MASTER'S THESIS

A User Interface for Discrete Optimization-Based Tradeoff Analysis

by John M. Splain
Advisor: Michael O. Ball

M.S. 98-2

ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site http://www.isr.umd.edu

ABSTRACT

Title of Thesis: A USER INTERFACE FOR DISCRETE OPTIMIZATION-BASED TRADEOFF ANALYSIS

Degree Candidate: John M. Splain

Degree and year: Master of Science, 1998

Thesis directed by: Dr. Michael Ball
College of Business and Management
Institute for Systems Research

The motivation of this thesis is to perform research in the arena of discrete tradeoff analysis. There are two fundamental aspects to the problem domain to be studied. The first involves modeling the manner in which the user quantifies the relative merits of alternative solutions. This requires creating a graphical user interface that presents the solution in an easily understandable form. The second aspect of this research involves structuring the interaction between the optimization model and the user interface.

The research was conducted within the context of a manufacturing application that is centered on the optimization of product and process design. In addition to designing the overall system, the essential design activities revolve around building the graphical user interface necessary to present the tradeoff solutions

as well as the interface that will permit the user to interact with the optimization algorithm.

A USER INTERFACE FOR DISCRETE OPTIMIZATION-BASED TRADEOFF ANALYSIS

by

John M. Splain

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Master of Science
1998

Advisory Committee:

Dr. Michael Ball, Chair
Dr. Dana Nau
Jeffrey Herrmann

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DLL	Dynamic Link Library
GUI	Graphical User Interface
IP	Integer Programming
MIC	Microwave Integrated Circuits
PCB	Printed Circuit Boards
PFC	Powersoft Foundation Class
RAD	Rapid Application Development
SQL	Structured Query Language

CHAPTER 1

INTRODUCTION

This chapter defines the problem and motivation for this thesis, discusses the design approach for the software development aspect of the research, and provides a review of the literature.

Problem Definition and Motivation

The motivation for this thesis stems from two separate, yet similar, efforts to determine designs for printed circuit boards (PCB) (Ball et. al. 1995) and microwave integrated circuits (MIC) (Trichur et. al. 1996). Each of these efforts is concerned with solving a multicriteria optimization problem defined to address decision problems that arise because PCBs and MICs comprise many components such as transistors and capacitors. Each of these components has several attributes such as the process used to fasten the component to the assembly, the cost of the component, and the quality rating of the component. The final design is composed of a discrete combination of these components and is ultimately affected by their individual attributes.

To complicate matters, some measures of overall solution "goodness" may be complimentary while others may not. For example, cost and quality are frequently directly proportional, i.e. higher quality components tend to cost

more than lower quality ones. However, from an optimization perspective these two attributes are at odds. Generally, a designer wants to minimize cost while maximizing quality.

The goal of this thesis is to design and compare alternate graphical user interface (GUI) strategies which allow a user to control a discrete optimization-based tradeoff analysis. It also addresses the design of a system/software architecture for decision support systems of this type.

At each major step in the overall process, a discrete optimization problem is solved. The GUI enables the user to iterate through the process, changing the relative weights of each criterion like cost and quality. The GUI then displays the results of the optimization process graphically so the user may determine the consequences of increasing or decreasing specific weights. This process can be repeated until the user is satisfied he/she has reached an optimal design for the problem under consideration.

Design Approach

Several design options were entertained at the beginning. Black box methods were used in order to achieve a modular design that is independent of both the underlying database management system and the optimization engine. Despite initially being driven by manufacturing examples, the modular

structure of the design and the implementation enable the results to be used to address any type of multicriteria optimization problem.

Upon determining the top-level design of the overall system, rapid application development techniques were employed to prototype and demonstrate functionality as well as define additional requirements. This process can continue until the interface is fielded to the end users for acceptance testing.

Traditionally, optimization tools produce the results of a problem solving session as a table of numerical values. These values can then be imported into a spreadsheet program and graphed. The goals of this thesis are to study the design of graphical display methods and graphical user interfaces for discrete optimization-based tradeoff analysis that employ normalized ranges for comparing criteria and the comparison of software architectures for such systems.

Literature Review

Linear programming is a special class of optimization problems (Ringuest 1992, Cohon 1978, Sundaran 1996, Shoup and Mistree 1987) where the objective function is expressed as a linear relationship of the design variables and the constraints are expressed as linear equalities or inequalities, as shown in the following:

$$\text{Min } f = \sum_{i=1}^{n} c_i x_i$$

$$\text{subject to } \sum_{i=1}^{m} a_{mi} x_i \leq b_i$$

Typically, given a problem of this type, multicriteria tradeoff analysis consists of a composite merit function where each of the individual functions is assigned a weight based on its perceived relative importance. For example, assume three merit functions and a composite merit function as follows:

$$f_i \text{ where } i = 1, 2, 3$$

$$F = c_1 f_1 + c_2 f_2 + c_3 f_3$$

If f_1 is perceived to be five times more relevant than f_2 and f_3, it's weight, c_1, is assigned 0.5, and c_2 and c_3 are assigned 0.1.

An integer program is a linear program with the added restriction that the variables can only take on integer values. Integer programming is frequently applied in applications where decisions are essentially discrete, e.g., yes/no, go/no go, such that only one alternative may be selected. This is relevant for the application domain of this thesis since, for example, a part is either included in a PCB or it is not.

Providing an interactive approach to multicriteria optimization serves two purposes. On the face of it, it allows the user to directly interact with the optimization problem. However, a secondary benefit that is realized is the

ability to perform sensitivity analysis. When the user believes an optimal solution has been reached, the solution can be tested to determine whether the solution is stable. If the solution is unaffected by relatively large changes in the input parameters, then it can be safely assumed to be meaningful. If, however, the solution is affected by changes to the input parameters, its value should be viewed as compromised.

As later discussion will make evident, this thesis employs a different approach to multicriteria tradeoff analysis. The relative merit of a criterion is based upon the range of good and bad values for that criterion. The user manipulates the relative importance of different criteria by altering the good and bad values. Instead of applying arbitrary weights to each merit function f_i, the approach used in this thesis is to minimize the maximum normalized deviation from the bad value for each criterion, as shown in the following formulas:

$$f_i^{norm} = \frac{f_i^{nominal} - f_i^{bad}}{f_i^{good} - f_i^{bad}}$$

$$\text{(1)} \qquad \text{Min } \{\text{Max}_i (f_i^{norm})\}$$

f_i^{good} is a target good value for criterion i. f_i^{bad} is a limit on the worst acceptable value for criterion i. Note that the smaller the normalized range the larger the importance given to a criterion.

This thesis was motivated by two specific examples of multicriteria discrete optimization, but is part of a more general framework (Trichur and Ball 1998). In their discussion, they review a model for describing a hierarchical system by means of an AND-OR tree. They apply this model to the design of microwave modules, which consist of MICs, and PCBs. These models result in the formulation of integer programming (IP) problems, which are used as the basis for this thesis.

Simplicity and usefulness are described as the preeminent goals of selecting and developing a tradeoff analysis model (Blanchard and Fabrycky 1990). Some characteristics of a successful model include reliability in the repeatability of results, timely and efficient implementation, and ease of modification and/or expansion. The model should be simple enough to understand easily, but remain close enough to reality so as to yield meaningful results. Factors that may be deemed more relevant than others should be called to the fore, while others are suppressed (Tufte 1990). Tufte refers to this as “layering and separation”, a design strategy that reveals the detail and complexity of the data instead of dismissing the data as overly complicated. A favorable design will eliminate non-data detail and focus attention on the data itself (Tufte 1983). For example, by eliminating lines connecting data points on a graph, the data points themselves can be seen more clearly and invalid relationships intimated by the lines are removed.

This thesis addresses these traits by permitting the user to directly interact with the process of solving an optimization problem in a visual manner, instead of extrapolating mentally from tabular data. A modular design is employed to keep modifications in one module of the software tool from affecting the others. For instance, the data store used to house information about the problem can be changed from flat files to a relational database without requiring changes to the user interface. Since the software tool is visual, cues are given to the user to aid in the problem solving process. Criteria that exceed certain bounds are flagged and the results of each optimization iteration are presented graphically to allow the user to see immediately the effects of changes to specific weights.

Due to the relatively small size and limited number of users of the software tool developed for this thesis, a Rapid Application Development (RAD) approach was employed. An understanding of the users and tasks (Galitz 1997) associated with multicriteria optimization-based tradeoff analysis was adapted to the visual domain of a graphical user interface. By prototyping the design, various techniques for displaying information to the user were tested and refined (Martin and Eastman 1996). This process of testing, revising, and repeating allowed for the creation of a software tool that is composed of highly cohesive, tightly coupled windows (Shneiderman 1998). The importance of this is that users are presented with an interface where

modifications made in one part of the tool are reflected in other, related parts of the tool. For example, there is one window that is considered to be the main driver of the software tool. The other windows have meaning only within the context of this first window. If the user closes the first window, the other, dependent windows are also closed. Similarly, when the user resumes his/her problem solving efforts and restarts the tool, the windows are all restored to their previous states.

CHAPTER 2

SYSTEM ARCHITECTURE

This chapter discusses the basis for decomposing hierarchical systems into subsystems, introduces the concept of AND-OR trees to aid in mathematically describing a hierarchical system, and describes the architectures of the software tool developed for this thesis and the system upon which it was based.

Mathematical Motivation

By definition, a hierarchical system is a system that can be decomposed into subsystems that are further decomposed into sub-subsystems. This recursive decomposition continues until the subsystems are atomic elements that cannot decomposed any further (Figure 1). System A can be decomposed into subsystems B and C. Subsystem B can be decomposed into subsystem D and atomic element A3. Subsystem D can be decomposed into two atomic elements, A1 and A2. Similarly, subsystem C can be decomposed into atomic elements, A4 and A5.

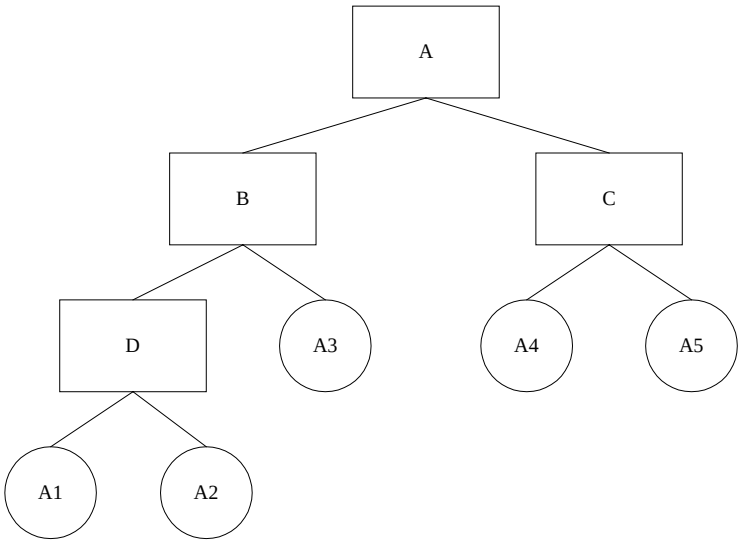

Figure 1 - A Hierarchical System

The concept of an AND-OR tree expands upon the hierarchical decomposition, which is comprised only of “AND arcs”, by introducing “OR arcs”, e.g., the arcs cutting across the lines connecting system A and subsystems B and C (Figure 2). The significance of these arcs is defined in the following manner. System A is composed of subsystems B and C, as indicated by the AND-arc. Subsystem B is composed of either subsystem D or E. Subsystem D is composed of atomic elements A1 and A2. Subsystem E is composed of atomic elements A3 and A4. Subsystem C is composed of either atomic element A5 or A6.

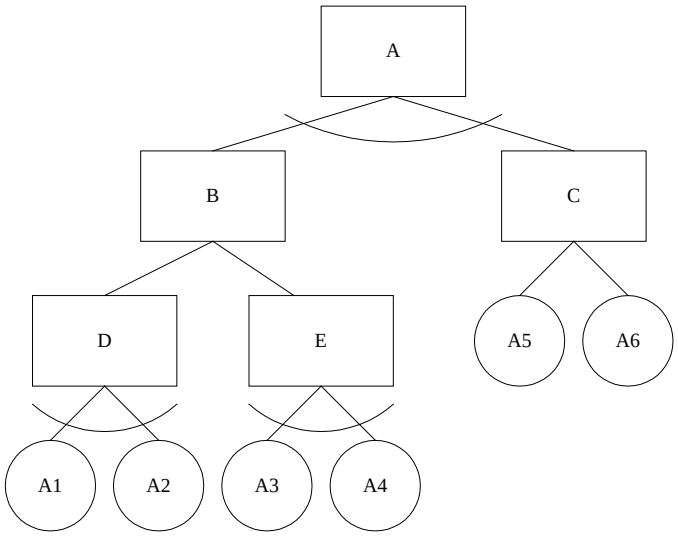

Figure 2 - An AND/OR Tree

A decision model can be associated with any AND-OR tree by defining the concept of AND-nodes, OR-nodes, dummy nodes, associated decision variables, and a corresponding system of equations. The structure of the decision model is as follows. If an AND-node is chosen then all of its children must be chosen. The AND-nodes (Figure 2) are A, D, and E. If an OR-node is chosen then one of its children must be chosen. The OR-nodes (Figure 2) are B and C. Dummy nodes are inserted into an arbitrary AND-OR tree in order to transform it into an AND-OR tree of the standard form, one that contains only AND-nodes and OR-nodes.

With these definitions, the following notation can be used to provide a mathematical definition of the decision model associated with an AND-OR tree.

ν_{AND} = set of AND-nodes

ν_{OR} = set of OR-nodes

ν = $\nu_{AND} \cup \nu_{OR}$ = set of nodes in the tree

ROOT = the root node of the tree

CHILD(i) = set of nodes that are children of node i

The decision variables are:

$$x_i = \begin{cases} 1 & \text{if node i is selected to be in the system, } i \in \nu \\ 0 & \text{otherwise} \end{cases}$$

and $x_{root} = 1$, because the root node is always selected.

The AND-constraints are denoted by the following:

$$x_i = x_j \qquad \forall i \in \nu_{AND} \qquad j \in CHILD(i)$$

The OR-constraints are denoted by the following:

$$\sum_{j \in CHILD(i)} x_i = x_j \qquad \forall i \in \nu_{OR}$$

CONSOL-OPTCAD Architecture

The starting point for the design used in creating the software tool was the CONSOL-OPTCAD system (Fan et. al. 1992) (Figure 3).

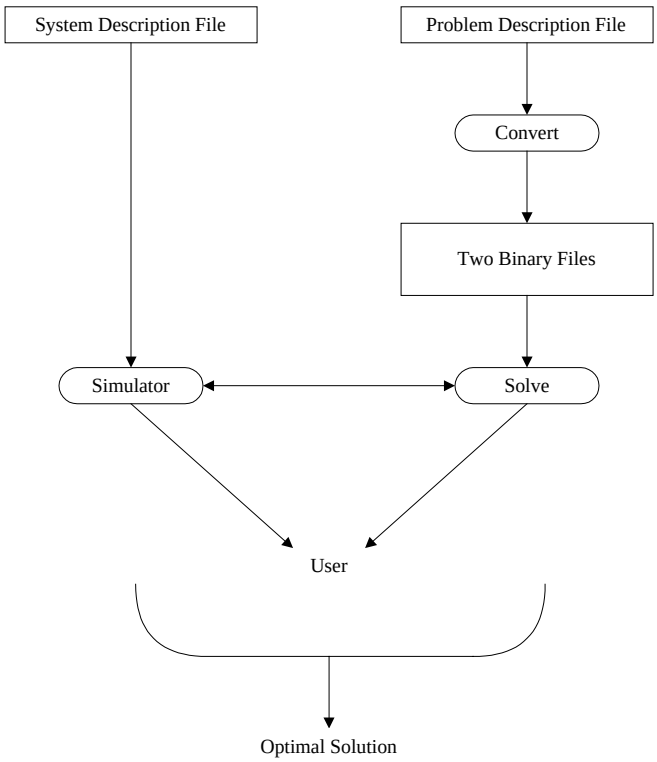

Figure 3 - Structure of CONSOL-OPTCAD

CONSOL-OPTCAD is a tool for interactively designing systems involving non-linear objective functions and constraints with continuous values. It has been applied to the design of circuits, controllers of a flexible arm, high performance aircraft, and robotic manipulators, and the determination of the optimal flow rate and temperature profile for a co-polymerization reaction.

CONVERT (Figure 3) is a batch program that takes as input a Problem Description File, which contains a description of the optimization problem to be solved. This input file is compiled into two binary files which are used as input by SOLVE. One of these files contains the names of design parameters and specifications and the "good" and "bad" values to be used for the tradeoff analysis. The other file is a compiled version of the various objectives, functional objectives, constraints, and functional constraints. The user interacts with SOLVE and any appropriate SIMULATORs to solve the optimization problem.

CONSOL-OPTCAD makes use of hard and soft constraints to define the boundaries of each objective or criterion within a specified problem domain. Hard constraints must be satisfied, otherwise the solution has no inherent value. Soft constraints should be achieved or, at least, be close to being satisfied. CONSOL-OPTCAD employs a "good" and "bad" value for each criterion. The good value is the value the user would like the raw value to approach, while the bad value is just the opposite.

A scaled value is achieved by normalizing the raw value with respect to the good and bad values for a particular criterion according to the formula:

$$\text{scale\_value} = \frac{\text{raw\_value} - \text{good\_value}}{\text{bad\_value} - \text{good\_value}}$$

When the bad value is achieved, the scaled value is 1, and when the good value is achieved, the scaled value is 0. Fan et. al. define the optimization problem (P) as follows:

$$\begin{aligned} \text{Min}_{\ x} \quad & \text{obj}_k(x)\ \forall k \\ \text{subject to} \quad & \text{soft}_i(x) \le 0\ \forall i \\ & \text{hard}_j(x) \le 0\ \forall j \\ & \text{hard\_bound}_l(x) \le 0\ \forall l \end{aligned}$$

where obj_k, soft_i, and hard_j, respectively, are scaled values of objectives, soft constraints, and hard constraints, and where hard_bound_l represents the hard bound type constraints. Based upon the feasibility or infeasibility of x with respect to hard and soft constraints, problem (P) is assigned three different meanings, which are derived from satisfying the constraints and objectives to varying degrees as described below.

When not all hard constraints are satisfied, (P) takes the form:

$$\begin{aligned} \text{Min}_{\ x} \quad & \{\text{Max}_{\ j}\ \text{hard}_j(x)\} \\ \text{subject to} \quad & \text{hard\_bound}_l(x) \le 0\ \forall l \end{aligned}$$

When all hard constraints are satisfied and not all scaled objectives and soft constraints are non-positive, (P) takes the form:

$$\begin{aligned} \text{Min}_{\ x} \quad & \{\text{Max}_{\ k,\ i}\ \text{obj}_k(x), \text{soft}_i(x)\} \\ \text{subject to} \quad & \text{hard}_j(x) \le 0\ \forall j \\ & \text{hard\_bound}_l(x) \le 0\ \forall l \end{aligned}$$

When all hard constraints are satisfied and all scaled objectives and soft constraints are non-positive, (P) takes the form:

$$\begin{aligned} \text{Min}_{\ x} \quad & \{\text{Max}_{\ k}\ \text{obj}_k(x)\} \\ \text{subject to} \quad & \text{soft}_i(x) \le 0\ \forall i \\ & \text{hard}_j(x) \le 0\ \forall j \\ & \text{hard\_bound}_l(x) \le 0\ \forall l \end{aligned}$$

Tradeoff Analysis Utility Architecture

The software tool created as part of this thesis (Figure 4) relies on two, related input files, a Linear Programming file and a Problem Information file.

The Linear Programming file is produced as the output of a generator that takes as input files that completely describe the problem to be optimized. A Linear Programming file contains the definition of the objective functions and constraints to be solved for by the optimization engine. For the example of a PCB or MIC, these files include the following information: the list of parts

on the board; the alternatives for each of these parts; detailed parts information including part ID numbers, cost (in dollars), defect rate (a number between 0 and 1), supplier ID, the number of groups of processes to be performed on the part, the number of processes precluded by the part, the number of process alternatives, the incremental time for each process ID, and the precluded

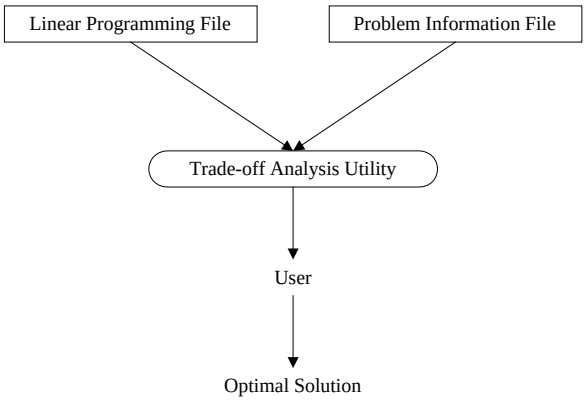

Figure 4 - Structure of Tradeoff Analysis Utility

processes; process information including the total number of processes and the setup time and yield for each process ID; supplier information including the number of suppliers and the lead time associated with each supplier; and miscellaneous information like the batch size associated with the design and the labor cost.

The Problem Information file (Figure 5) contains information about the criteria that comprise the optimization problem. For each of these criteria, the file contains the type of the criterion, e.g., “maximizing” or “minimizing”; the units of the criterion; and a bound for the criterion. The type of the criterion is

used to differentiate between criteria like cost, which is considered a minimizing criterion because it's value it minimized, and quality, which is considered a maximizing criterion because it's value is maximized. The units are used for display purposes in the user interface. The bound for the criterion is discussed below.

```
[number]
criteria=4

[criteria_1]
name="Cost"
type=”Min”
bound="2500"
units=”Dollars”

[criteria_2]
name="Quality"
type=”Max”
bound="0.4"
units=”%”

[criteria_3]
name="Lead Time"
type=”Min”
bound="10"
units=”Hours”

[criteria_4]
name="Number of Suppliers"
type=”Min”
bound="1"
units=””
```

Figure 5 - Sample Problem Information file

The normalization scheme used for this thesis was based on the scheme used by CONSOL-OPTCAD, but differs in the following ways. All the

constraints specified for a problem definition are hard constraints. The constraints are set to good and bad values based upon the contents of the Problem Information and the results of initializing a new optimization problem.

The scaled value for a criterion is obtained by normalizing the raw value with respect to the hard constraints. In the case of a minimizing criterion, e.g., cost, normalization occurs as follows:

$$\text{scale\_value} = \frac{\text{raw\_value} - \text{bad\_value}}{\text{good\_value} - \text{bad\_value}}$$

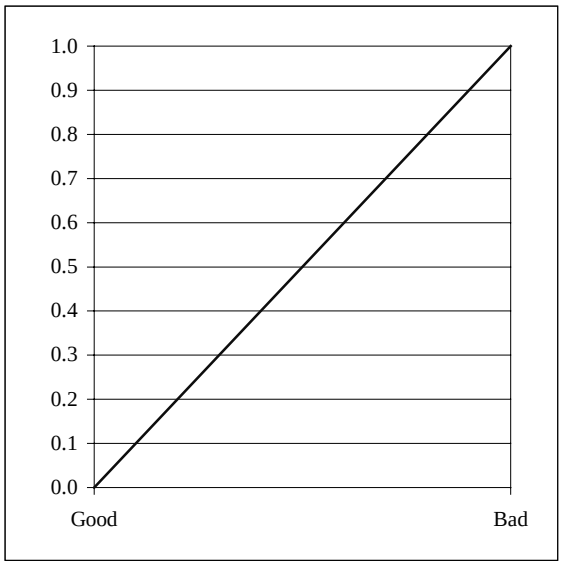

Figure 6 - Range of Normalized Values for a Minimizing Criterion

In the case of a maximizing criterion, e.g., quality, normalization occurs as follows:

$$\text{scale\_value} = \frac{\text{raw\_value} - \text{good\_value}}{\text{bad\_value} - \text{good\_value}}$$

Figure 7 - Range of Normalized Values for a Maximizing Criterion

The two different types of criteria, maximizing and minimizing, are differentiated in this manner so they may be compared equivalently. Notice that for the solution obtained when the problem is solved for a single criterion, the scale_value is 0, regardless of whether it is a maximizing or minimizing criterion. The crux of this is that desirable values for a criterion normalize towards 0, and undesirable values normalize towards 1 (Figure 6 and Figure 7).

CHAPTER 3

COMPARISON OF ALTERNATE SOFTWARE ARCHITECTURES

This chapter examines the design alternatives identified as part of this thesis with respect to the design goals of modularity and database independence.

Design Goals

Three alternatives in the overall system architecture were evaluated. Each one is detailed below with respect to two design goals: modularity and database independence. Modularity is what is achieved when a complex problem is divided into simpler pieces called modules (Pressman 1992). This is sometimes referred to as a "divide and conquer" strategy to system design. A user interface is database independent if it is unaffected when the means of storing information is changed, e.g., when relational database tables are substituted for text files.

The goals of a modular design (Ghezzi, Jazayeri, and Mandrioli 1991) are to decompose a complex problem into intellectually manageable parts and to attempt to compose the solution to the problem from existing modules. Decomposing a complex problem into modules is done in a manner similar to that illustrated for decomposing a hierarchical system. The problem is divided

into sub-problems in a top-down fashion, and the sub-problems are recursively divided as well. One of the by-products of a modular design is an increased capability of understanding. For example, a designer is much more likely to understand a modular system than a monolithic one, because the smaller sizes of individual modules will reduce the number of control paths, the number of variables, and the span of reference of these variables.

Two measurable attributes of a modular design are cohesion and coupling. Cohesion is a measure of how strongly related the modules are. A cohesive module is one in which the elements of the module are all logically related. Coupling is a measure of the interdependence of two modules. Modules that exhibit low coupling are independent of one another, while modules that exhibit high coupling are dependent on one another. Low coupling is desirable because it limits the "ripple effect" or propagation of changes when a module is modified. As part of a collaborative effort in the arena of integrated product and process design, this thesis is concerned with minimizing the ripple effect so as not to affect other aspects of the effort.

Database independence is desirable because it allows the designers of a software tool to treat database access as a black box. In a very simple system, the user interface can be thought of as a module that is primarily concerned with interacting with the user. A second module that is concerned only with database access can be used to model database access. By definition, these

two modules will exhibit high cohesion and low coupling. Low coupling allows any database access module to be plugged into the system without requiring changes to the user interface module.

Design Alternatives

The first design alternative (Figure 8) is composed of an Object Store database, a Problem Extraction module, a graphical user interface, and a dynamic link library (DLL).

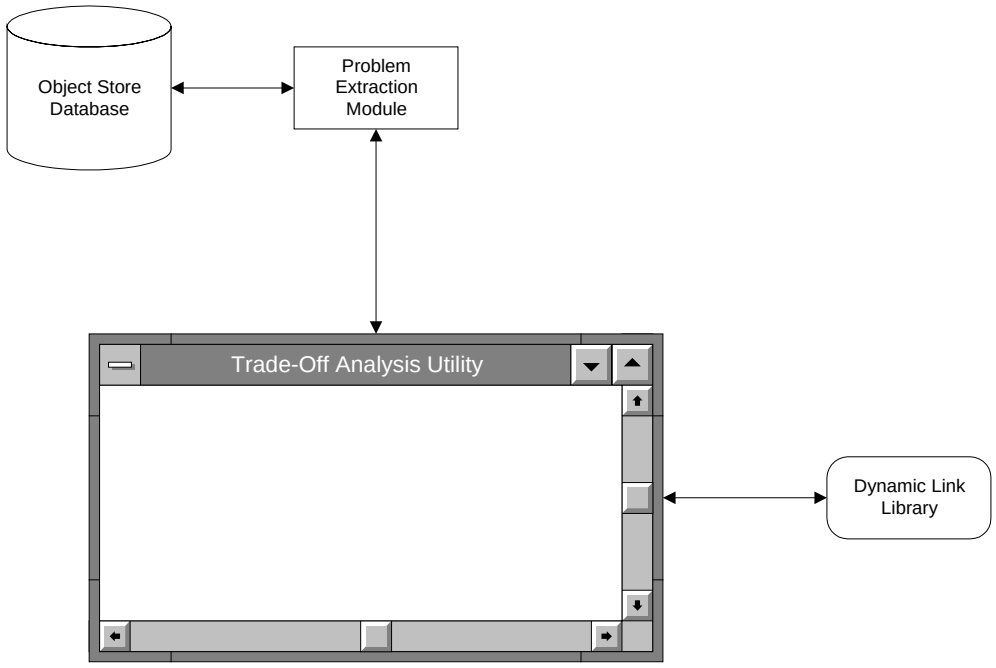

Figure 8 - First Design Alternative

The primary function of the Object Store database is to serve as a repository for information that defines the optimization problem. For example, for an optimization problem involving a PCB this includes information about

parts, suppliers, and processes. The Problem Extraction module was identified as the mechanism for retrieving this information from Object Store and converting it to a format that could be ingested by the optimization engine to solve the problem within a specific context. The optimization engine is accessed by calling functions that are compiled into the dynamic link library.

The second design option (Figure 9) comprised similar elements of a graphical user interface, Object Store database, and an interface to CPLEX for the optimization algorithms, again embodied within the DLL.

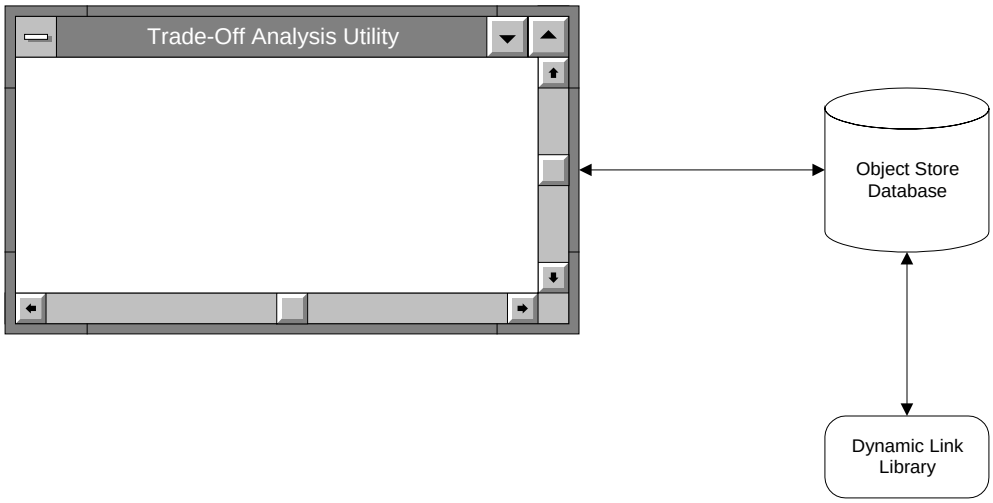

Figure 9 - Second Design Alternative

As before, the Object Store database is to function as a repository for information that defines the optimization problem. However, in this alternative the user interface interacts directly with Object Store to instruct it to solve a problem by calling the optimization engine. That is, the Object Store database contains active components that generate calls to the DLL based on

database updates made by the user interface. To improve this design, each component was placed in its own C++ container to foster language independence and to reduce dependence on specific product vendors.

The third design option (Figure 10) is a database-independent solution that divides the architecture into three layers.

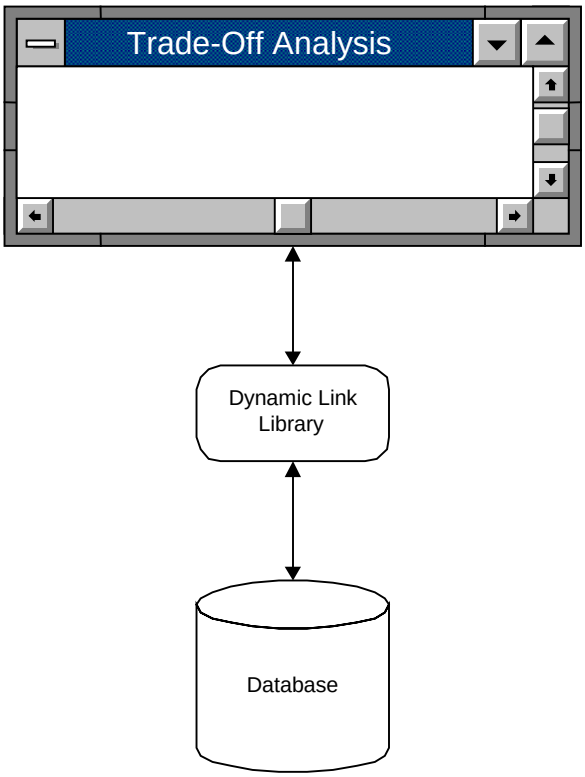

Figure 10 - Third Design Alternative

The top layer consists of the graphical user interface. The middle layer is a dynamic link library of function calls to access the database and/or the optimization algorithm. The bottom layer is the database and optimization algorithm. Although pictured here as a database, the data store used by the

optimization engine could be any means of storage available, e.g., text files, a relational database, or an object database.

The user interface maintains a local database to keep track of the criteria for the current problem definition and the solutions generated during a problem solving session. It calls functions compiled into the DLL to perform such functions as initializing a problem definition, re-solving the problem within a new context, or deleting a solution. The DLL is implemented to interface with the appropriate database(s) and optimization engine.

Comparison of Design Alternatives

The first design alternative is composed of four cohesive modules that have reasonably low coupling. Each module is dedicated to an individual aim: the user interface, problem extraction, database access, and the optimization engine. However, this design places the bulk of the processing in the user interface. The interface calls the Problem Extraction module to format everything needed by CPLEX to optimize the problem within the current context as defined by the user. It then calls routines in the CPLEX library to solve the problem. Finally, it must write the solution back to the database. One advantage to this design is that it minimizes the degree to which the DLL must be customized for this problem. Thus, new optimization engines can be more easily substituted for one another. While this design separates the details

of the user interface, the optimization engine, and the data store, this alternative is dependent on Object Store as its means of storage.

The second alternative is comprised of one less module than the first. This was achieved by consolidating the Problem Extraction module and database access module. Additionally, the responsibility of interfacing with the optimization engine was moved from the user interface into the database access module. As in the first alternative, the optimization engine is accessible via the DLL. Although this is a modular solution, the degree of cohesion within each of the modules is less than that achieved by the first design because the database access module is responsible for performing functions related to optimization. This design is also dependent upon the database, which might be viewed as advantageous under certain circumstances. Since the database is aware of the state of the software tool, any function can be invoked that interacts with the data stored in the database. This leads to the possibility that alternate user interfaces and applications could be built over the database to interact with the optimization engine and related data files.

The third alternative meets both design goals of modularity and database independence by employing three highly cohesive modules that are loosely coupled. The user interface does not require modification to use different database management systems or to take advantage of different optimization engines. This implementation of the graphical user interface is

independent of the underlying data representation and algorithm implementation. The user interface is designed with a predefined set of function prototypes for initializing, solving, and deleting solutions for optimization problems. Provided the functions compiled into the DLL comply with these function prototypes, any combination of compiler, database, and optimization engine can be used to suit the environment in which the software tool will be deployed. This design alternative was selected as the basis for the software tool because it supports the greatest degree of flexibility.

The results of this comparison are summarized below (Table 1).

Alternative #	Cohesion	Coupling	Modular	Database Independent	Optimization Engine Independent
1	High	Low	Yes	No	Yes
2	Medium	Medium	Yes	No	No
3	High	Low	Yes	Yes	Yes

Table 1 - Summary of Design Alternatives Comparison

CHAPTER 4

USER INTERFACE DESIGN

This chapter describes the technique used to develop and test the user interface, the final design of the software tool, and the details of the tool's implementation.

Development Technique

A Rapid Application Development technique was employed to develop the user interface. A small set of initial requirements was identified and development was begun. As these requirements were completed, the prototype was demonstrated and additional requirements and enhancements were identified. This process could be continued to produce a final product that is fielded to the end users for acceptance testing.

Final Design

The principle motivation for the use of scaled values is to allow the user to tradeoff criteria of different units and scales equally. For instance, the user can decide whether an increase of y% of the scaled value of one criterion is equitable against an x% increase of the scaled value of a second criterion,

where x > y. That is, within the scaled ranges, do the scaled values tradeoff equally.

When the user runs the software tool, he/she can either start analyzing a new problem or continue with an existing one by selecting Open from the File menu. The tool then displays a dialog box of the available Problem Information files for the user (Figure 11).

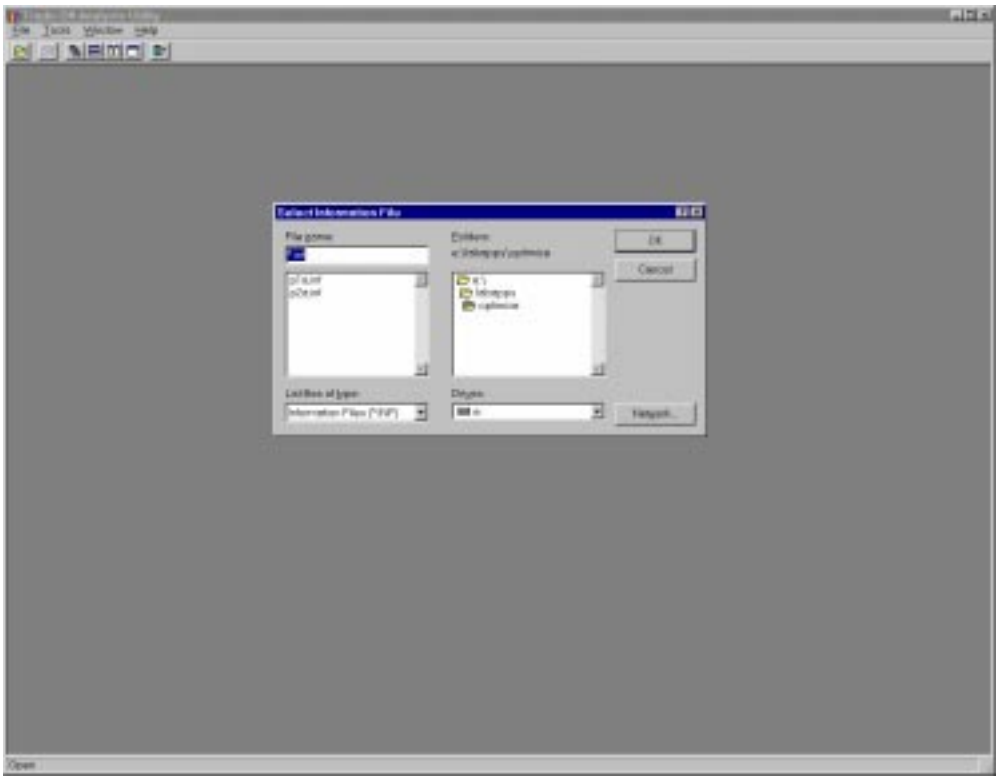

Figure 11 - Starting an Optimization Process

When the user selects a Problem Information file, the software tool determines if the user is continuing with an existing problem or beginning a new one. If the problem is an existing one, the tool is restored to the state in

which the user left it previously. If the problem is a new one, the tool initializes the problem and allows the user to select a starting point for iterating the optimization process. The Problem Information file is read and stored for use during the user's session.

The initialization process involves optimizing the problem once for each criterion without regard to the others, i.e., minimizing f_i rather than f_i^{norm} as was indicated in (1). For example, if the criteria that define a particular problem include cost, quality, and lead time, the problem is initialized by optimizing for cost alone, then optimizing for quality alone, and finally optimizing for lead time alone.

When initialization is complete, the user is presented with a graph that displays the initial solutions for the problem (Figure 12). There is one solution for each criterion, the result of solving the problem while optimizing for only that criterion. The user can compare these solutions relative to one another. By presenting the information graphically, the user can see which, if any, of the initial solutions may be more desirable than the others. The user can then begin the process of iteratively solving the problem by selecting this solution.

An additional feature of this graph of initial solutions is the user can immediately see the best possible value for each criterion. While not entirely accurate, the user can gain an intuitive sense of the "best" overall possible

solution that might be achieved if it were possible for all the criteria to achieve their optimal values simultaneously.

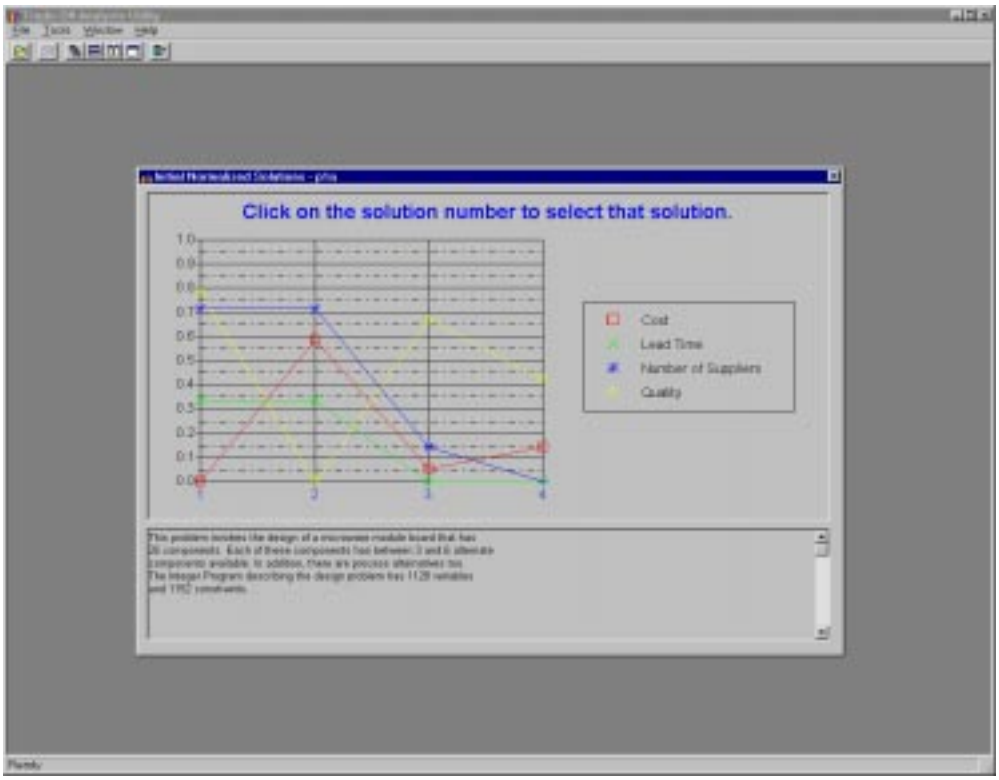

Figure 12 - Initial Normalized Solutions Window

The first attempt at providing the user with a starting point was to display a list of the criteria that define the optimization problem. When the user selected one of the criteria from the list, the solution associated with optimizing the problem for that criterion alone was chosen as the starting point for further iterations. When this method was tested, it was quickly determined that it was not intuitive to the user. By simply selecting a criterion by name, the user had no idea how the individual solutions achieved during initialization

compared to each other. This lead to the current method of providing a starting point, the Initial Normalized Solutions Window.

The primary window with which the user interacts while analyzing a problem (Figure 13) displays a graph of the current normalized solution and a table of the minimum, good, normal, bad and maximum values for that

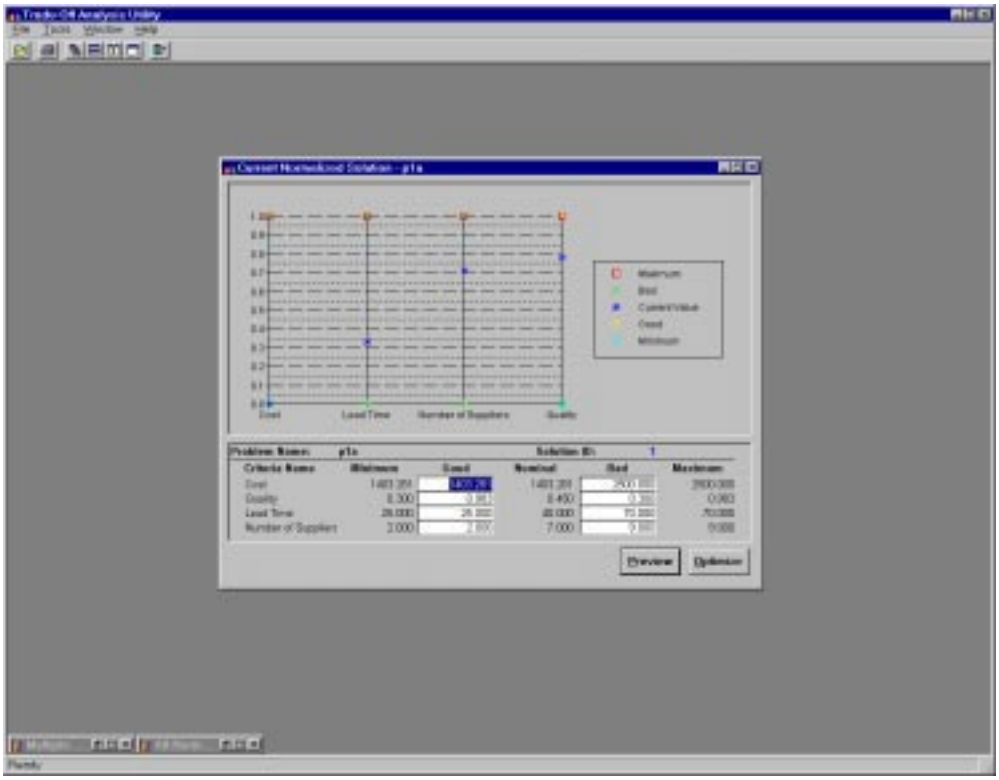

Figure 13 - Current Normalized Solution Window

solution. This window is referred to as the Current Normalized Solution window. The user can modify the good and bad values associated with any or all the criteria, preview these changes relative to the current normalized

solution, and re-solve the problem within this new context. This new solution is then graphed automatically in this and two other windows.

The Current Normalized Solution window is the central focus of both the software tool and this thesis. From the tool's perspective, it is the main control with which the user interacts. With respect to the thesis, it is where the user assigns the weights to be used in solving the problem within a new context. If this tool were constructed along the lines of a typical tradeoff analysis, the user would simply input arbitrary weights for each of the criteria. However, this window adds value to the problem solving process by allowing the user to tradeoff the acceptable and unacceptable levels of each of the criteria in an attempt to achieve what he/she considers the optimal solution. For example, by reducing the bad value (Figure 14), the user reduces the range and makes a particular solution less attractive. Thus, it is more likely that when the user re-solves the problem in this new context, a solution with an improved value for this criterion would result.

It is important to note that while the software tool uses the maximum and minimum values of a criterion for normalization, the optimization engine uses the good and bad values associated with the criterion to re-solve the problem within the new context. It was determined that normalization of each criterion with respect to its good and bad values would not facilitate comparison of different criteria for a particular solution or different solutions

for the same criterion. As a result of this determination, it was decided that for presentation purposes all normalization would be performed relative to the maximum and minimum values. Normalizing each criterion with respect to its maximum and minimum values not only allows the comparison of criteria with different units, but also allows different iterations of the optimization algorithm to be compared.

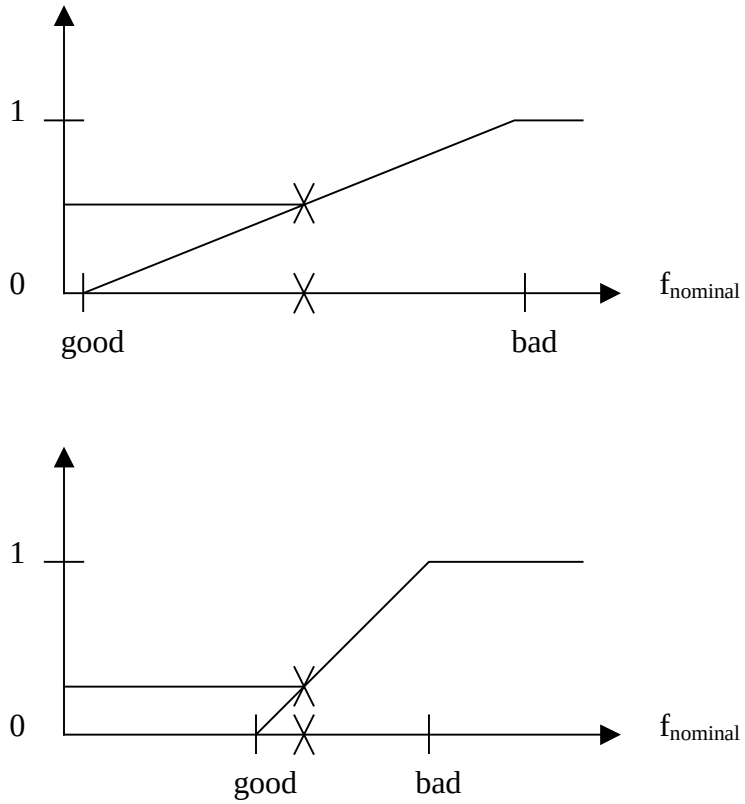

Figure 14 - Example of Previewing New Good and Bad Values

The Multiple Solutions window (Figure 15) displays a series of tabs. Each tab displays a single criterion and consists of a graph of the nominal

values of that criterion for all solutions. This window supplies the user with a graph of each criterion in its own units, which displays the nominal values of the criteria over all the iterations of the problem solving process. This window adds value to the user interface by showing the user each criterion in its own units, not as a normalized value. From these graphs, the user can gain an accurate understanding of the actual values yielded by successive iterations of optimizing the problem in different contexts.

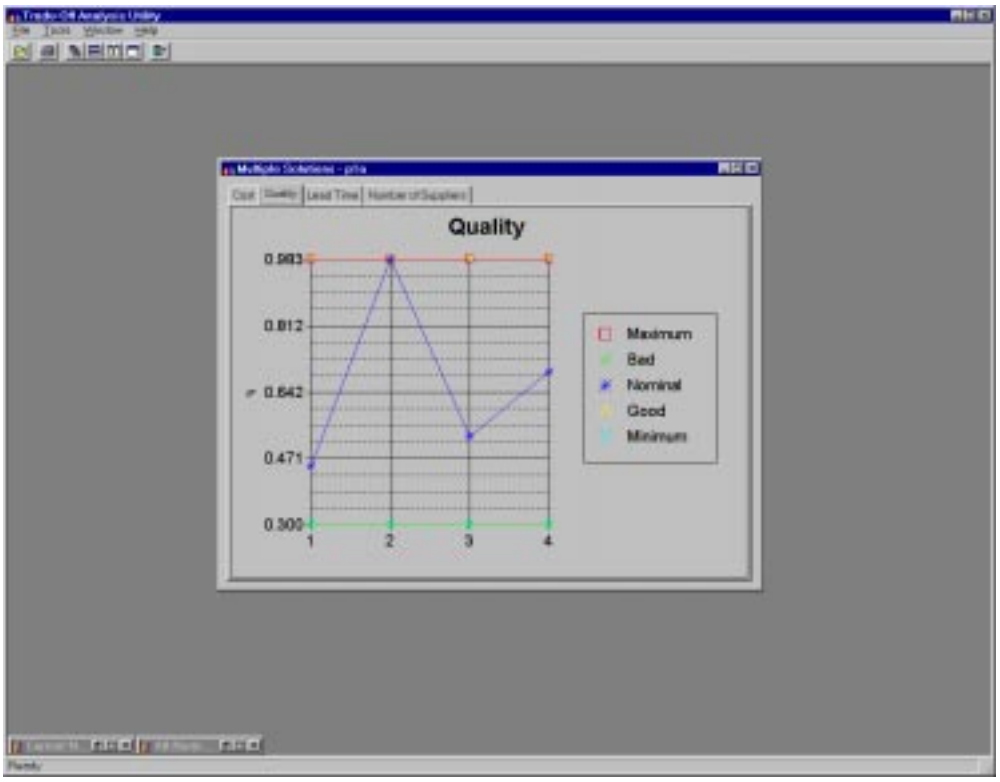

Figure 15 - Multiple Solutions Window

The user can exploit this graph to identify possible trends for the criterion over successive iterations or determine if the nominal value for a

specific solution is acceptable compared to the nominal value of another solution. For example, the problem that was solved to generate the figures in this thesis is defined by the criteria cost, quality, lead time, and number of suppliers. The graph of the nominal values of quality (Figure 15) indicate, as expected, that solution 2 was optimized for quality alone. Additionally, this graph also clearly displays the inverse relationship between cost and quality. The first solution was optimized for cost alone, and the nominal value for quality for that solution is the lowest on the graph.

The All Normalized Solutions window (Figure 16) displays a graph of all the solutions, which have been normalized with respect to each criteria's maximum and minimum values, and a table of the minimum, low, normal, high and maximum values for a solution selected by the user. This window is

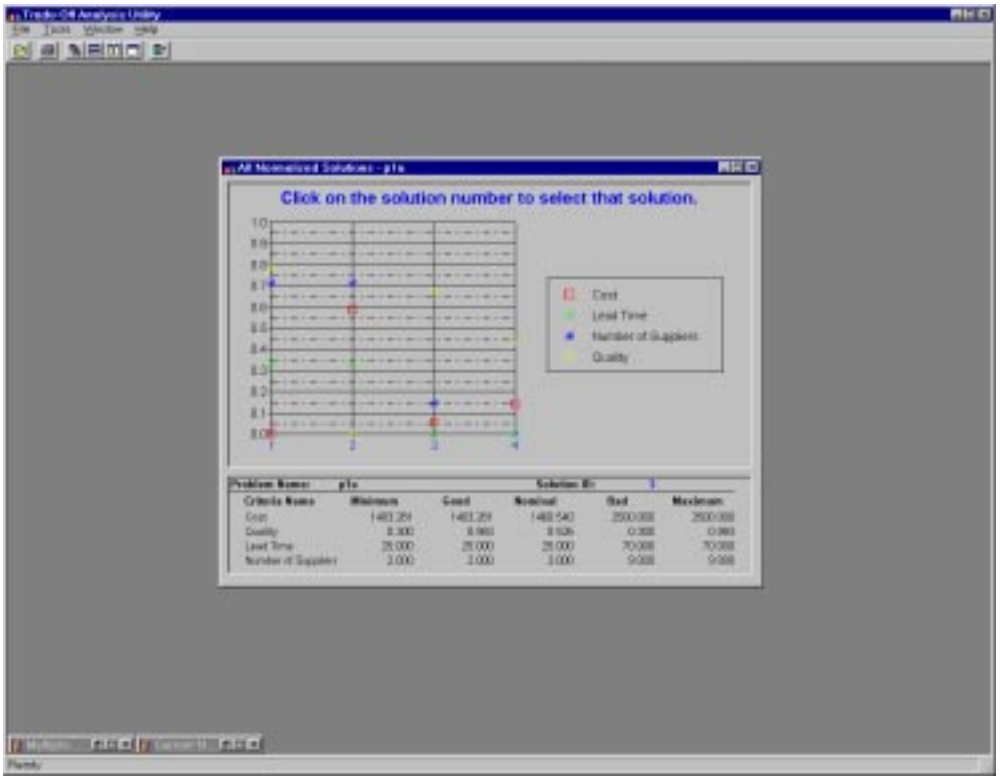

Figure 16 - All Normalized Solutions Window

similar to the Multiple Solutions window in that it displays the history of the user's optimization process. However, it displays the normalized values of each criterion for each iteration. The graph on this window is the same graph as is displayed in the Initial Normalized Solutions window (Figure 12).

This window allows the user to assess the impact of the changes made to the good and bad values for each criterion in the Current Normal Solution over the course of a problem solving session.

The Delete Solutions window (Figure 17) allows the user to delete individual solutions associated with an optimization session. This window allows the user to discard solutions that are deemed to be undesirable.

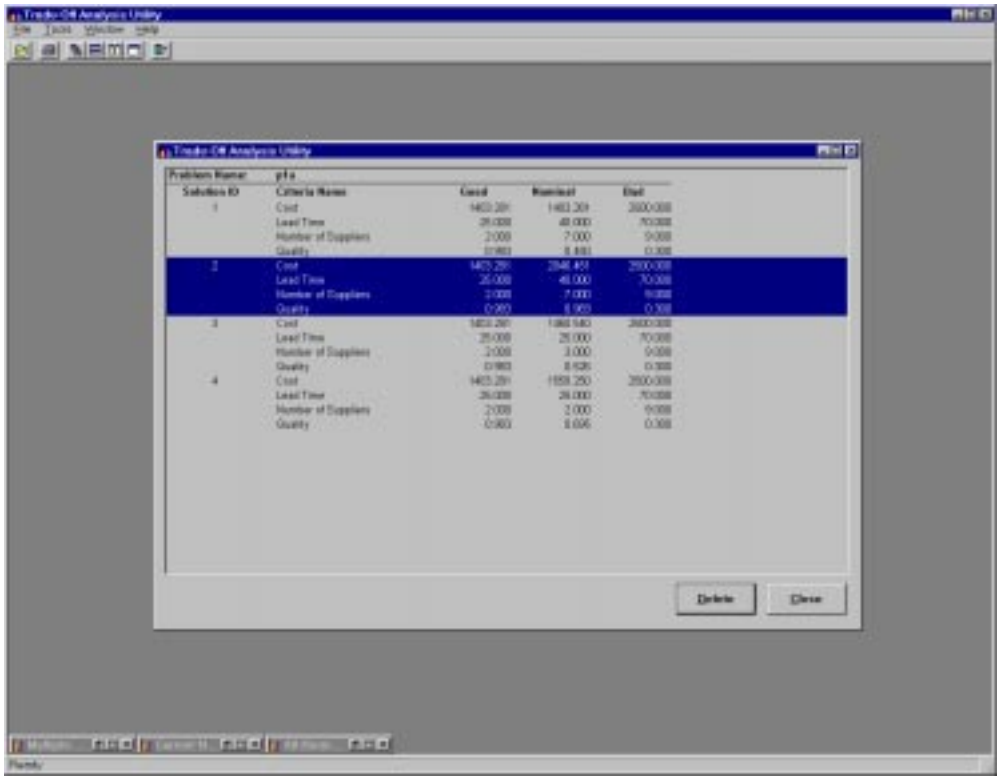

Figure 17 - Delete Solutions Window

The Print Solutions window allows the user to print a report that details the individual solutions associated with a particular problem (Figure 18). This window allows the user to obtain a hard copy printout of the results of a problem solving session.

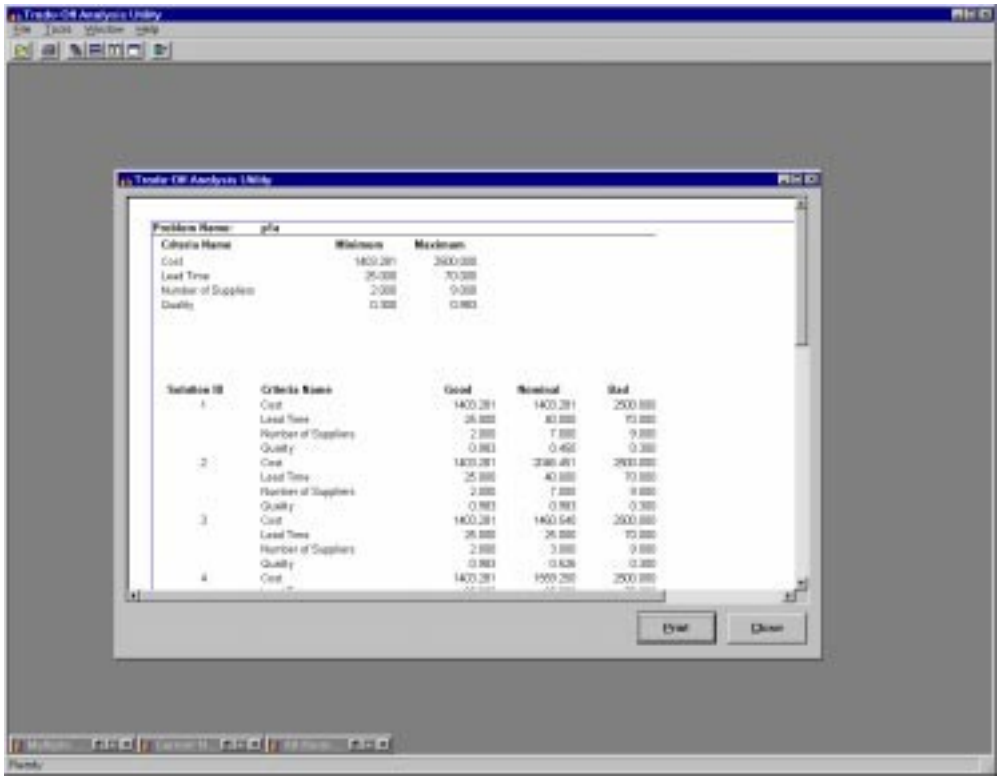

Figure 18 - Print Solutions Window

A synopsis of each of the windows' purposes is given below (Table 2).

Window	Purpose(s)
Initial Normalized Solutions	provide an educated starting point for an optimization session
	display the initial solutions for a new problem where each solution has been optimized for a single criterion
Current Normalized Solution	modify the good and bad values of the criteria to create a new context in which to optimize the problem
	preview the relative effects of creating a new context
	re-solve the problem in the current context
Multiple Solutions	graph nominal values for each criterion for all solutions
	provide a history of solutions for the problem
All Normalized Solutions	assess the impact of modifications to the context in which the problem is optimized
	provide a history of solutions for the problem
Delete Solutions	delete undesirable solutions
Print Solutions	print a report of an optimization session

Table 2 - Summary of Software Tool Windows and Their Purposes

C H A P T E R 5

SOFTWARE IMPLEMENTATION ISSUES

The final design operates around the concept of Problem Information files and Linear Programming files. Each file is named for the problem it defines, but ends with a different file extension. Problem Information files are named <problem name>.inf and Linear Programming files are named <problem name>.lp. A Problem Information file (Figure 5) stores the total number of criteria in the problem definition, the criteria name(s), the types of the criteria, i.e., maximizing or minimizing, and a bound for each criterion. A Linear Programming file contains the definition of the objective functions and the constraints to be solved for by the optimization engine.

Two database tables are maintained locally in a Sybase SQLAnywhere 5.0 database by the software tool. The first table stores the problem name, the criteria name(s) and the solution ID for each initial solution of the problem definition. The second table stores the problem name, criteria name, criteria type, solution ID, good, bad, maximum, minimum, nominal, normal, revised good and revised bad values for each criteria for each solution for a problem definition.

There are two main functions implemented in the DLL that are used to initialize and solve a problem. The software tool passes the name of the

problem to be initialized and the number of criteria associated with that problem to the initialize function. The initialize function then calls the optimization engine to solve the problem once for the original design optimized for cost, once for the original design optimized for quality, and once for each criterion alone. The nominal values of each criterion for each solution are then passed back from the function to the software tool. To solve the problem in a new context, the software tool passes the problem name, the number of criteria associated with the problem, the good and bad values associated with each criterion, and the identification number of the new solution to the solve function. A parameter specifying a time limit within which the solve function must return is also passed. If the optimal solution is not returned before this time limit expires, the best solution to that point is returned. The solve function can return one of three results to the software tool. First, it can return the new solution for the problem. Second, it can return the best solution available before the expiration of the time limit. Third, it can determine that based upon the new context, no feasible solution exists.

By default, the software tool sets the good and bad values for a new solution to the maximum and minimum based upon the type of the criterion as defined in the Problem Information File. For a minimizing criterion, both the maximum and initial bad value are set equal to the bound specified in the Problem Information File, whereas the minimum and good values are set to the

nominal value of the problem optimized for that criterion alone. For a maximizing criterion, both the minimum and initial bad value are set equal to the bound specified in the Problem Information File, whereas the maximum and good values are set to the nominal value of the problem optimized for that criterion alone.

Two Windows development tools were considered for use in creating and testing the graphical user interface for this software tool: Borland's Delphi 2.0 and Powersoft's PowerBuilder 5.0. Since the presentation style of much of the data in the user interface is graphical in nature, a development tool that allows the programmer to easily and efficiently build graphs was desired. Between the two, PowerBuilder possesses a unique control that automatically displays a graph based on a Standard Query Language (SQL) query. Delphi 2.0 requires the developer to produce the same result manually. This fact, coupled with a longer learning curve, lead to the selection of PowerBuilder 5.0 as the development tool of choice.

PowerBuilder 5.0 also offers another feature that was not leveraged until later in the development lifecycle, namely the Powersoft Foundation Class (PFC). The PFC is a service-based foundation class that allows a developer to inherit from pre-coded object classes, thereby avoiding coding functionality that already exists. Two examples of functionality leveraged from the PFC are the automatic resizing of controls within a window when the

window is resized by the user and the saving and restoring of the windows' states when the user starts the tool.

C H A P T E R 6

CONCLUSIONS

This chapter summarizes this thesis with respect to the user interface, software tool architecture, and future activities.

User Interface

The software tool developed as part of this thesis adds value to discrete multi-objective problem solving by offering the user a method of performing tradeoff analyses that are based on the normalized ranges of criteria. The user can change the acceptable and unacceptable, i.e., good and bad, values of a criterion in an attempt to tradeoff one criteria against another. For example, the user can accept a higher price for the PCB by increasing the bad value of cost, or he/she can select quality to be more important than other criteria by decreasing the difference between the good and bad values.

Aside from the direct manipulation of the problem solving process discussed above, the software tool's user interface provides the user with multiple methods of viewing the same information. The user can examine the latest solution to the problem in the given context, as well as compare successive iterations of the process to determine whether an optimal solution is being approached. Additionally, the user can view the nominal values of each

criteria in its own units, so as not to get too distracted by the unit-less measures displayed on the normalized graphs.

Software Tool Architecture

Aside from the main motivation of this thesis, providing a user interface for discrete optimization-based tradeoff analysis, there were two design goals for the software tool: modularity and database independence. Database independence was actually achieved as a by-product of the modularity of the final design. By dividing the software tool into separate components with well defined interfaces, this tool can be applied in any application domain where the interface to the product database and the optimization engine adhere to these interfaces.

Future Activities

The software tool created as part of this thesis has been tested using a few examples specific to PCB and MIC design. These problems are defined in terms of parts, alternative parts, processes, and alternative processes. When a solution is generated within a specific context of good and bad values for each criteria, alternative parts and alternative processes might be selected instead of the particular parts and processes designated by the PCB designer. A modification to the software tool suggested by a user is to graphically display

an initial solution for comparison purposes. In many applications, such a solution is available since the process might begin with the user defining a solution rather than defining an optimization problem.

The software tool keeps track of the output from the optimization engine in a database local to the tool. Additionally, the optimization engine also produces detailed results that are stored in files. The software tool could be enhanced in two ways as a result of these detailed result files. First, changes could be made to keep the local database and the detailed result files synchronized. If the user deletes a particular solution, both the database entry and the detailed result file could be erased. Second, the ability to select and view these detailed results files could be added to the tool, for the convenience of the user.

Finally, additional user requirements could be identified and implemented to enhance the reporting capabilities of the software tool. Although, the tool itself helps the user identify an optimal solution by iteratively solving a problem in different contexts, the report the user can print is rudimentary at best. It is important the user be able to end a problem solving session with a detailed report he/she deems valuable.

REFERENCES

Ball, Michael O., John S. Baras, Sridhar Bashyam, Ramesh K. Karne and Vinai S. Trichur. 1995. “On the Selection of Parts and Processes during Design of Printed Circuit Board Assemblies”. College Park, MD: University of Maryland at College Park, Institute for Systems Research.

Blanchard, Benjamin S., and Wolter J. Fabrycky. 1990. *Systems Engineering and Analysis*. 2d ed. Prentice Hall International Series in Industrial and Systems Engineering. Englewood Cliffs, NJ: Prentice Hall.

Cohon, Jared L. 1978. *Multiobjective Programming and Planning*. New York: Academic Press.

Fan, Michael K.H., Andre L. Tits, Jian L. Zhou, Li-Sheng Wang and Jan Koninckx. 1992. *CONSOL-OPTCAD User’s Manual*. College Park, MD: University of Maryland at College Park.

Galitz, Wilbert O. 1997. *The Essential Guide to User Interface Design: An introduction to GUI design principles and techniques*. New York: Wiley Computer Publishing, John Wiley & Sons, Inc.

Ghezzi, Carlo, Mehdi Jazayeri, and Dino Mandrioli. 1991. *Fundamentals of Software Engineering*. Upper Saddle River, NJ: Prentice Hall.

Martin, Alexander, and David Eastman. 1996. *The User Interface Design Book for the Applications Programmer*. Chichester, NY: John Wiley & Sons, Ltd.

Pressman, Roger S. 1992. *Software Engineering: A Practitioner’s Approach.* 3d ed. New York, NY: McGraw-Hill, Inc.

Ringuest, Jeffrey L. 1992. *Multiobjective Optimization: Behavioral and Computational Considerations*. Boston: Kluwer Academic Publishers.

Shoup, Terry E., and Farrokh Mistree. 1987. *Optimization Methods with Applications for Personal Computers.* Englewood Cliffs, NJ: Prentice-Hall.

Shneiderman, Ben. 1998. *Designing the User Interface: Strategies for Human-Computer Interaction*. 3d ed. Reading, MA: Addison-Wesley.

Sundaran, Rangarajan K. 1996. *A First Course in Optimization Theory*. Cambridge, NY: Cambridge University Press.

Trichur, Vinai S., and Michael O. Ball. 1998. "A Conceptual Framework For Product Design". College Park, MD: University of Maryland at College Park, Institute for Systems Research.

Trichur, Vinai S., Michael O. Ball, John S. Baras, Kiran Hebbar, Ioannis Minnis, Dana S. Nau and Stephen J.J. Smith. 1996. "Integrating Tradeoff Analysis and Plan-Based Evaluation of Designs for Microwave Modules". College Park, MD: University of Maryland at College Park, Institute for Systems Research.

Tufte, Edward R. 1983. *The Visual Display of Quantitative Information.* Cheshire, CT: Graphics Press.

_____. 1990. *Envisioning Information*. Cheshire, CT: Graphics Press.