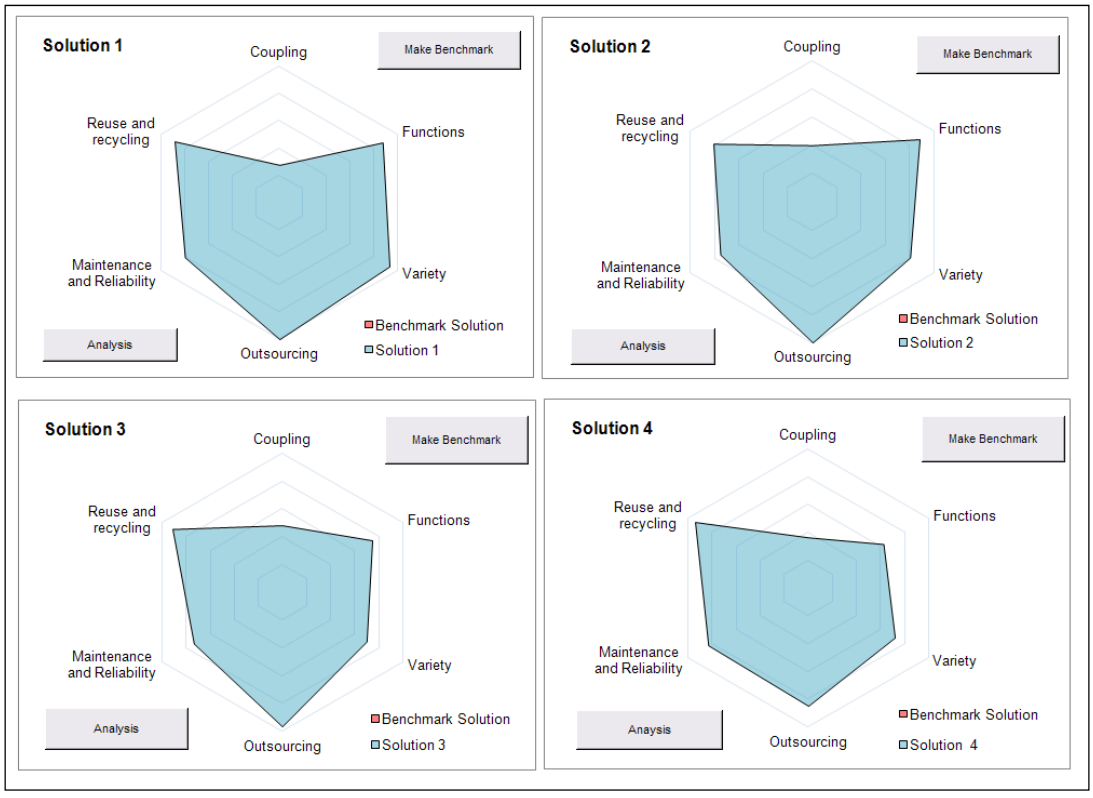

Figure 8.15. Optimal solutions for the automotive climate control system with the focus on strategic modularity

Anlayis Matrix

Module Indepedence: Coupling, Function

Module Coherence: Variance, Outsourcing, Maintenance, Recycling

Analyse

Back to Soultion Exloration

Set solution to benchmark

module Number	componnet no		Evaporator Case	Actuators	Air Controls	Refrigeration Controls	Sensors	Command Distribution	Blower Controller	Blower Motor	Heater Hoses	Heater Core	Evaporator Core	Compressor	Radiator	Engine fan	Condenser	Accumulator
1	1	Evaporator Case	1	0.5					15	4		5	5					
1	2	Actuators	0.5	1	1			2.5										
2	3	Air Controls		1	1	1	1	0.5	1					1				
2	4	Refrigeration Controls			1	1	1	0.5	1					1				2.5
2	5	Sensors			1	1	1	0.5	1									
2	6	Command Distribution		2.5	0.5	0.5	0.5	1	0.5	2.5				2.5		2.5		
2	7	Blow er Controller	15		1	1	1	0.5	1	6.5								
3	8	Blow er Motor	4					2.5	6.5	1			15					
4	9	Heater Hoses									1	0						
4	10	Heater Core	5								0	1						
5	11	Evaporator Core	5							15			1	2.5			5	4
6	12	Compressor			1	1		2.5					2.5	1			2.5	4
7	13	Radiator													1	1	1	1
7	14	Engine fan						2.5							1	1	1	1
7	15	Condenser											5	2.5	1	1	1	1
7	16	Accumulator				2.5							4	4	1	1	1	1
		Functions																
		Cool engine													1	1		
		Produce heat									1	1						
		Produce cold											1	1			1	1
		produce airflow								1								
		control airflow			1		1	1	1									
		control temp				1	1	1										
		mix cold/hot air	1	1														
		provide enclosure	1															
		Variance mode	V	V	V	V	V	V	V	C	V	C	C	C	C	C	C	C
		Hose length									1							
		Command distribution Length						1										
		Evaporator case size	1															
		control airflow (manual/ auto models)			1		1	1	1									
		control temp (manual/ auto models)				1	1	1										
		mix cold/hot air (manual/ auto models)	1	1														

Figure 8.16. Analysis matrix for the automotive climate control system - showing loose coupling between modules (independence) and variance interactions within modules (coherence).

Scenario analysis 3: Focus on design and manufacturing stage

In this scenario it is assumed that the DM has considered technical modularity and strategic modularity of equal importance and then moves down the hierarchy and decides that 'design and production' is more important than 'after sales'. The best solutions according to these preferences can be seen in figure 8.17. The benchmark solution can now be seen and is compared to these four 'best' solutions. However, none of these solutions is considered better than the benchmark.

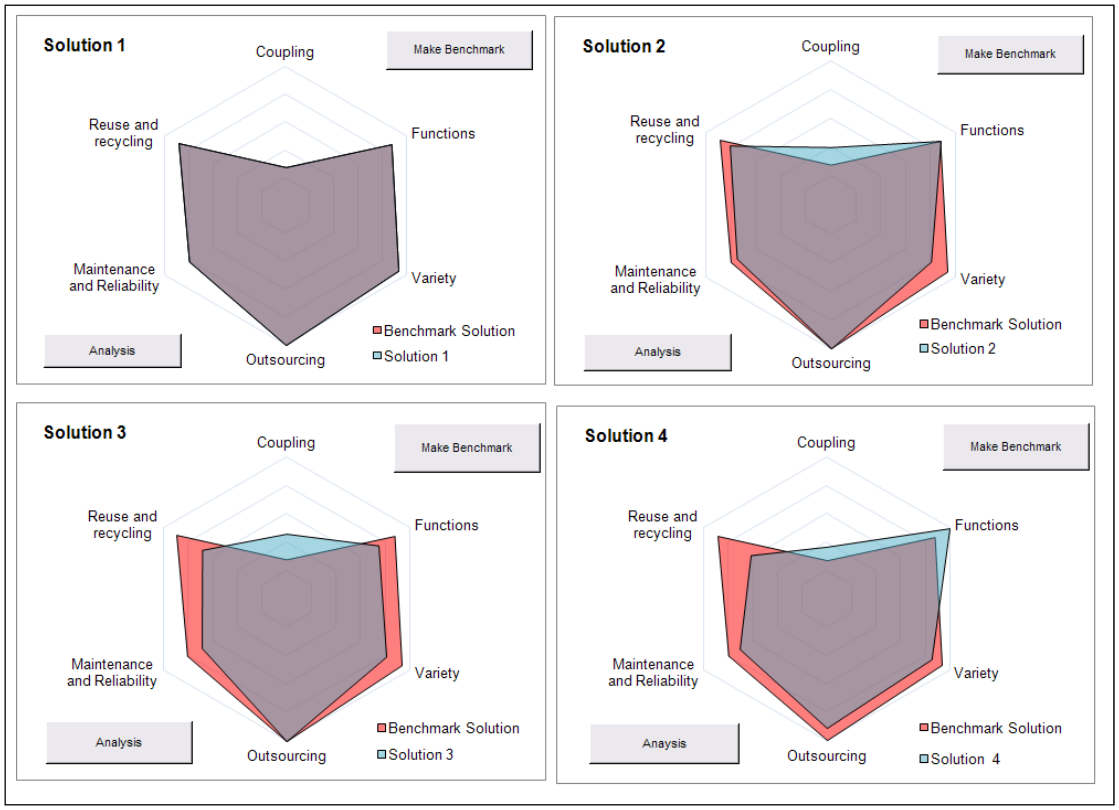

Figure 8.17. Optimal solutions for the automotive climate control system with the focus on design and manufacturing stage

Scenario analysis 4: Focus on after sales phase

In this scenario, the DM wishes to explore the best solutions when 'after sales' objectives are considered more important than the 'design and production'. The new best solutions can be seen in figure 8.18. However the benchmark is still preferred as the small improvements in other objectives that the new solutions offer mean that too great a compromise must be made for the other objectives.

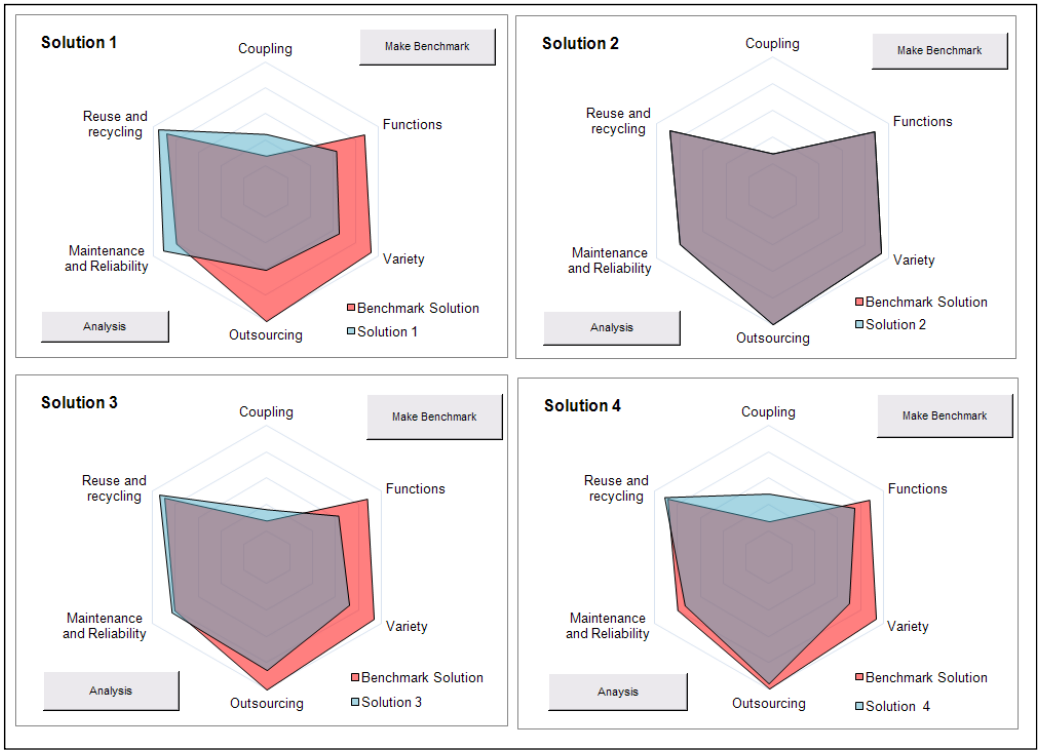

Figure 8.18. Optimal solutions for the automotive climate control system with the focus on after sales phase

Chosen Solution for the Automotive Climate Control System

The chosen solution is seen in figures 8.19 and 8.20. Although this is a hypothetical decision, it is a solution that offers good performance in most of the modularisation objectives. The relatively poor performance of the 'loose coupling' can be 'tackled' by the careful design of the interfaces between modules.

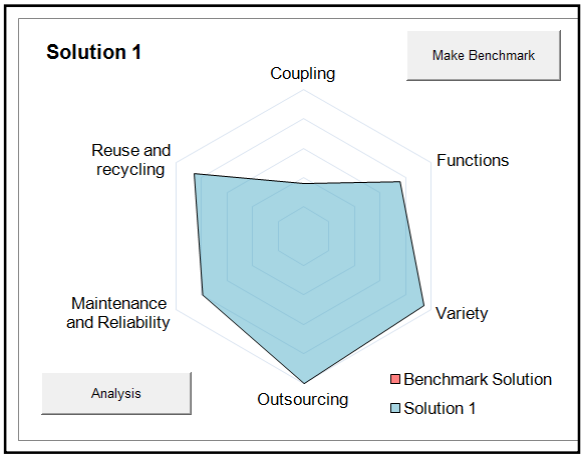

Figure 8.19. Chosen solution for the automotive climate control system

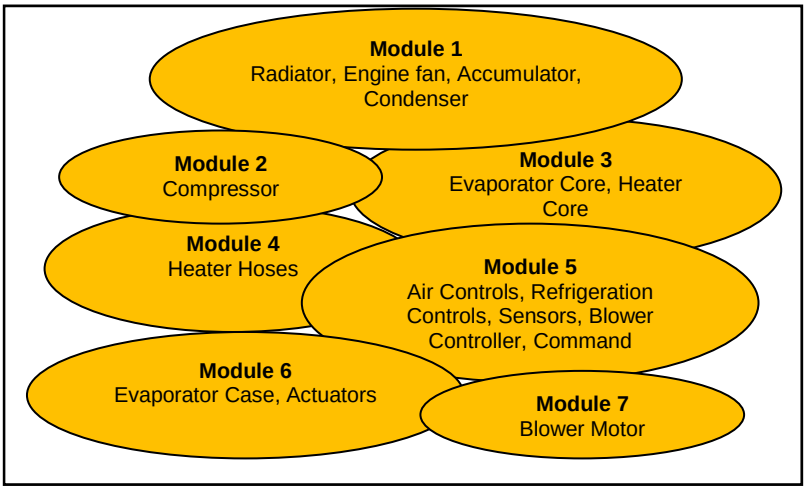

Figure 8.20. Chosen modular architecture for the automotive climate control system

8.4. Comparison to Other Methods

In this part of the case study the chosen modular architecture will be compared to existing climate control systems and case study results obtained by previous researchers.

Comparison of results to Existing Automotive climate control systems

The chosen modular product architecture has been compared with the existing modular structure in currently manufactured climate control systems. The information regarding this current modular structure comes from Nepal (2005), who performed the same case study with a number of tier one automotive suppliers. Nepal discusses that the current climate control system was not systematically modularised in the past, and hence very few modules existed. This modular structure can be seen in figure 8.21.

Figure 8.21. Existing modules for automotive climate control system

When comparing this existing product structure to the new one proposed in the previous section (figure 8.20) it can be seen that there are fewer modules in the existing product. This structure may well be optimal for assembly time, and reduces the interface complexity needed between modules. However, a number of issues may arise from this configuration. When this existing configuration is evaluated using the CAMO framework, it can clearly be seen (in figure 8.22) that, although

the module independence (coupling and function) is high, the modularisation objective achievements for the various strategic considerations are considerably lower than the chosen modular configuration.

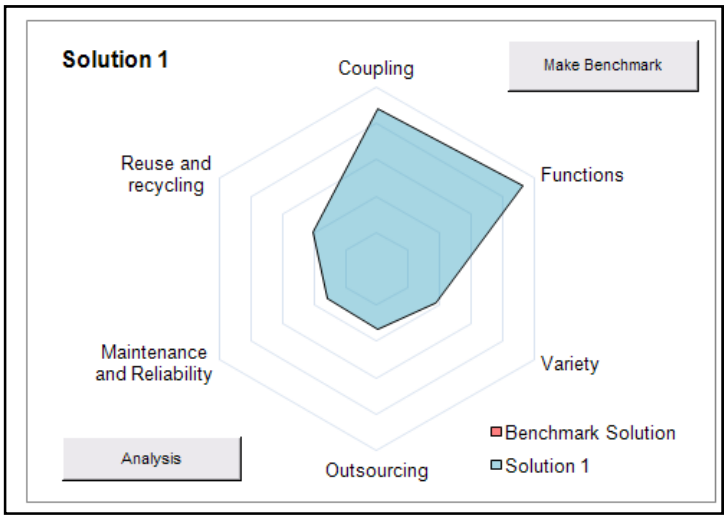

Figure 8.22. Evaluation result of existing automotive climate control system with CAMO

For example, the large front end module (module 1) will have poor performance in terms of maintainability because not all the components have common maintenance requirements. The cost of implementing and managing product variety is also going to be higher because, depending upon the vehicle type and size, there may be different requirements for the type of controls, cases and connectors i.e. to make the whole module a variant, the costs will be considerably higher than splitting the modules further into common and variant modules. These costs are of course difficult to quantify as, like other frameworks, the framework does not contain any detailed means of module cost analysis. Yet, it can be implied the CAMO framework does provide a clear insight into how costs may be affected by looking closely at how the chosen configurations respond to the different modularisation objectives.

Comparison to Pimmler and Eppinger Framework and Results

The results obtained from this study have been compared with those of Pimmler and Eppinger (1994). In their approach four modules were suggested for the climate control system (see figure 8.23). In their study module formation is based upon the formation of design/ development teams and they only used one modularisation objective – the functional and physical interactions between components.

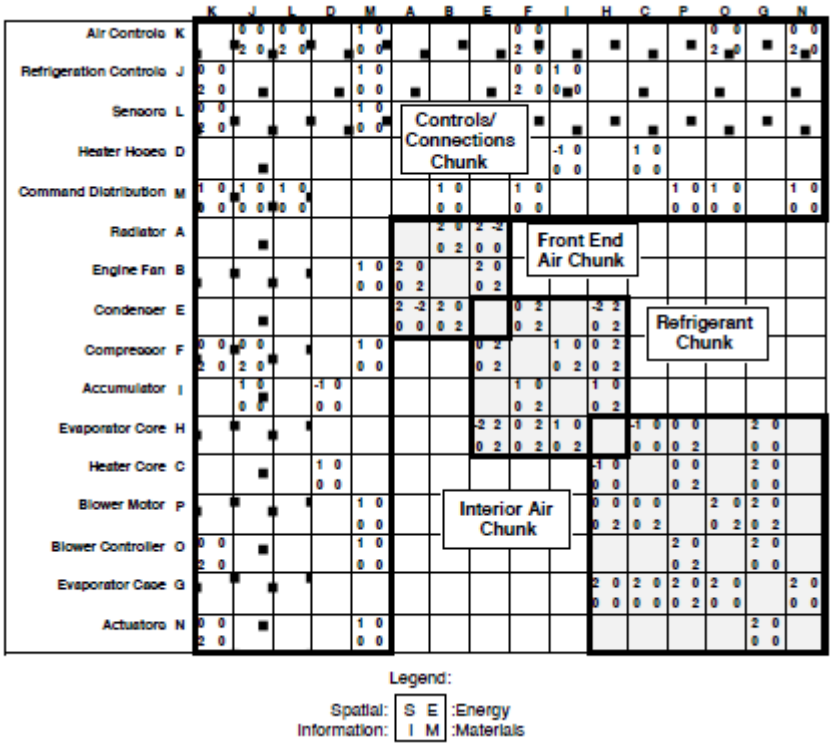

Figure 8.23. Recommended modules for automotive climate control system (Pimmler and Eppinger, 1994)

However, for an optimal modular product architecture, other modularisation objectives must also be considered. As mentioned previously, having too few modules may affect the strategic coherence of the modules. This will in turn adversely affect the performance of the product architecture in terms of other factors such as the variance needs of the product, outsourcability of modules, maintenance needs and the reusability and recyclability of modules.

In addition, Pimmler and Eppinger (1994) state that the matrix should be clustered to form modules, but do not describe such an algorithm. Even when the matrix is clustered the choice of module boundaries is still ambiguous.

Comparison to Stone's Framework and Results

The results from the CAMO framework are also compared with the "module heuristics" developed by Stone (1997). In Stone's method there are three heuristics used to identify the modules based on functional flow patterns: "dominant flow", "branching flow", and "conversion-transmission flow". According to these heuristics, Stone suggests 10 modules for the climate control system, as can be seen in figure 8.24.

As with the Pimmler and Eppinger technique, it emphasises module formation based only upon design team formation, and hence the product architecture is based on the functional interactions.

Stone's method is a graphical method that includes flow information to identify modules. The method provides a strong basis for modularising the product from a functional perspective, yet the construction of a functional diagram showing all the flows is quite complex and tedious. Furthermore, as it relies on only the functional perspective, it does not pursue a physical decomposition of the product to identify the primary physical components - even though the product can be readily decomposed. This makes the method even more confusing for the designer unfamiliar with this way of product decomposition. In contrast, the decomposition analysis in the CAMO framework is relatively straightforward and more consistent with what designers will be familiar with.

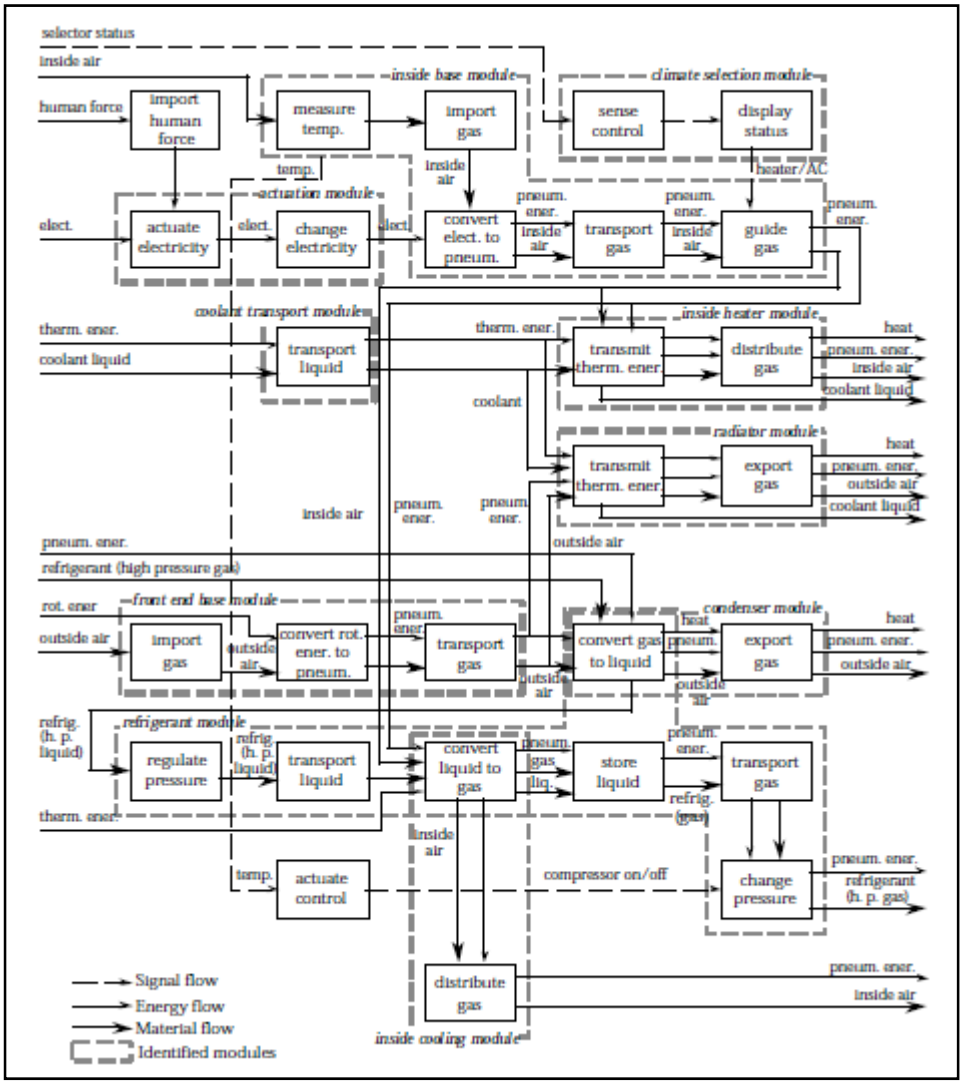

Figure 8.24. Recommended modules for automotive climate control system (Stone, 1997)

Comparison to Nepal's Framework and Results

Nepal (2005) uses a matrix based approach that provides fuzzy logic based interaction evaluation to integrate four sets of modularisation sub-objectives into four modularity performance goals that are then handled by a goal programming based optimisation model. The four performance goals used for the climate control system are: cost, maintenance and reliability maintenance, quality and manufacturability.

This model allows the DM to define preferences, such as higher quality, lower cost product etc. These preferences can be changed by varying the aspiration levels in the goal-programming model. In this way a rudimentary 'what-if' scenario analysis can be carried out by changing the goals aspiration levels and rerunning the optimisation model.

In Nepal's case he undertook a number of scenario analyses and arrived at the modular structure seen in figure 8.25. Performing scenario analysis is what has been advocated in the CAMO framework and in this case the chosen module structure is not too dissimilar to the proposed chosen structure of the CAMO framework. However, Nepal's framework does not address the issue of recycling and reuse, outsourcing or product variance. If the designer considers these objectives of high importance then the modular structure may look somewhat different.

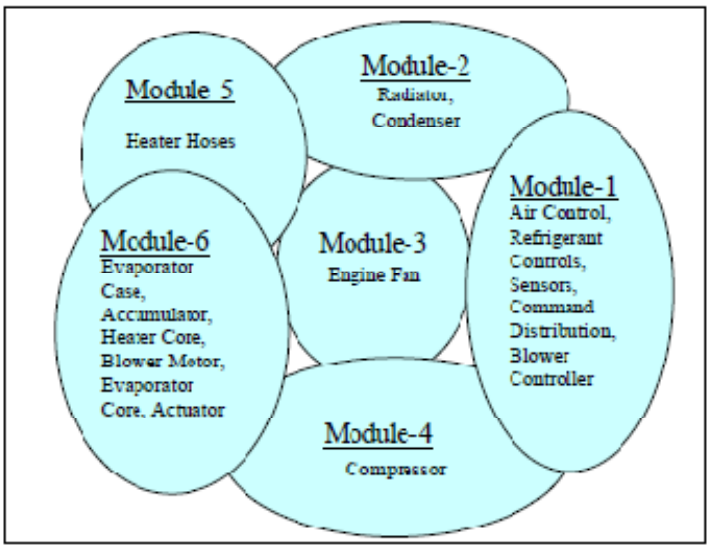

Figure 8.25. Recommended modules for automotive climate control system (Nepal, 2005)

Although the approach of the CAMO framework may appear to share some similarities to the approach advocated by Nepal, there are a number of key differences.

Firstly, in Nepal's approach only interactions within modules are considered and the interactions between modules are not considered during module formation. That is the framework is only concerned with module coherence and module independence is ignored. Module independence is an important aspect of modularity. In particular, module interfaces should be a simple as possible in order to pursue the plug-in plug-out characteristics of modularity, to improve product maintainability, recyclability, outsourcabilty and variance management. Thus, one has to question the so called optimality of modules formed using Nepal's framework. Not considering module independence also has ramifications for the post modularisation stage, for the analysis and design of interfaces between modules.

In the CAMO framework the various modularity drivers have been reconciled into a clear hierarchy that considers both module independence and module coherence, providing the DM with a method to analysis and choose modular solutions that are a suitable balance between these two characteristics.

Nepal's approach also uses a goal attainment method to solve the multi-objective problem and uses an 'off-the-shelf' solver to perform the optimisation. These solvers are poorly equipped to deal with the complexities of grouping problems. There are also some problems with using a goal based aggregated objective approach to multi-criteria optimisation. The setting up of appropriate goals can be time consuming and tedious. First one must normalise each objective by running the algorithm separately for each objective. Then to find the most appropriate goal settings the algorithm will often need to be rerun many times. Furthermore, there is actually no guarantee that true Pareto-optimal solutions can actually be found if the objective space is non-linear or convex.

With the CAMO framework however the DM is not presented with the problems of setting up preferences. Furthermore, the algorithm generates a whole set of Pareto-optimal solutions in one run, so the DM is then free to explore alternative solutions in real time without having to rerun the algorithm multiple times.

8.5. Conclusion

The case study example of the automotive climate control system has demonstrated the five steps that have been advocated in the CAMO framework and the potential of this true multi-objective approach has been highlighted. It has been seen that trade-off analysis can be carried out by searching the set of non-dominated solutions and comparing the results of different preference settings.

Furthermore, the results obtained from the framework have been compared to previous methods. It has been seen that most of the previous works have only pursued a single objective during modularisation and do not provide suitable grouping algorithms for module formation. The framework on the other hand, is able to generate a whole set of alternative solutions, as well as providing a suitable means to compare these alternatives. Ultimately the chosen solution will involve compromises between objectives and it will be down to the designer to choose the most suitable product architecture from the alternatives presented.

CHAPTER 9

9. Conclusions and Further Work

9.1. Summary of Thesis

Modular product architecture is often seen as a key strategy to help address a number of challenges, such as reduced product development times, increased variety to the market, globalised product development and improved recycling and reuse. As can be expected, over the years a broad range of measures, methods and techniques have been created in attempts to guide the development of modular product architectures. A thorough literature review has in fact revealed that modularity is a diverse and large research area, with researchers approaching modularity from various viewpoints.

Modularity is however often defined as a means of controlling product complexity by decomposing the product system into smaller more manageable chunks. The vast majority of developed methods advocate the decomposition of the product into a number of smaller elements (components) which are then grouped to form larger product elements (modules). It has been found that the objectives for module grouping vary considerably, but can be defined as either technically based objectives such as the physical and functional interactions between components or strategically based objectives such as the same product maintenance needs of components.

It has also been found that the vast majority of the actual methods and frameworks that have been created are matrix based, using interaction matrices to represent the complex functional, physical and strategic based interactions that occur between components. Matrix representations are primarily used as they can be readily manipulated with optimisation algorithms to identify modules.

There are of course problems with the existing modularity methods. In summary, the process of product modularisation is highly ambiguous. Even for a relatively

simple product there are a vast number of different ways the product can be modularised, according to the different objectives of modularisation. With each different solution there will of course be compromises that have to be made between the different objectives. Ideally these compromises should be explored before arriving at a final decision. This implies that a good set of alternative solutions can in fact be found in order to make the comparisons. However, current algorithms for product modularisation are simplistic (aggregated objective) approaches. Finding a set of optimal solutions (for comparison) with these algorithms is problematic and time-consuming.

The multi-objective modularity optimisation is further complicated by the very fact there are so many modularisation goals defined, ranging from up front design objectives such as product variance, to end of life objectives such as ease of recycling.

The overall aim of this thesis has thus been to develop a computerised multi-objective optimisation framework for product modularisation. In the framework numerous modular design principles have been reconciled and integrated and a state-of-the-art multi-objective optimisation algorithm has been developed to perform the modularisation.

The computer aided modularity optimisation CAMO framework has four main steps: 1) product decomposition 2) interaction analysis 3) formation of modular architectures 4) scenario analysis.

The important aspect of the framework is that it presents a novel multi-objective approach to product modularisation, in which a whole set of alternative modular product architectures are generated in one single run of the algorithm without the need to set up preference weights for the various objectives. The solution set can then be further analysed using the analytical hierarchical process (AHP) inspired modularity objective hierarchy to choose the best compromise solution.

The multi-objective algorithm has also been tested and compared against a conventional aggregated objective algorithm and is able to produce solutions of equal quality. This is significant because an aggregated objective algorithm is conducting a more focused search (only searching for one optimal solution at a time) and therefore it is possible the solutions will be better than solutions produced by the developed multi-objective grouping genetic algorithm (MOGGA). This is not the case however as the design of the MOGGA has ensured that the search space is explored in an appropriate manner, such that optimal solutions (as good as the aggregated objective solutions) are being found.

A case study has been carried out for an automotive car climate control system. This has been used to demonstrate the various steps of the framework. It has been shown that the method can be successfully followed to perform a true multi-objective product modularisation. The results have also been compared to existing methods and it has been seen that the CAMO framework, unlike previous methods (that only generate one solution), is able to quickly generate a whole range of optimal product modularisations that can be further explored by the user it order to arrive at a suitable compromise solution.

9.2. Research Contributions

- *Critical review of product modularity literature.* A thorough literature study has been carried out to understand what defines a modular product and to identify what are the main advantages and disadvantages of modular product architectures. The main contribution of the review has however been the categorisation of different modularity viewpoints (coupling, function, variety and lifecycle) and a critical analysis of current modularity methods (these are; configuration methods; domain mapping approaches and step-wise redesign methods).

- *Technical review of GA-based optimisation algorithms.* One of the most obvious weaknesses of current modularity methods was found to be the lack of suitable optimisation algorithms. Genetic algorithms (GA) have been found to be the most suitable type of optimisation algorithm able to handle the multi-objective nature of product modularisation. Thus a detailed technical review of multi-objective GA based optimisation has also been carried out. The conclusion that can be drawn from this review is that although there have been many advances in the field, no 'off-the-shelf' algorithms are deemed suitable for the multi-objective modularity problem. This is mainly due to the lack of adaptation to grouping problems and the poor performance of the algorithms when solving 'many' (more than four) objective problems.

- *Reconciliation of main drivers of product modularisation.* One of the important facets of the framework has been the reconciliation of modularity objectives present in the literature and their subsequent arrangement into an objective hierarchy based upon both the technical and strategic considerations of product modularisation. In total six modularisation objectives have been identified (loose coupling, function binding, variety, outsourcing, maintenance and reliability, and recycling and reuse).

- *Development of modularity representation method and metrics.* The developed framework presents a multi-matrix based approach to represent the various dependencies and strategic similarities that occur between a product's components. In the framework the product is being represented in both the physical (component level) and functional domains. A cross-domain functional mapping approach allows the various modularity objectives to be analysed at two different levels of abstraction.

  Two modularity metrics have been developed for this research; these are the module independence ratio and the module coherence ratio, and these will be used as the criteria for module grouping during optimisation. These metrics are based upon two key modularity principles discussed in the literature. The

objective of modularisation from the module independence perspective is to achieve loosely coupled, independent modules. This can be achieved by ensuring that component dependencies are kept *within* modules rather than *between* modules. Module coherence is concerned with ensuring that components *within* modules are similar in terms of the modularisation objective they are addressing. In the proposed framework the module independence metric is used to as goal to improve the more technical aspects of modularity (function binding and coupling) whereas the module coherence metric is associated with the strategically aspects of modularity (variety, maintenance, recycling etc.).

- *Novel new multi-objective algorithm.* Because no suitable multi-objective genetic algorithm (MOGA) could be found to successfully handle the multi-objective product modularity problem, the author has developed a novel new algorithm (entitled MOGGA) for multi-objective optimisation of product modularity. The algorithm has been coded in VBA and integrated within a prototype software. The MOGGA incorporates a number of novel features such as the group based genetic operators, local search, random vector based search and Pareto-dominance based ranking. In has been shown in chapter 7 that the algorithm is able to produce a whole set of Pareto-optimal solutions, easier, faster and of equal quality than the individual solutions produced using a traditional aggregated objective single solution approach.

  It is also argued that there are other similar multi-objective grouping problems that the algorithm could be used to solve. Hence the algorithm itself presents a significant contribution to knowledge.

- *Software prototype for multi-objective product modularisation.* A prototype software has been created in an excel environment using VB coded macros to create a genetic algorithm (GA) based optimiser and a VB programmed user interface. The software has three main modules in which the various steps of the framework are undertaken. In the input module of the software the product is

decomposed at the physical and functional levels and component interactions are entered into a number of matrices using VBA based evaluation forms to define the interaction strengths for each modularisation objective. Once all the data has been input the optimisation module produces a set of Pareto-optimal solutions using the VB programmed MOGGA. The analysis module is then used to explore the solution set and choose the most suitable modular architecture. Again a number of VBA macros have been produced for this stage.

9.3. Future Work

Cost Based Evaluation of Modular Architecture Alternatives

At the moment the means of comparing and contrasting the alternative modular architectures is purely on the comparison of what their achievements are in each of the technical and strategic modularisation objectives. However, to add another dimension to the analysis, it would be interesting to develop a method of assessing the cost implications of the different modular structures. Cost factors that could be evaluated are assembly and disassembly costs, module replacement costs, the costs of implementing variety and cost saving from common platforms and estimated gains of recycling and reuse revenues though simplified product structures.

Application to Complex Products

So far the CAMO framework, and the majority of other frameworks for that matter, have been limited to the modularisation of relatively simple products. Thus future work could be done to apply the framework to complex products with a larger number of components.

Further Application of Domain Theory for Product Family Modularisation

During the development of the framework the ideas of cross domain mapping have been used. This is where the functional domain is mapped to the physical domain in a similar manner to axiomatic design. These principles have been used for the 'function binding' objective, in which components can be grouped into the same module if they are contributing to the same functions.

However, products are generally designed as part of a product family, where certain functions may be shared between products or where multiple physical solutions may exist for the same functions. This presents a problem for the matrix representation of the product, as they are only designed for single product representation. In fact previous matrix based modularity methods (CAMO included) presume that a generic physical component structure is present for all products in the family. To better deal with the complexity of multi-product design the modularisation framework needs to be developed further. A good place to start would be to produce a better method of domain mapping that deals with multiple products (product families) coupled with a suitable multi-product matrix representation. This could be done by developing a suitable representation method that clearly shows the mapping between customer needs, product functions and the physical components, for not just one product but multiple products.

Linking to a Modular Solution Repository

Linking the CAMO framework to a repository would allow chosen modular configurations to be stored. The DM would then be able to query/ search the repository to find existing design solutions (modules) that can be reused for new product developments or evolutionary product redesign. Furthermore modules that already exist could be used to introduce constraints to module formation during application of the CAMO to modularisation/ re-modularisation of products. A possible means of implementing such a system would be to produce a database system to store solutions, possibly within the MS Access environment. This would be relatively straightforward and could link to the current Excel based CAMO tool.

Semi- Automation of Objective Evaluations

The most time consuming aspect of the CAMO framework (and other frameworks for that matter) is the interaction evaluation step (step 2). When evaluating all six objectives this can take a considerable amount of time. If a complex product was to be evaluated for modularisation, then the process may well become unwieldy and impractical. In the CAMO framework steps have been taken to semi-automate the interaction evaluation of certain strategic modularity objectives, thus reducing the evaluation burden.

There are however ways in which further work could reduce the information needed for product modularisation. For the technical modularity evaluations, the CAMO framework could be linked to a knowledge based CAD system to automatically extract the relevant component interaction data straight from the geometric component mating data present in the CAD assembly model.

9.4. Concluding Remarks

- Many modularity definitions and methods exist in the literature, which although it enriches our understanding of the subject, also constitutes a major obstacle in understanding what product modularity actually is and how optimal modular product architectures are formed. Indeed as there are so many modularity objectives defined the process of modularisation can be seen as highly ambiguous. The developed framework has attempted to solve this problem by presenting a more holistic product modularisation method.

- There is a definite lack of algorithms able to solve complex multi-objective grouping problems. Hence the algorithm developed for this research presents a significant contribution to knowledge within the field of multi-objective optimisation.

- When attempting to modularise a product architecture according to multi-objectives there is often a potentially vast number of possible solutions. It is often impossible to find one solution that is optimal for all objectives. This inevitably means that compromises need to be made between different solutions. The framework presents an effective means to create a whole set of alternative solutions and compare and contrast these solutions to enable the decision maker to arrive at a more informed decision on the most appropriate modular architecture for their product.

References

Agard, B and Kusiak, A. (2004) “A Data-Mining Based Methodology for the Design of Product Families”, International Journal of Production Research, Vol. 42, No. 15, pp. 2955-2969.

Baldwin C.Y. and Clark K.B. (1997). “Managing in an age of modularity”, Harvard Business Review, Sept-Oct: 84-93.

Baldwin, C.Y. and Clark, K.B. (2000), “Design Rules, Volume I, The Power of Modularity”, MIT Press, Cambridge, MA, USA.

Beasley, D., Bull, D.R. and Martin, R.R. (1993). “An overview of genetic algorithms: Part 1, Fundamentals”, University Computing, 15(2):58-69, 1993.

Behr. 2010. www.behr.de, accessed January 2010, http://behr.de/internet/behrmm.nsf/lupgraphics/AufmacherbildBehr0051.jp/

Blackenfelt, M. (2001). “Managing complexity by product modularisation”, Doctoral Thesis, Royal Institute of Technology, Stockholm, Sweden.

Boothroyd, G., Dewhurst, P., and Knight, W. (1994). “Product Design for Manufacture and Assembly”, Marcel Dekker, Inc., New York, USA, ISBN: 0-8247-9176-2.

Brown, E C. and Sumichrast, R T. (2003) “Impact of the replacement heuristic in a grouping genetic algorithm”, Computers & Operations Research, Volume 30, Issue 11, September 2003, Pages 1575-159.

Carrascosa, M., Eppinger, S.D. and Whitney, D.E., (1998). “Using the Design structure matrix to estimate product development time”, Proceedings of ASME Design Engineering Technical Conference, September 13-16, Atlanta, GA, USA, Paper no. DETC98/DAC-6013.

Chakravarti, A.K. and Balkrishnan, N. (2001). “Achieving product variety through optimal choice of module variations”, IIE Transactions, 33, pp 587-598

Corne, D. and Knowles, J. (2007). “Techniques for highly multiobjective optimization: Some non-dominated points are better than others”. Proc. of 2007 Genetic and Evolutionary Computation Conference, 773-780. ACM Press, New York.

Dahmus, J.B, Gonzalez-Zugasti, J.P. and Otto, K.N. (2000). "Assembly quality method: a tool in aid of product strategy, design, and process improvements", Proceedings of ASME Design Engineering Technical Conference, September 10-13, Baltimore, MD, USA, Paper no. DETC2000/DTM-14565.

Dahmus, J.B, Gonzalez-Zugasti, J.P. and Otto, K.,N. (2001). "Modular product architecture", Design Studies, vol. 22, pp. 409-424, 2001. DETC2001/DAC-21125.

Deb, K. (2001). "Multi-Objective Optimization Using Evolutionary Algorithms", John Wiley & Sons, Chichester.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan T. (2002). "A fast and elitist multiobjective genetic algorithm: NSGA-II", IEEE Trans Evolutionary Computing 2002;6(2):182–97.

Deb, K. and Saxena, K. (2006). "Searching for Pareto-optimal solutions through dimensionality reduction for certain large dimensional multi-objective optimization problems", Proc. of 2006 IEEE Congress on Evolutionary Computation, 3353-3360.

Deb, K. and Sundar, J. (2006). "Preference point based multiobjective optimization using evolutionary algorithms", Proc. of 2006 Genetic and Evolutionary Computation Conference, 635-642.

Duray, R.,Ward, P.T., Milligan, G.W. and Berry, W.L. (2000). "Approaches to mass customization", Journal of Operations Management, 18, pp 605-625.

Erens, F. and Verhulst, K. (1997). "Architectures for product families", Computers in Industry, 33, 165–178.

Ericsson, A. and Erixon, G. (1999). "Controlling design variants: Modular product platforms", ASME Press. New York, NY.

Falkenauer, E. (1998). "Genetic algorithms and grouping problems", John Wiley & Sons Inc., Chichester, First edition.

Farrel, R.S. and Simpson, T.W. (2003). "Product platform design to improve commonality in custom products", Journal of Intelligent Manufacturing, 14, pp541-555.

Feitzinger, E. and Lee, H.L. (1997). “Mass customization at Hewlett-Packard: The power of postponement”, Harvard Business Review, 75, pp 116-121.

Fixson, S. (2003). "The Multiple Faces of Modularity – A Literature Analysis of a Product Concept for Assembled Hardware Products." Technical Report 03-05, Industrial & Operations Engineering, University of Michigan, Ann Arbor, MI: 87.

Fixson, S. K. (2005). “Product architecture assessment: a tool to link product, process, and supply chain design decisions”. Journal of Operations Management, 23(3/4) (Special Issue: Coordinating product design, process design and supply chain design decisions): pp345-369.

Fonseca, C.M. and Fleming, P.J. (1995). "An overview of evolutionary algorithms in multi-objective optimization", International Journal of Evolutionary Computation, Vol. 3 pp.1–15.

Fujita, K. and Yoshida, H. (2001). “Product variety optimization: simultaneous optimization of module combination and module attributes”, Proceedings of ASME Design Engineering Technical Conference, September 9-12, Pittsburgh, Pennsylvania, USA, Paper no. DETC2001/DAC-21058.

Fujita, K. (2002). “Product variety optimization under modular architecture”, Computer Aided Design, 34, pp 953-965.

Gershenson, J.K. and Prasad, G.J. (1997a). “Modularity in product design for manufacturing”, International Journal of Agile Manufacturing, 1(1), pp99–109.

Gershenson, J.K. and Prasad, G.J. (1997b). “Product modularity and its effect on service and maintenance”, Proceedings of the 1997 Maintenance and Reliability Conference (MARCON) (Knoxville, TN).

Gershenson, J.K., Prasad, G.J. and Allamneni, S. (1999). “Modular product design: a life cycle view”, Journal of Integrated Design and Process Science, 3(4), pp13–25.

Gershenson, J. K., Prasad, G. J., and Zhang, Y. (2004). “Product modularity: measures and design methods”. Journal of Engineering Design, 15(1): 33-51.

Goldberg, D.E. (1989). “Genetic algorithms in search, optimization, and Machine learning”, Reading, MA: Addison-Wesley

Gonzalez-Zugasti, J.P., Otto, K.N. and Baker, J.D. (1999). “Assessing value for product family design and selection”, Proceedings of 25th Design Automation Conference, ASME Design Engineering Technical Conference, September 12-15, Las Vegas, Nevada, USA, Paper no. DETC99/DAC-8613.

Gonzalez-Zugasti, J.P., Otto, K.N., and Baker, J.D. (2001). “Assessing value in platformed product family design”, Research Engineering Design, 13, pp 30-41.

Griffin, A. (2002). “Product development cycle time for business to business products”, Industrial Marketing Management, 31, pp 291-304.

Gu, P. and Sosale, S. (1999). “Product modularization for life cycle engineering”, Robotics and Computer Integrated Manufacturing 15, pp387-401.

Hajela, P. and Lin, C.Y. (1992). “Genetic search strategies in multicriterion optimal design”, Structural Optimisation, Vol. 4, pp99-107.

Hata,T. Kato, S., and Kimura, F. (2001) "Design of Product Modularity for Life Cycle Management," Ecodesign, pp.93, 2nd International Symposium on Environmentally Conscious Design and Inverse Manufacturing (EcoDesign'01).

Holland, J.H. (1975). “Adaptation in natural and artificial systems”, Ann Arbor: University of Michigan Press.

Holtta, K. and Otto, K. (2005). “Incorporating design effort complexity measures in product architectural Design and Assessment”. Design Studies. Vol 26, Issue 5, pp. 445-564..

Hsuan, J. (1999). “Impacts of supplier-buyer relationships on modularization in new product devlopment”, European Journal of Purchasing & Supply, 5, pp 197-209.

Huang, C.C, and Kusiak, A. (1999). “ Synthesis of modular mechatronic products: a testability perspective”, IEEE/ASME Transactions on Mechatronic 4(2), 119-132.

Hughes, E. J. (2005). “Evolutionary many-objective optimization: Many once or one many?”, Proc. of 2005 IEEE Congress on Evolutionary Computation, 222-227.

Ishibuchi, H. and Murata, T. (1996). “Multi-objective genetic local search algorithm”, Proceedings of the IEEE international conference on evolutionary computation, Nagoya, Japan: IEEE.

Ishibuchi, H., Yoshida, T. and Murata T. (2003). “Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling”, IEEE Trans Evol Comput 2003;7(2): pp204–23.

Ishibuchi, H., Doi, T. and Nojima, Y. (2006). “ Incorporation of scalarizing fitness functions into evolutionary multiobjective optimization algorithms”. Lecture Notes in Computer Science 4193: Parallel Problem Solving from Nature - PPSN IX, Springer, Berlin , pp493-502.

Ishibuchi, H., Tsukamoto, N. and Nojima, Y. (2008a). “Evolutionary many-objective optimization: A short review”, Proc. of 2008 IEEE Congress on Evolutionary Computation (in press).

Ishibuchi, H., Tsukamoto, N., Hitotsuyanagi, Y. and Nojima, Y. (2008b). “Effectiveness of scalability improvement attempts on the performance of NSGA-II for many-objective problems”. In Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (Atlanta, GA, USA, July 12 - 16, 2008). M. Keijzer, Ed. GECCO '08. ACM, New York, NY, pp649-656

Ishii, K. (1998). “Modularity: a key concept in product life cycle engineering”, Handbook of life cycle engineering, Kluwer, Dordrecht, pp511–531.

Jaszkiewicz, A. (1998). “Genetic local search for multiple objective combinatorial optimization”, Research report, Institute of Computing Science, Poznan University of Technology, RA-014/98, 1998.

Jaszkiewicz, A. (2002). “On the performance of multiple-objective genetic local search on the 0/1 knapsack problem – A comparative experiment”, IEEE Trans. on Evolutionary Computation 6, 4, pp402-412.

Jiao, J. and Tseng, M.M. (2000). “Fundamentals of product family architecture”, Integrated Manufacturing Systems, 11. pp 469-483.

Jose, A. and Tollenaere, M. (2005). "Modular and platform methods for product family design: literature analysis". Journal of Intelligent Manufacturing, 16(3): pp371-390.

Kamrani, A.K. and Gonzalez. F. (2003). "A genetic algorithm-based solution methodology for modular design". Journal of Intelligent Manufacturing. Vol 14, pp599-615.

Kapur, K.C. (1988). "Techniques of estimating reliability at design stage", Handbook of Reliability Engineering and Management, Eds.: Ireson W.G. and Coombs C.F. New York:

Kim, K. and Chajed, D. (2000). "Commonality in product design: cost saving, valuation change and cannibalization", European Journal of Operational Research, 125, pp 602 621.

Kinutani, H. (1997). "Modular Assembly in Mixed-Model Production at Mazda", in Transforming Automobile Assembly (eds. Shimowaka, K., Juergens, U., and Fujimoto, T.), Springer, Berlin.

Knowles, J.D and Corne, D.W. (2000) ."Approximating the nondominated front using the Pareto archived evolution strategy", Evol Comput 2000;8(2): pp149–72.

Knowles, J. Thiele, L. and Zitzler, E. (2006) "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers", TIK Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich,

Kreng, V., and Tseng, L., (2004). "Modular product design with grouping genetic algorithm: a case study", Computers and Industrial Engineering, Vol 46 No.3, pp443-460

Kreng, V.B. and Lee, T.P. (2004). "QFD-based modular product design with linear integer programming- a case study". Journal of Engineering Design, 15, pp261-284.

Krishnan, V. and Gupta, S. (2001)." Appropriateness and impact of platform-based product development", *Management Science,* Vol. 47, No.1, Pp 52-68.

Kukkonen, S. and Lampinen, J. (2007). "Ranking-Dominance and Many-Objective Optimization". IEEE Congress on Evolutionary Computation 2007: pp3983-3990

Kukkonen, S. and Deb, K. (2006). "A fast and effective method for pruning of non-dominated solutions in many-objective problems. In Proc. of the Int. Conf. on Parallel Problem Solving from Nature, volume 4193 of LNCS, pp553-562.

Kusiak, A. (2002). "Integrated product and process design: a modularity perspective, Journal of Engineering Design, 13, pp223-231

Kusiak, A., and Chow, W.S. (1987), "Efficient solving of the group technology problem. Journal of Manufacturing Systems", 6(2), pp117–124.

Lai, X. and Gershenson, J.K. (2008)."Representation of similarity and dependency for assembly modularity", The International Journal of Advanced Manufacturing Technology, Volume 3, Issue 7, pp803- 827.

Loch, C. H., Mihm, J. and Huchzermeier, A. (2003). "Concurrent Engineering and Design Oscillations in Complex Engineering Projects". Concurrent Engineering: Research and Application, 11(3): pp187-199.

Marks, M., Eubanks, C. and Ishii, K. (1993). "Life cycle clumping of product designs for ownership and retirement", Proceedings of ASME Design Engineering Theory and Methodology Conference, DE Vol. 53.

Marshall, R. (1998). "Design modularisation: a systems engineering based methodology for enhanced product realisation", PhD Thesis, Loughborough University, UK.

Martin, M.V. and Ishii, K. (2000). "Design for variety: a methodology for developing product platform architectures", Proceedings of ASME Design Engineering Technical Conference, September 10-13, Baltimore, MD, USA, Paper no. DETC2000/DTM-14021.

Meyer, M.H and Lehnerd, A.P (1997), "The power of product platforms", The Free Press, New Press, New York, USA.

Michalewicz, Z., Dasgupta, D., Le Riche, R.G. and Schoenauer, M. (1996) "Evolutionary Algorithms for Constrained Engineering Problems", Computers & Industrial Engineering Journal, Vol.30, No.2, pp851-870.

Mikkola, J. and Gassman, O. (2003). "Managing modularity of product architectures: Toward an integrated theory", IEEE Trans. Eng. Manag., Vol 50, No 2, pp204–218.

Mikkola, J.H. (2001). “Modularization Assessment of Product Architecture: implications for substitutability and interface management”, DRUID’s Nelson and Winter Conference, Aalborg, Denmark, June 12-15.

Moore, W.L., Louvuere, J.J. and Verma, R. (1999). “Using conjoint analysis to help design product platforms”, Journal of Product Innovation Management, 16, pp 27-39.

Muffato, M. (1999). “Introducing a platform strategy in product development”, International Journal of Production Economics, 60-61, pp145-153.

Muffato, M. and Roveda, M. (2000). “Developing product platforms: analysis of the development process”, Technovation, 20, pp617-630.

Nayak, R.U., Chen, W. and Simpson, T.W., (2000). “A variation-based methodology for product family design”, Proceedings of ASME Design Engineering Technical Conference, September 10-13, Baltimore, MD, USA, Paper no. DETC2000/DTM-14264.

Nepal,B. (2005) "An integrated framework for modular product architecture", PhD Dissertation, *ETD Collection for Wayne State University.* Paper AAI3198695.

Nepal, B. P., Monplaisir, L. and Singh, N., (2005). “Integrated fuzzy logic based model for product modularization during concept development phase”, International Journal of Production Economics, 96, 157-174.

Nepal, B. P., Monplaisir, L. and Singh, N. (2006). “A methodology for integrating design for quality in modular product design”, Journal of Engineering Design, 17.

Nepal, B. P., Monplaisir, L. and Singh, N. (2007). “A framework to integrate design for reliability and maintainability in modular product design” International Journal of Product Development 2007 - Vol 4, No 5, pp459 – 484.

Newcomb, P.J., Bras, B. and Rosen, D.W. (1996). “Implications of modularity on product design for the life cycle“, Proceedings of the 1996 ASME Design Engineering Technical Conferences, 8th International Conference on Design Theory and Methodology (Irvine, CA).

Nobelius, D. and Sundgren, A. (2002). "Managerial issues in parts sharing among product development projects: a case study", Journal of Engineering Technology and Technology Management Jet-M, 1113, pp1-15.

O'Grady, P. (1999). "The Age of Modularity - Using the new world of modular products to revolutionize your corporation", Adams and Steel Publishing.

Pahl, G. and Beitz, W. (1984). "Engineering Design: A systematic approach", Springer-Verlag, First edition, London, UK, ISBN: 3-540-19917-9.

Pimmler,T. and Eppinger, S.D. (1994). "Integration analysis of product decompositions", Proceedings of ASME Design Engineering Theory and Methodology Conference, DE Vol. 68.

Pine, B. J. I. (1993). Mass Customization. Boston, Massachusetts: Harvard Business School Press.

Pisa source code, 2010, http://www.tik.ethz.ch/sop/pisa/?page=selvar.php, accessed January 2010.

Qian, X. (2003). "Environmental analysis model for modular design of electromechanical products", PhD dissertation, Texas Tech University, USA.

Robertson, D. and Ulrich, K. (1998). "Planning for Product Platforms", Sloan Management Review (summer), pp19-31.

Saaty, T.L. (1980). "*The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation"*, ISBN 0-07-054371-2, McGraw-Hill.

Saaty, T.L. (2004). "Decision making – the analytic hierarchy and network processes (AHP/ANP)" Journal Of Systems Science And Systems Engineering, Vol. 13, No. 1, pp1-35.

Sako, M. and Murray, F. (2000). "Modules in Design, Production and Use: Implications for the Global Automotive Industry" A Paper prepared for the International Motor Vehicle Program (IMVP) Annual Sponsors Meeting 5-7 October 1999, Cambridge Massachusetts, USA.

Salvador, F., Forza, C. and Rungtusanatham, M. (2002). "Modularity, product variety, production volume, and component sourcing: theorizing beyond generic prescriptions", Journal of Operations Management, 307, pp 1-27.

Salvador, F., Rungtusanatham, M. and Forza, C. (2004). "Supply-chain configurations for mass customization". Production Planning & Control, 15(4): pp381-397.

Sanchez, R. (1999). "Modular architectures in the marketing process", Journal of Marketing, Vol.63, Special Issue, pp92-111.

Sanchez, R. (2002). "Using modularity to manage the interactions of technical and industrial design", Design Management Journal , pp 2.

Schaffer, J.D. (1985). "Multiple objective optimization with vector evaluated genetic algorithms" , Proceedings of the international conference on genetic algorithms and their applications.

Silveria, G.D., Borenstein, D. and Fogliatto, F.S. (2001). "Mass customization: literature review and research directions", International Journal of Production Economics, 72, pp1-13.

Simpson, T.W., Maier, J.R.A. and Mistree, F. (2001). "Product platform design: method and application", Research in Engineering Design, 13, pp 2-22.

Sosa, M. E., Eppinger, S. D. and Rowles, C. M. (2003). "Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions". Journal of Mechanical Design, 125(June): pp240-252.

Srinivas, N. and Deb, K. (1994) "Multiobjective optimisation using nondominated sorting in genetic algorithms", Evolutionary Computation, Vol. 2, pp. 221-248.

Stake, R. (2000). "On conceptual development of modular products: development of supporting tools for the modularisation process", Doctoral Thesis, Royal Institute of Technology, Stockholm, Sweden.

Steward, D. V. (1981). System Analysis and Management: Structure, Strategy, and Design. New York/Princeton: Petrocelli Books.

Stone, R.B. (1997). "Towards a theory of modular design", Ph.D. Dissertation, University of Texas at Austin, USA.

Stone, R.B., Wood, K.L. and Crawford, R. H. (2000). "A heuristic method for identifying modules for product architectures", Design Studies, Volume 21, pp5-31.

Suh, N.P. (1995). “Designing-in of quality through axiomatic design”. IEEE Transaction on Reliability, 44, pp256-264.

Suh, N.P. (2001). “Axiomatic Design: Advances and applications”, Oxford University Press, New York, ISBN: 0-19-513466-4.

Takeishi, A. and Fujimoto, T. (2001). “Modularization in the auto industry: interlinked multiple hierarchies of product, production, and supplier systems”, MIT-IMVP Publications.

Tate, D., Lindholm, D. and Harutunian, V. (1998). Dependencies in axiomatic design. Journal of Integrated Design and Process Technology, 3, pp159–165.

Tatikonda, M.V. (1999). “Using conjoint analysis to help design product platforms”, Journal of Product Innovation Management, 16, pp3-25.

Ulrich, K. (1995). “The role of product architecture in the manufacturing firm”, Research Policy, 24, pp419-440.

Ulrich, K. and Tung, K. (1991). “Fundamentals of product modularity”. In Issues in Design/ Manufacture Integration 1991, edited by A. Sharon (New York: ASME), DE 39.

Ulrich, K.T. and Eppinger, S.D., (1995). “Product Design and Development”, McGraw-Hill Inc., USA, ISBN: 0-07-065811-0.

Warburton, M. and Sako, M. (1999), "Modularization and outsourcing project – preliminary report of European Research Team", prepared for the IMVP Annual Forum.

Whitfield, R.I., Smith J.S. and Duffy, A.H.B. (2002). “Identifying Component Modules. Artificial Intelligence in Design” (AID’02), Cambridge, UK

Wilhelm, B. (1997). "Platform and modular concepts at Volkswagen – their effects on the assembly process", in Shimokawa, K., Jürgens, U., Fujimoto, T. (Eds),Transforming Automobile Assembly, Springer, London, pp.146-55.

Yassine, A., Tian-Li, Y. and Goldberg, D. (2003). "Developing Modular Product Architectures Using Genetic Algorithms," PD-Lab Working Paper, PDL-2003-02, Dec. 2003 York.

Zhang, Y. and Gershenson J.K. (2003). "An Initial Study of Direct Relationships between Life-cycle Modularity and Life-cycle Cost." Journal of Concurrent Engineering Research and Applications, Volume 11, Number 3.

Zitzler, E. and Thiele, L. (1999). "Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach", IEEE Trans. on Evolutionary Computation 3, 4, pp257-271.

Zitzler, E., Laumanns, M. and Thiele, L. (2001). "SPEA2: improving the strength Pareto evolutionary algorithm", Swiss Federal Institute Technology: Zurich, Switzerland.

Zitzler, E., Laumanns, M. and Bleuler, S. (2003). "A tutorial on evolutionary multiobjective optimization". Workshop on Multiple Objective Metaheuristics (MOMH 2002), Springer, Berlin Heidelberg New York

Publication List

Lee, M ,Case, K.and Marshall, R. (2009). "Product Modularity: A Multi-Objective Optimisation Approach" Proceedings of the 7th International Conference on Manufacturing Research (ICMR09) University of Warwick, UK, September 8-10, 2009.

Case, K., Lee, M. and Marshall, R. (2008) "Multiple Objective Optimisation for Product Modularity". Proceedings of the 25th International Manufacturing Conference (IMC25), 'Manufacturing and Design: The Next Generation' (ed R Simpson), pp361-388, ISBN 1-0900454-28-9, 3rd– 5th September 2008, Dublin Institute of Technology, Ireland.

Lee, M, Case, K and Marshall, R. (2008) "Multiple Objective Optimisation of Product Modularity". Proceedings of the The 18th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2008),pp187-197, Skovde, Sweden.

Lee, M ,Case, K.and Marshall, R. (2007). "Methodology for the Multi-objective Optimisation of Product Modularity" Proceedings of the 5th International Conference on Manufacturing Research, (ICMR 2007),11th - 13th September 2007, De Montfort University, UK