Hierarchical Concurrent Engineering: Supporting Hierarchical Decomposition and Peer-to-Peer Problem Solving

Joseph G. D’Ambrosio & William P. Birmingham

Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI 48105
USA
wpb@eecs.umich.edu

Abstract

One primary task of engineering design is resolution of the conflicting objectives that are inherent in the design process. This problem is even more difficult when members of a design team, who each have different perspectives, must resolve these conflicts. We present a decision theoretic approach for resolving conflicting objectives during the design process, and discuss an extension to this approach that addresses coordination of hierarchical design organizations. Hierarchical design organizations arise from traditional hierarchical system decomposition, subcontracting, and supervisor/subordinate relationships. We believe the methods presented in this paper are general, and thus will provide benefits not only to concurrent engineering, but other domains as well.

1.0 Introduction

One primary task of engineering design is resolution of the conflicting objectives that are inherent in the design process. These conflicts arise due to the physical relations among objectives and resource limitations. Formal models of preferences (Thurston, 1991; Sykes and White, 1991) have shown promise in resolving conflicting design objectives. A formal model, e.g., a value (utility) function, in contrast to *ad hoc* techniques, provides a well-defined basis for predicting the quality of the results achieved. This property is highly desirable, since many engineering-design problems are so large that the desirability of the solution produced is difficult, if not impossible, to determine.

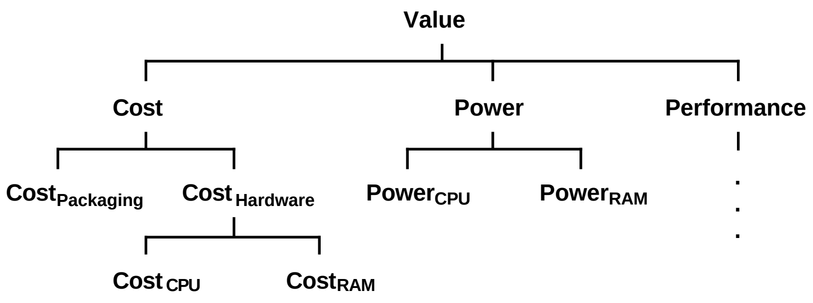

Value
Cost
Power
Performance
$\text{Cost}_{\text{Packaging}}$
$\text{Cost}_{\text{Hardware}}$
$\text{Power}_{\text{CPU}}$
$\text{Power}_{\text{RAM}}$
Cost_{CPU}
Cost_{RAM}

Figure 1.1: A portion of an objective/attribute hierarchy for a computer design problem.

To define a value function for a design problem, the attribute hierarchy of the problem must be identified. This hierarchy describes how the top-level attributes (objectives) of the problem relate to lower-level attributes, which are important for detailed design decisions. Figure 1.1 shows an example attribute hierarchy for a computer-design problem. The value of the design is determined based on three top-level attributes: cost, power, and performance. The top-level attribute cost is defined in terms of the cost of hardware and packaging used to construct the computer. With the attribute hierarchy defined, the top-level attributes of a design alternative can be calculated. A value function, which captures designer preferences, can be constructed by examining the top-level attribute values of all design alternatives. The best alternative can then be identified by applying the value function to evaluate the alternatives.

In general, it is not possible to generate and evaluate all design alternatives, due to problem-size restrictions. Some previous research has focused on hill climbing search of the design space, where in each iteration, the best alternative is chosen from all known alternatives. In this case, each iteration is considered a separate decision problem, and a new value function is created by examining all known alternatives. This process is time consuming and limits the amount of search that can be performed. What is needed is a technique for constructing a value function from incomplete knowledge of design alternatives, and then applying this value function during design optimization. In the section on current results, we describe our approach for solving this problem.

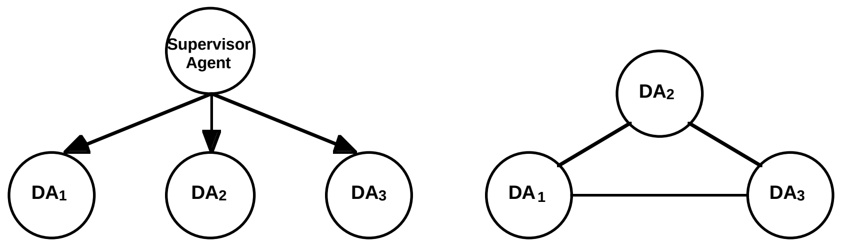
In addition to formally representing preferences, a mechanism is needed for combining and applying the preferences of the various decision makers involved in the design process. The growing practice of concurrent engineering (CE) has transformed the design process from solving a sequential series of independent design problems to a group decision process. Design decisions depend on inputs from a group, and can not be made from a single perspective. Members of a CE design team have different areas of expertise, and each decision maker is required to make decisions that impact others in the organization.

Currently, there exists two primary approaches for solving CE problems: multi-level optimization and agent-based systems. The multi-level optimization approach is based on mathematical programming techniques for decomposing and optimizing large systems. A single monolithic problem is decomposed into a number of subproblems, where each subproblem corresponds to a specific discipline involved in the design process. A coordination program assigns optimization parameters to each subproblem, subproblems are solved, and the coordination program analyzes the results and assigns new optimization parameters. This process is iterated until a convergence test is passed. Depending on the type of problem, the resulting solution is either locally or globally optimal.

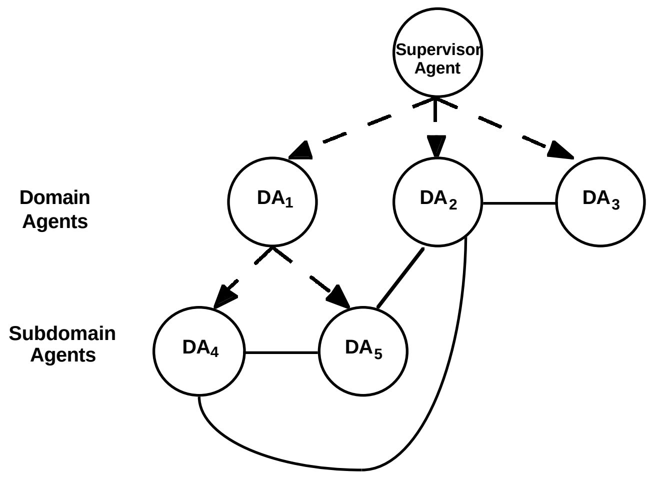
Agent-based systems for CE are based on distributed-artificial-intelligence techniques for distributed problem solving. An agent can be viewed as a computer program with the ability to independently choose what actions to take. The agent model is attractive since computer tools are used extensively in engineering design, and computer networks provide a means for communication and collaboration among the computer-based engineers. In general, an agent exists for each discipline involved in the CE problem, and it is assumed the knowledge representation varies from one discipline to another. The expertise required to solve the problem is distributed among the agents, and they must collaborate to find a good solution. A distributed problem-solving algorithm (e.g., distributed constraint satisfaction) coordinates the activities of the agents. Unlike the global optimization focus of multi-level optimization, agent-based systems focus on peer-to-peer problem solving.

We believe that the solution of a CE problem requires both global direction and exploitation of local expertise. Multi-level optimization approaches do not allow the local expertise of the participating disciplines to impact the value of the final design, since each discipline must solve the optimization problem specified by the global coordinator (Figure 1.2 (a)). In this approach, an *optimal* solution is obtained by restricting the definition of optimality to the preferences of the supervisor agent, and ignoring the preferences of domain agents. A domain agent is an expert only in the sense that it knows how to solve the subproblem specified by the supervisor agent; a domain agent has no ability to reflect its preferences into the design process. Without this ability, a domain agent is simply a demon that waits to dutifully respond to subproblem optimization requests.

In agent-based systems, peer-to-peer problem solving exploits local expertise (Figure 1.2 (b)), but ignores the global preferences that exist. Global preferences arise due to the standard, hierarchical system decomposition that is often required to solve large complex problems. To achieve desirable, overall system performance, the preferences of supervisors and system architects must be accounted for during the design of subsystems. In addition, portions of a design may be subcontracted to outside suppliers, and the preferences of the design organization must be accounted for by the subcontractor. This contractor/subcontractor relationship also occurs within an organization, where a designer reports to a supervisor, who's own objectives must be reflected in the design process. These agent relationships create a hierarchical agent organization, that must be accounted for during the design process.



(a) A two-level optimization organization. (b) A peer-to-peer agent organization.



(c) A hierarchical agent organization.

Figure 1.2: Agent Organizations for CE. Solid lines indicate problem solving networks, arrows indicate control relationships.

Figure 1.2 (c) illustrates our view of a typical CE problem. A supervisor agent contracts out a CE problem to three domain agents, DA1, DA2, and DA3. DA1 in turn subcontracts its responsibilities to two other agents, DA4 and DA5. Problem solving takes place through the interaction of agents DA2, DA3, DA4, and DA5, with coordination provided by the supervisor agent and DA1. The preferences of all agents are considered in the context of the existing hierarchical organization. Thus, a hierarchy of preferences is applied during problem solving.

In this paper we discuss a tool, ACME, which facilitates hierarchical concurrent engineering. ACME uses a formal model of preferences based on utility theory. We first present a review of previous work in the area of engineering design and concurrent engineering. Next we discribe our preference model and discuss results from the application of this model. We then describe how this model can be incorporated into an overall process for facilitating hierarchical concurrent engineering. We conclude with a summary and discussion.

2.0 Background

In this section we examine current approaches to multiattribute design and CE, as well as relevant background material. Our interest in CE is focused on the automation and optimization of multiattribute CE problems, thus the material reviewed will be limited to this area. The topics to be covered include the application of mathematical programming, distributed artificial intelligence, and decision theory to engineering design. There are other subject areas that are important to CE, such as collaboration technology and group dynamics, but these are not directly relevant to this paper, and thus are not discussed here.

We will first examine the formulation of multiattribute design problems. Next, multi-level optimization is introduced, and its application to CE is discussed. Finally, an overview of relevant topics from distributed artificial intelligence is given, and the application of these techniques to CE is discussed. Table 2.1 gives the notation developed in this paper.

2.1 Formulating a Design Problem

A mathematical-programming (Nemhauser, Kan, & Todd, 1989) or constraint-satisfaction (Mackworth, 1987; Mittal & Falkenhainer, 1990) formulation of a design problem includes:

- a tuple of n design variables, $X = (x_1, x_2, ...x_k, x_{k+1}, ...x_n)$, where variables 1 to k measure the k attributes to be optimized,
- a tuple of n domains, $D = (d_1, d_2, ..., d_n)$, such that $x_i \in d_i$,
- a set of constraints among design variables, $R = \{r_1, r_2, ..., r_m\}$, that restrict the domains of the variables,
- an objective function, $z(X)$, to evaluate the desirability of a design alternative.

Integer-valued design variables select the topology or components of the design, and real-valued design variables select design parameters. For topology decisions, a variable value of one indicates that a certain configuration of components should be used, while a zero value indicates that it should not. By setting up appropriate constraints between integer design variables, different topologies with different combinations of components can be considered. A wide range of constraints can be supported, including if-then constraints between integer design variables (Grossmann, 1990). In this paper, we focus on topology and component design decisions.

The majority of single-attribute design problems can be formulated and solved as mixed-integer non-linear programs (Grossmann, 1990) as follows:

$$\begin{aligned} &\text{Minimize } z(X) = x_1 \\ &\text{Subject to:} \\ &\quad R \\ &\quad x_1 ... x_i \in \Re \\ &\quad x_{i+1} ... x_n \in \{0,1\} \end{aligned}$$

where the non-linearities are present in the real variables, and x_1 measures the level[1] of a single attribute, such as cost. Mathematical programming for design has been applied to the fields of electronic engineering (Gebotys & Elmasry, 1993; Hafer & Parker, 1983; Lee, Hsu, & Lin, 1989), chemical engineering (Grossmann, 1990; Pekny, 1992), and mechanical engineering (Vanderplaats & Sugimoto, 1985). Constraint satisfaction approaches to engineering design include COSSACK (Mittal & Frayman, 1987), MBESDSD (Wu et al., 1990), and MICON (Birmingham, Gupta, & Siewiorek, 1992).

2.2 Multiattribute Optimization

A major difficulty in solving a design problem is resolving conflicting attributes. Improving the level of one attribute is usually only achieved at the expense of other attribute levels. Trade-offs must be made among the

[1] When discussing the value of an attribute, we use "level" to refer to the raw value of the attribute, and use "value" only to refer to the output of the attribute's value function.

attributes, but the exact information required to make such trade-offs is rarely available. One way to address such a problem is to generate the Pareto-optimal set of designs and select the alternative that best meets organizational goals.

$X,\ \Delta x_k$	Tuple of n design variables, $X = (x_1, x_2, \ldots x_k, x_{k+1}, \ldots x_n)$, where variables 1 to k measure the k attributes to be optimized; the difference in attribute k between two alternatives.
$D,\ \bar{d}_n$	Tuple of domains for design variables, such that $x_n \in d_n$; *a subset of* d_n, $\bar{d}_n \subseteq d_n$.
$R,\ r_k(X),\ r_{nk}(x_n)$	Set of constraints; constraint for attribute k, such that $x_k = r_k(X)$; constraint for x_n that calculates the contribution of x_n to attribute k.
$z(X)$	Objective function or search heuristic
S	Set of solutions.
$PX,\ PS$	A partition of X; set of partition solutions.
$w_k,\ W'$	Weight associated with attribute k; subspace of weight values.
$v_k(x_k),\ V,\ \Delta v_k$	Value of attribute k; set of attribute values, $V = \{v_1(x_1), v_2(x_2), \ldots v_k(x_k)\}$; difference in the value of attribute k between two alternatives.
$v(V),\ v(a_i),\ \Delta v$	Value of an alternative; value of alternative i; difference in the value between two alternatives.
$v_n(x_n),\ \Delta v_n,$	Contribution of x_n to $v(V)$; difference in contribution of x_n to $v(V)$ between two alternatives.
$v_{nk}(x_n),\ \Delta v_{nk}$	Contribution of x_n to $v_k(x_k)$; the difference in the value of attribute k for two alternatives due to x_n.
$m_k,\ m_k(x_k),\ b_k,\ b_k(x_k)$	Slope of a linear value function; slope at a given point; y intercept; y intercept projected using slope $m_k(x_k)$ from point corresponding to x_k.
$\bar{x}_{in},\ \bar{X}$	Assigned level of variable x_n for alternative i; tuple of assigned design variables, $\bar{X} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$.
$\Omega(f,\ x_i,\ x_1,\ \ldots,\ x_j)$	*Max or Min derivative of function* $f(x_i, x_1, \ldots, x_j)$ *with respect to* x_i.

Table 2.1: List of symbols.

Pareto optimality and multiattribute design evaluation

An alternative a_i is *Pareto preferred* to alternative a_j if all attribute levels of a_i are better than or equal to those of a_j, with at least one being strictly better. An alternative *dominates* another if it is preferred. The set of nondominated alternatives based on Pareto preference is the *Pareto-optimal set*. Identification of the Pareto-optimal set is also known as vector minimization (Murty, 1983). The optimal design is guaranteed to be in the Pareto-optimal set. The Pareto-optimal set is usually found by enumeration or by repeated optimization based on a single attribute while changing the bounds on others. GOPS (Haworth & Birmingham, 1993) produces the Pareto-optimal set of designs for electronic system problems using a combinatorial-optimization technique. Bradley and Agogino (1993) use Pareto plots to choose components from a catalog.

[2] When discussing the value of an attribute, we use "level" to refer to the raw value of the attribute, and use "value" only to refer to the output of the attribute's value function.

Decision-theoretic approaches, which provide accurate multiattribute evaluation, and thus solve the problem of specifying trade-off weights, have been applied with great success. Thurston (1991) demonstrates the application of multiattribute decision making for selecting the best design alternative. The utility of each design alternative is evaluated, and the one that maximize utility is chosen. Evaluation based on utility analysis requires detailed quantitative analysis, which may be difficult if a design problem has a high degree of imprecision, such as during the early stages of the design process. Wood *et al*. (1990) note the difficulties involved in using utility analysis when most design information is incomplete, and Wood & Antonsson (1989) describe an alternative approach based on fuzzy analysis, which allows evaluation based on qualitative information.

In theory, a mathematical-programming or constraint-satisfaction tool that identifies Pareto optimal set could be used for optimal design. Once the set is identified, utility or fuzzy analysis could be used to select the optimal alternative. Unfortunately, the Pareto-optimal set is often extremely large for interesting design problems (Murty, 1983), making generation of the entire set impractical. Artificial constraints can be added to reduce this set to a manageable size, but adding these constraints may eliminate the optimal solution from the feasible set.

Multiattribute evaluation during the design process

Instead of generating the entire Pareto-optimal set and then evaluating designs, multiattribute evaluation can be performed during the design process. Techniques for formulating multiattribute (multiobjective) mathematical programs include weighted evaluation function, goal programming, and compromise (Murty, 1983; Yu, 1989). A weighted evaluation function combines the level of attributes into a single objective function by using a weighted sum of attribute levels:

$$z(X) = \sum_k w_k \, x_k$$

The relative importance of one attribute *versus* another is determined by relative levels of the weights. For example, MICON and MBESDSD use a weighted evaluation function as a search heuristic, therefore reducing the multiattribute optimization problem to a single-objective-function problem. In goal programming, constraints assign goals for attribute levels, and slack and surplus variables measure deviation from these goals. The objective function, which is minimized, is a weighted summation of slack and surplus variables, with the relative levels of the weights specifying the relative importance of the goals. The compromise approach attempts to minimize the distance from an ideal state.

The compromise approach often fails, since there is no guarantee that the design alternative closest to the ideal will be optimal (Yu, 1989). The main limitation of a weighted evaluation function and goal programming is the difficulty in correlating weight levels to the solutions produced: *the process for assigning levels to weights does not guarantee the optimal design alternative will be identified*.

As an alternative, decision-theoretic techniques can be used in an iterative design algorithm (Sykes and White, 1991). Each iteration is considered a separate decision problem, and a search method, such as hill-climbing, is applied to identify a good design. These techniques do not guarantee optimality, and construction of an utility function for each iteration may require a significant amount of effort to support. In Section 3 of this paper, we describe a design technique, preference-directed optimal design, that performs decision-theoretic evaluation during design space exploration, and solves a large class of problems optimally.

2.3 Decomposition and Optimization of Large Systems

The first significant technique developed for decomposition and solution of mathematical programs was the Dantzig-Wolfe decomposition principle (Dantzig and Wolfe, 1960). This procedure is based on the observation that large systems are often comprised of one or more independent blocks linked by coupling equations. It performs a gradient search by iterating between the set of independent subproblems and a master program. Shared resources among subproblems are represented by constraints, and the master program sets a *price* for each, the subproblems are solved, and the master determines a new set of prices. Iteration proceeds until an optimality test is passed.

The two-level approach pioneered by Dantzig and Wolfe has been extended and applied by others to solve a wide range of problems. Mesarovic et al. (1970) specify a general theory for coordination of hierarchical systems. They discuss various coordination strategies based on known two-level mathematical-programming approaches. Wagner and Papalambros (1993a, 1993b) and Michelena and Papalambros (1994, 1995) specify decomposition strategies for a two-level mathematical-programming approach to design large systems. These decomposition techniques attempt to partition a large problem into subproblems such that the resulting two-level optimization problem can be efficiently solved.

The above two-level approaches are ideal for solving large, monolithic problems, but to apply them a single problem representation including a single multiattribute objective function must be identified. Seo and Sakawa (1982) and Tarvainen and Haimes (1981) describe approaches to eliminate the problems associated with specifying multiattribute objective functions by combining mathematical programming with decision making. For example, Tarvainen and Haimes use the Surrogate Worth Trade-Off (SWT) method (Haimes and Hall, 1974), which is a technique for analyzing and optimizing noncommensurate attributes with single or multiple decision makers, as part of a higher-level coordinator for hierarchical-multiobjective optimization (HMO). Although these approaches are an improvement over standard multi-level optimization, they are still focused on solving a single monolithic problem. In CE, multiple agents, each with their own perspective, interact to solve a problem, so it is not reasonable to consider such a problem as a single one.

2.4 Multi-Level Optimization for Concurrent Engineering

Haimes and others presents several extensions to HMO that attempt to address the needs of CE: hierarchical overlapping coordination (HOC) (Macko and Haimes, 1978; Haimes et al., 1990), and hierarchical holographic modeling (HHM) (Haimes, 1981; Haimes et al., 1990). Since alternative decompositions of a single HMO problem may be feasible and desirable, HOC coordinates multiple decompositions. In other situations, multiple models, with some shared variables and objectives, may be required, so HHM coordinates multiple HOC models with shared variables and objectives.

Barthelemy (1983) implements a multi-level optimization approach for CE based on a proposal by Sobieski (1982). Coordination over design activities in several disciplines is achieved by analyzing the sensitivities of subproblem objective functions and design variables to changes in coordination variables. Based on the sensitivities, new values of coordination variables are determined, and subproblems are resolved. Building on this work, Sobieski and others describe an approach to discipline-based decomposition (Bloebaum et al., 1992) and concurrent subspace optimization (CSSO) (Sobieski, 1990). Renaud and others extend this approach to including sequential global approximation (Renaud and Gabriele, 1993) and design variable sharing among disciplines (Wujek et al., 1995).

Azarm and Li (1988) propose an approach based on model coordination (Schoeffler, 1971) and global monotonicity concepts (Papalambros and Wilde, 1988). They discuss the application of sensitivity analysis to determine the impacts of parameter changes on global value (Li and Azarm, 1989).

The work of Sobieski and others is similar to that of Azarm and Li in that both focus on solving a discipline-based decomposition of a single problem. These approaches do not support the preferences of discipline agents. Although HHM provides a method for capturing and applying discipline preferences, it does not provide a framework for agent hierarchy: it only supports peer agents. This approach is similar to agent-based systems, which will be discussed next.

2.5 Distributed Artificial Intelligence

The formulation of a design problem, as previously described, can be solved by constraint satisfaction (Mackworth, 1987). An extension of this approach, distributed constraint satisfaction, allows a problem to be solved by a network of agents (Yokoo et al., 1990). Each agent is given a set of variables to assign, and these assignments must satisfy existing constraints among agents. Consistency and backtracking algorithms are two methods for solving these problems. Extensions to constraint satisfaction, which are useful in solving design problems (Darr and Birmingham, 1994), are interval propagation (Davis, 1987) and dynamic constraints (Mittal and Falkenhainer, 1990). Yokoo and Durfee (1991) investigate optimization aspects of distributed constraint networks from a group decision making perspective. Although this work does not address hierarchical organizations, it does identify several possible strategies for defining the optimal solution for peer-to-peer organizations, and describes a binary search algorithm for optimizing the solution of the problem.

A number of researchers have identified the need to organize networks of agents for problem solving. Fox (1981) studies results from management science to identify efficient agent organizations. Corkill and Lessor (1983) investigate organizational structuring to provide coordination among agents in a network. Ishida (1992) states that to achieve efficient agent organizations, methods are needed for decomposing and distributing goals, and for an agent to merge its goals with others. Durfee and Montgomery (Durfee and Montgomery, 1991; Montgomery and Durfee, 1993) discuss the benefits of forming hierarchical organization of agents based on shared behaviors. Although the mechanisms for forming and controlling hierarchical agent organizations, and associated benefits, identified by the above research are applicable in general to our work, we are more concerned with optimizing the performance of an existing organization where the desired behavior can be described by preferences and constraints, which allow efficient problem representation and solution. Our work is closer to that of Pan and Tenenbaum (1991), who discuss developing agent framework for enterprise integration. This latter work, however, does not address the need for multiattribute decision making.

In many agent-based system, an agent evaluates a payoff matrix to determine which action to take. As demonstrated by Horvitz (1988) and Dean and Boddy (1988), preferences (utility theory) can be used for evaluating alternatives. These specific approaches apply utility theory to metalevel decision making, accounting for not only the expected value of an alternative, but also the value of resources expended in making a decision. This work is focused on determining the actions of an individual agent, as opposed to a group of agents.

Ephrati and Rosenschein (1992) describe a hierarchical coordination approach referred to as non-absolute control. A subordinate agents attempts to achieve the goals assigned by a supervisor agent, but must also make use additional relevant information that is available only to the subordinate. The motivation for this research is to prevent a disaster, such as if an agent on Earth assigned a goal to a robot agent on Mars, and completion of that goal would result in the unforeseen destruction of the robot agent.

In our approach, contracting of preferences mandates that subordinates adhere to the preferences of supervisors, although subordinates are free to choose among alternatives that a supervisor is indifferent to. Since subordinates have access to supervisor's preferences, the subordinate is in a position to perform multiattribute evaluation, from a supervisor's point of view, of all of the subordinates alternative actions. If the problem is well characterized, a subordinate would never select an alternative that would be detrimental to the supervisor. In addition, we assume that solving additional iterations of a CE problem is practical, so that preferential knowledge gained during solution of a problem can be incorporated into a subsequent solution of the same problem.

2.6 Agent-based systems for Concurrent Engineering

Several agent-based systems focus on the needs of cooperative problem solving among domain experts. PACT (Cutkosky, et al., 1993) is a testbed for building large-scale, distributed CE systems, where agents are grouped by discipline and communicate through facilitators. First-Link (Park, et al., 1994) emphasizes collaboration among specialist and the development of hierarchical design representations. Agent communication and coordination mechanisms allow members of a design team to work concurrently at different levels of detail. DesignWorld (Huyn, et al., 1993) is an automated engineering environment for the design and manufacturing of digital circuits. In this system, a facilitator accepts an address-free message containing a task to be performed from an agent, and routes the task to the appropriate agent(s), possibly decomposing the tasks in the process. The current implementation allows communication of complete designs, part of a design, or constraints restricting the set of feasible alternatives The focus of the above systems is on knowledge representation and communication for a peer-to-peer problem solving network; there is no concept of hierarchical control or preferences in these systems.

Some agent-based system have attempted to address the need for hierarchical control and preferences. DFI (Werkman, 1992) is a CE tool for steel-connection design. A system designer hierarchically controls a network of agents, each of which is a domain expert with its own preferences. The preferences of the system designer are limited to specifying a single attribute to optimize, while an ad hoc scheme for representing agent preferences allows agents to negotiate over values for the remaining attributes. In ACDS (Darr and Birmingham, 1994), a system agent distributes its preferences to catalog agents, who uses these preferences to select components. In this system, there is no ability for catalog agents to apply their own expertise (preferences). AGENTS (Huang and Brandon, 1993) is a object-oriented Prolog-based language for cooperating expert systems. To implement concurrent engineering, a system designer interacts with a directorate design agent, who controls two agent committees, comprised of design agents and analysis agents respectively. There is no concept of preferences in AGENTS. Although these systems attempt to address the needs for hierarchical control and preferences, they lack a uniform approach, and thus are limited in their capabilities.

As has been described, neither multi-level optimization nor agent-based systems provides a general framework for hierarchical preferences. What is needed is an approach that provides global coordination, similar to multi-level optimization, and exploits local expertise, similar to agent-based systems. In the next section, we present our approach to representing designer preferences, and in Section 4, we discuss how this representation can be applied in solving hierarchical CE problems.

3.0 Preference-Directed Design

We now introduce an approach to engineering design that emphasizes applications of preferences during the design process (D'Ambrosio and Birmingham, 1995). The approach blends previous research in the areas of utility theory, constraint networks and constraint propagation, and combinatorial optimization. To solve the problems associated with defining and applying preferences during the design process, we have chosen a formal model of preferences based on utility theory and have developed a process for applying this model to solve problems of reasonable size. The model is referred to as an *imprecise value function,* and is created by specifying preferences among a subset of design alternatives. This function specifies a partial order over all design alternatives. Thus, from incomplete knowledge of all design alternatives, a value function is created that captures a significant portion of a designer's preferences.

To solve large problems, we rely on problem-solving techniques such as partitioning, thereby creating an additional problem related to identifying the value of a subset of a design. Partitioning reduces the overall complexity of a problem by decomposing it into a number of subsets that are independent and can be solved in isolation. In our approach, an imprecise value function defines a preference structure that ranks design alternatives; however, if partitioning of design variables is performed, imprecise value functions for each variable must be derived from the global imprecise value function. These design-variable value functions are required if preferential decisions for partition alternatives are to be made. We identify design-variable value functions from the objective/attribute hierarchy, which provides the ability to relate the impact of design decisions (*e.g*., the selection of a CPU) to the overall value of the design. *The key contribution of this research is the theory for deriving the value function for a subset of a design from the value function of the entire design.*

In additional to that theory, we have developed two preference-directed design algorithms for applying the model and theory during the design process. The first algorithm, *Preference-Directed Optimal Design*, identifies a set of optimal solutions for problems of small to moderate size. When compared to a similar algorithm based on identifying the Pareto-optimal set, our test results indicate that our algorithm extends the class of problems that can be solved optimally. The second algorithm, *Preference-Directed Evolution*, finds good solutions to problems of any size. By providing a means to solve problems regardless of complexity, we believe we have demonstrated the flexibility of our preference model and theory.

The main contributions of the research presented in this section are:

- Development of theory and heuristics for identifying the value of a design variable with respect to the value of the entire design, thus increasing the ability to applying preferences *during* the design process.
- Demonstration of how this capability extends the set of designs that can be solved optimally.

- Demonstration that our preference model is flexible, in that it can be used in conjunction with different optimization algorithms, thus providing solutions to a wide range of design problems

3.1 Modeling Preferences

Many problems are characterized by multiple conflicting attributes, and in many cases a multiple-attribute value function can represent the preference structure of the problem. Two condition must be met for a multi-attribute value function to exist. The first condition is a monotonicity condition, which states if the value of one attribute improves while there is no loss in value for other attributes, preference must increase. The second is a continuity condition, which states if $a_i \succeq a_j \succeq a_k$, then there must be a unique point where the decision maker is indifferent between the increase from a_k to a_j and the increase from a_j to a_i.

One drawback of representing preferences by a multiattribute value function is the amount of work required to construct the function, which is due to the multi-dimensionality of the problem. In many cases, the amount of work required can be reduced by decomposing the value function into subsets of attributes that are independent of the others. Krantz et al. (1971) show that if each attribute is *preferentially independent* of its complement, the value function can be decomposed such that $v = v(v_1, v_2, ... v_k)$, where v_k is the attribute value function for attribute k. An attribute k is preferentially independent if the weak order specified by v_k is independent of the level of other attributes. A physical relationship may exist between preferentially independent attributes (*i.e.*, a constraint $r \in R$); it is only required that the preference order for attribute levels be independent of the others. In the case of all attributes being preferentially independent, attribute value functions can be constructed independently of other attributes, resulting in a significant reduction in the work required to construct the overall function.

The most desirable form of a decomposable value function is an additive value function. An additive value function is a weighted sum of attribute values, such that the value of an alternative with k attributes is given by:

$$v(V) = \sum_k w_k v_k(x_k) \qquad \textbf{(3.1)}$$

where w_k is the tradeoff weight for attribute k, and $v_k(x_k)$ is the value produced by the value function of attribute k. Keeney and Raiffa (1976) describe the process for determining the attribute value functions and tradeoff weights. This form of the multi-attribute value function requires that the attributes be *mutually, preferentially independent*. A set of attributes is mutually, preferentially independent if every subset of these attributes is preferentially independent of its complementary set. It can also be shown that in general if each pair of attributes is preferentially independent of its complement, then Eqn. 3.1 is valid (Gorman, 1968). Unlike standard value functions, the additive value function is unique up to a positive linear transform.

Imprecise Value Functions

In addition to decomposing a multi-attribute value function, the amount of quantitative analysis required to construct the function can be reduced by using an imprecise value function. Imprecisely Specified Multi-Attribute Utility Theory (ISMAUT) (White et al., 1984) creates a partial order based on preference relationships among a subset of alternatives. ISMAUT uses a weighted sum of attribute values. Thus, the value of alternative a_i is given by Eqn. 3.1. A preference statement of the form "a_i is preferred to a_j" implies an inequality in the space of possible weights according to the following relation:

$$\sum_k w_k \left[v_{ik}(x_k) - v_{jk}(x_k) \right] \geq 0$$

According to this interpretation, the statement that a_i is preferred to a_j means that the trade-off weights are such that the total weighted value of a_i is at least as great as that of a_j. Direct pair-wise preferences among attributes also translate to inequalities, such as:

$$w_{power} \geq w_{dollars}$$

Furthermore, all weights must be positive and their sum must be unity:

$$\forall k,\ w_k \geq 0$$

$$\sum_k w_k = 1$$

All these inequalities confine the weight space to a subspace, *W*', that satisfies the inequalities. Thus, from pair-wise preference statements, ISMAUT determines ranges of attribute weights consistent with designer's preferences.

The imprecise value function *v*(*V*) can order pairs other than those specified by the designer: a_i is preferred to a_j if, for every possible vector of weights $<w_1, w_2, ..., w_k>$ within *W*', the value of a_i is greater than the value of a_j, *i.e.*,

$$\text{Min} \sum_k w_k \left[v_{ik}(x_k) - v_{jk}(x_k) \right] \geq 0, \qquad w_k \in W'$$

This relation can be tested for every pair of alternatives that the designer has not already stated a preference. Thus, the preferences specified by a designer create a partial order over *all* design alternatives, and this partial order can identify the nondominated set of design alternatives.

ISMAUT provides the means to partially order a set of alternatives using a small set of preference statements, and thus identify a set of nondominated alternatives guaranteed to contain the optimal one. *This partial order is stronger than Pareto preference*, in that there will be fewer alternatives in the final nondominated set. In fact, without any preference statements, the nondominated set produced by ISMAUT will be the Pareto-optimal set.

ISMAUT Tools

We have developed a software package, ISMAUT Tools, to support decision making based on imprecise value functions. ISMAUT Tools provides a library of routines for defining attributes, alternatives, and preferences, and for performing dominance checks between alternatives. In addition to the library, a text-based interface has been developed. The text-based interface allows ISMAUT Tools to be used in isolation as a decision support tool, and the library of routines provides other applications the ability to perform dominance checks based on an imprecise value function.

Our experience with ISMAUT Tools indicates that decision making based on an imprecise value function is both practical and desirable. Although each dominance check requires the solution of a linear program, our test cases indicate that selection problems with over 1000 alternatives can be solved in less than a minute when executed on a standard Unix workstation. In one test case, nine pair-wise preference statements allowed a Pareto-optimal set containing over 350 alternatives to be reduced to a nondominated set containing five alternatives in less than one minute. Reducing the set of alternatives from 350 to five allows a decision maker to focus analysis efforts, improving the ability to identify the best alternative.

3.2 Optimization Techniques

Given a preference structure modeled by an imprecise value function, there are several ways to apply preferences to select the best alternative. The most straight forward approach is to generate the entire set of feasible alternatives, and use the ISMAUT Tools text-based interface to evaluate pairs of alternatives with the imprecise value function to identify a nondominated set of alternatives. Although only a small set of preferences statements are required to construct the imprecise value function, *generation* of all alternatives and performing the necessary dominance checks are not practical for many problems. To solve this problem, we propose to use the imprecise value function to prune inferior paths during design-space search (D'Ambrosio and Birmingham, 1995). Many design alternatives can be eliminated from consideration before they are fully constructed, thus reducing the complexity of the design process.

In this section, we describe two main concepts related to our approach, unconstrained variables and isolating design-variable value functions.

Unconstrained Variables

Constraint-graph analysis is the process of examining the feasibility and preferential dependencies between design variables. A graph describing the dependencies can be constructed where the nodes are design variables and the arcs are constraints. Problem decomposition can be achieved by partitioning the graph such that nodes connected by arcs are placed in the same partition. The constraint graph given in Figure 3.1 can be into three subsets of design variables, $\{x_1, x_2, x_3\}$, $\{x_4, x_5\}$, and $\{x_6\}$. Design decisions related to feasibility for a subset can be made independently from the assignment of variables in other sets, since there is no constraint connecting them. The third subset, $\{x_6\}$, is a special case, because it contains only one variable. The variable in this subset, x_6, is *unconstrained*, since the feasibility of the assignment does not depend on the assignments of any other variables. Thus, dominance checks between possible assignments of an unconstrained design variable can be performed, which, if successful, reduce the size of the design space by eliminating inferior alternatives.

Dynamic constraint-graph analysis improves problem decomposition during search. For the graph given in Figure 3.1, the assignment of variables x_2 and x_3 is constrained via constraints r_1 and r_2, respectively, by the assignment of x_1. If the current state of the search process is such that x_1 has been assigned, then the feasible domains for x_2 and x_3 cannot be impacted by any additional assignments. Domain-consistency checks can be performed between $x_1$and x_2, and between x_1 and x_3 to identify the assignments of x_2 and x_3 that are consistent with the assignment of x_1. With their domains limited, x_2 and x_3 become unconstrained variables, and dominance checks can be performed. Similar to the methods proposed by Dechter and Pearl (1987), dynamic constraint-graph analysis can be used to order the assignments of variables during search such that the number of unconstrained variables encountered along a search path is maximized.

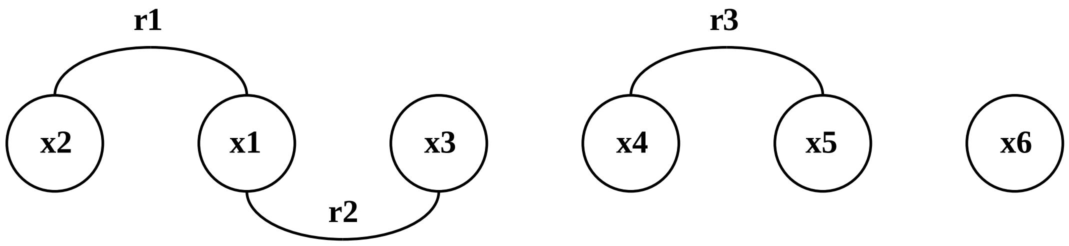


Figure 3.1: Constraint graph with variables $x_1,...,x_6$ and constraints r_1,r_2,r_3

Isolating Design-Variable Value Functions

An imprecise value function, created by specifying preferences over competing designs, is only valid for comparing complete designs, but evaluation of a subset of a designs is often required. Partitioning a problem dictates that the value of prospective solutions for a partition be compared. For example, in solving a partition, the best partition alternative must be found. To identify this alternative, a value function reflecting the value of the partition to the entire design is required. To obtain a value function for a partition, the global value function must be mathematically decomposed[3] into values of the design variables contained in the partition. Similarly, when evaluating alternative assignments for an unconstrained design variable, the value of the variable to the entire design must be the basis for comparison. To support this capability, ISMAUT must be extended to allow comparisons among alternatives for subsets of a design. We now describe an approach for comparing assignments of unconstrained variables.

An additive attribute k is defined as an attribute with a level given by the summation of attribute levels corresponding to design variables:

$$x_k = r_k(X) = \sum_n r_{nk}(x_n)$$

One interpretation of the above equation is that if $r_{nk}(x_n) \neq 0$, x_n is a subsystem of the design, and in isolation can be meaningfully characterized by attribute k, e.g. cost. A linear attribute value function for attribute k can be expressed in slope-intercept form:

$$v_k(x_k) \;=\; m_k\; x_k + b_k$$

Corollary 1[4]: If for a given problem:

- *a variable x_n is unconstrained*
- *attributes are additive and mutually preferentially independent*
- *attribute value functions are linear*

Then the assignment of variable x_n is independent of the assignment of all other design variables, and the optimal alternative will have an assignment for x_n such that $v_n(x_n)$ is maximized, where

$$v_n(x_n) = \sum_k w_k\, v_{nk}(x_n)$$

and

$$v_{nk}(x_n) \;=\; m_k\; r_{nk}(x_n) + \frac{b_k}{N}$$

When the conditions of Corollary 1 are satisfied, the change in total design value, Δv, with respect to assignments $\bar{x}_{in}$ and $\bar{x}_{jn}$ is equal to the change in design variable value, Δv_n, thus $\bar{x}_{in} \succeq \bar{x}_{jn}$ if Min $\Delta v_n \geq 0$.

Theorem 2: *If for a given problem:*

- *a set of variables is unconstrained*
- *attributes are additive and mutually preferentially independent*
- *attribute value functions are non-linear*

Then the optimal assignment to variable x_n is dependent on the assignment of other design variables.

Corollary 1 states that under restrictive conditions, the value of a design variable can be found independently of the values of other design variables. Theorem 2 states that if the restrictions on attribute value functions are relaxed, then the value of a variable cannot be conveniently isolated from the assignments made to other variables. This situation

[3] Decomposing a value function into values of design variables should not be confused with the standard value-analysis decomposition of attributes into preferentially independent sets.

[4] Development and proofs of Corollary 1 and Theorem 2 is given by D'Ambrosio and Birmingham (1995).

is complicated by the fact that multiple assignments may still be possible for other individual variables, so the calculation of a design variable value depends on the range of attribute and variable assignments still possible.

To perform checks of this latter type, Δv_n can be estimated or bounded by a form of sensitivity analysis. If the difference, Δx_n, is small, then the following first-order model, or other higher-order models, closely estimates Δv_n,

$$\Delta v_n \approx \frac{\partial v(\overline{X}_j)}{\partial x_n} \Delta x_n$$

In our applications, x_n is an discrete variable representing selection of a part or a topology, therefore the change in x_n cannot be assumed to be small. In addition, traditional ISMAUT dominance checks only eliminate an alternative if it is guaranteed to be dominated, so the above estimation approach may not be desirable in that it represents a departure from this philosophy.

To remedy this situation, we propose the following check that conservatively bounds Δv_n, guaranteeing that only dominated alternatives will be eliminated. Recall that the ISMAUT dominance check for a variable determines the change in total design value with respect to the change between the worst possible value of one assignment compared to the best possible value of another. If the change is positive, $\Delta v_n \geq 0$, and the first assignment dominates the second. In the following calculation, the minimum or maximum value of $\partial v / \partial x_n$ bounds Δv_n,

$$\Delta v_n \geq \begin{cases} \Omega_{Min}(v, x_n)\Delta x_n & \text{if } \Delta x_n \geq 0 \\ \Omega_{Max}(v, x_n)\Delta x_n & \text{if } \Delta x_n < 0 \end{cases}$$

where,

$$\Omega_{Min}(v,x_n) = \quad \text{Minimize} \quad \frac{\partial v(X)}{\partial x_n} = \sum_k w_k \frac{\partial v_k(r_k(X))}{\partial x_n}$$

$$\text{Subject to:} \quad x_i \in \overline{d}_i, \quad w_k \in W'$$

$\overline{d}_i$ is current range of possible assignments for variable x, and $\Omega_{Max}(v,x_n)$ is similarly defined. Thus if $\Omega(v,x_n)\ \Delta x_n \geq 0$, then $\overline{x}_{in} \underline{\succ} \overline{x}_{jn}$.

Calculation of $\Omega(v,x_n)\ \Delta x_n$ is not performed directly, since this expression can be decomposed into subexpressions that are easier to determine. First recall that the value assigned to x_n represents a choice among a finite set of elements, e.g., the selection of a part. When evaluating the difference between two assignments, Δx_n only indirectly takes part in the calculation, since the actual calculation is based on attribute changes, $\Delta r_{nk}(x_n)$, that are induced by Δx_n. Thus an equivalent expression with the $\Delta r_{nk}(x_n)$ as parameters is evaluated to determine $\Omega(v,x_n)\ \Delta x_n$,

$$\Omega(v,x_n)\ \Delta x_n = \text{Minimize} \sum_k w_k\, \Omega(v_k, r_{nk})\Delta r_{nk}(x_n), \qquad w \in W'$$

In addition, the calculation of $\Omega(v_k,r_{nk})\ \Delta r_{n\,k}(x_n)$ can be simplified depending on the type of attributes and attribute value functions. Consider the following chain rule expansion[6] of $\Omega(v_k,r_{nk})\ \Delta r_{n\,k}(x_n)$,

$$\Omega\left(v_k, r_{nk}\right)\Delta r_{nk}\left(x_n\right) = \text{Minimize} \quad \frac{\partial v_k}{\partial r_k}\frac{\partial r_k}{\partial r_{nk}}\Delta r_{n\,k}\left(x_n\right), \qquad x_i \in d_i$$

If a design problem is characterized by additive attributes, then $\partial r_k / \partial r_{nk} = 1$, and if the problem also has linear attribute value functions, then $\partial v_k / \partial r_k = m_k$, where m_k is the slope of $v_k(r_k)$. These conditions are identical to those specified in Corollary 1, as is the following linear program for Δv_n

$$\Delta v_n \geq \Omega\left(v, r_{nk}\right)\Delta r_{nk}\left(x_n\right) = \text{Minimize} \sum_k w_k\, m_k\, \Delta r_{n\,k}, \qquad w \in W'$$

The above linear program has the desirable property that no information on design variable assignments, other than for x_n, is required to solve the program.

With nonlinear attribute value functions, $\partial v_k / \partial r_k$ is not a fixed number, and depends on the value of r_k. If $\partial v_k / \partial r_k$ is first bounded by $\Omega(v_k,r_k)$, then a problem with additive attributes and nonlinear attribute value functions can be solved by the following linear program,

$$\Delta v_n \geq \Omega\left(v, r_{nk}\right)\Delta r_{nk}\left(x_n\right) = \text{Minimize} \quad \sum_k w_k\, \Omega\left(v_k, r_k\right)\ \Delta r_{n\,k}(x_n), \qquad \mathrm{w} \in \mathrm{W}'$$

Once $\Omega(v_k,r_k)$ is solved, the above linear program does not require any information on design variable assignment, other than for x_n. Furthermore, $\Omega(v_k,r_k)$ is easily solved, since construction of $v_k(r_k(X))$ by the method of specifying indifference points results in $v_k(r_k(X))$ being piece-wise linear, allowing quick determination of the maximum and minimum derivatives over some range of X.

Returning to linear attribute value functions, relaxing the constraint on additive attribute requires that $\partial r_k / \partial r_{nk}$ be bounded by $\Omega(r_k,r_{nk})$, since $\partial r_k / \partial r_{nk}$ depends on the levels of $r_{ik}(x_i)$ for each design variable x_i that contributes to $r_k(X)$. If $\partial r_k / \partial r_{nk}$ is bounded, the following linear program solves problems with nonadditive attributes and linear attribute value functions,

$$\Delta v_n \geq \Omega\left(v, r_{nk}\right)\Delta r_{nk}\left(x_n\right) = \text{Minimize} \quad \sum_k w_k\, m_k\, \Omega\left(r_k, r_{nk}\right)\Delta r_{nk}\left(x_n\right) \qquad w \in W'$$

Finally, removing preferential-independence restrictions increases the complexity related to defining value functions and performing dominance checks. If attributes are not preferential independent, complete decomposition of the problem is not possible. Subsets of attributes, that are independent of their complement, can be identified. Value functions for these subsets can be determined, and the overall form of the global value function identified. For this situation, assessment of value functions and dominance relations will require more effort. These calculations are similar to nonlinear attribute value functions, but they are harder to solve.

[6] This expansion is only valid if $r_k = f(r_{1k}(x_1), \ldots., r_{nk}(x_n))$, where the only function in f that depends on x_n is r_{nk}. If $r_k = f(r_{1k}(x_1),\ldots,r_{n\text{-}1\,k}\,(x_n), \ldots r_{n\text{-}z\,k}\,(x_n))$, where $r_{n\text{-}1\,k}\,(x_n)$ to $r_{n\text{-}z\,k}\,(x_n)$ depend on x_n, then a similar, but slightly more complex expansion must be applied.

Table 3.2 summarizes various forms of Δv_n depending on the type of attributes and attribute value functions. In most problems, the attributes will not all be of the same type, and Δv_n is a combination of the forms given Table 3.2.

As an example, consider a network of three resistors characterized by two attributes, resistance and cost. Resistors x_1 and x_2 are connected in parallel, and this parallel combination is connected in series with resistor x_3. The total resistance of the network is given by

$$R = \frac{r_1\, r_2}{r_1 + r_2} + r_3$$

while the total cost of the network is the just the summation of individual resistor costs. Assume that both resistance and cost have linear value functions. Total cost is an additive attribute, but total resistance is not.

Attributes	**Attr. Value Functions**	Δv_n	**Depends on**	
			Attr. Levels	**Design Vars.**
Additive	Linear	$\sum_k w_k\, m_k\, \Delta r_{nk}(x_n)$	No	No
Additive	Non-linear	$\sum_k w_k\, \Omega(v_k, r_k)\Delta r_{nk}(x_n)$	Yes	No
Non-additive	Linear	$\sum_k w_k\, m_k\, \Omega(r_k, r_{nk})\Delta r_{nk}(x_n)$	No	Yes
Non-additive	Non-linear	$\sum_k w_k\, \Omega(v_k, r_k)\Omega(r_k, r_{nk})\Delta r_{nk}(x_n)$	Yes	Yes

Table 3.2: Properties of Δv_n for mutually preferentially independent attributes.

The value of x_1 to the design is given by the following value function,

$$\Delta v_1 \geq \text{Min}\ \frac{\partial v(X)}{\partial x_1}\Delta x_1 = w_c\, m_c\, \Delta c_1(x_1) + w_r\, m_r\, \Omega\big(R, r_1, r_{2,}\big)\, \Delta r_1(x_1), \quad w \in W'$$

where,

$$\Omega(R, r_1, r_2) = \text{Min/Max}\ \frac{\partial R(X)}{\partial r_1} = \text{Min/Max}\left(\frac{r_2^2}{r_1^2 + 2 r_1 r_2 + r_2^2}\right) \quad x_i \in d_i$$

and c_1 is the cost of x_1 and m_c and m_r are attribute-value-function slopes.

To find the nondominated set of assignments for x_1, the interval of nondominated resistance levels possible for x_2 is needed, since the impact of x_1 on the total resistance changes depending on the resistance of x_2. The resistance value of x_3 is not needed, since its impact on the total resistance is additive. Cost values for x_2 and x_3 are not needed, since cost is an additive attribute.

In summary, unlike techniques that decompose value functions so that function creation is simplified, we have presented an approach that decomposes value functions to improve efficiency of the problem solver. For a given design variable, our approach identifies a subset of attributes and variables that must be considered when performing dominance checks. Depending on the type of problem to be solved, varying degrees of decomposition can be achieved. We now present several test cases to illustrate our technique and demonstrate its benefits.

3.3 Optimization Algorithms

We now overview two algorithms for solving configuration design problems. The first algorithm, *preference-directed optimal design*, provides optimal solutions for small to moderate sized design problems. The second algorithm, *preference-directed evolution*, provides good solutions for large design problems. Both algorithms rely on an imprecise value function for evaluating the desirability of a design.

Preference-directed optimal design

Preference-directed optimal design (D'Ambrosio and Birmingham, 1995) facilitates multi-attribute decision making while minimizing the enumeration and evaluation of design alternatives. An imprecise value function is created by ranking a random sample of design alternatives. Preference-directed search, which decomposes the value function for decision making, uses combinatorial optimization and constraint propagation to explore the design space and identify a set of nondominated solutions.

```
PD_OptDesign(in: $X$, $R$, $D$; out: $S$) {
    GenerateRandSample(in: $X$, $R$, $D$; out: $S$);
    RankSample(in: $S$; out: $V$);
    BuildConstraintNetwork(in: $X$, $R$, $D$);
    StaticPartitioning(in: $X$, $R$; out: $PX$);
    for each $px_i \in PX$ {
        PD_Search(in: $px_i$, $v$; in/out: $ps_i$, $D$);
    CombinePartitions(in: $PS$, out: S);
}
```

Figure 3.2: Preference-directed optimal design algorithm

Figure 3.2 describes the preference-directed optimal design algorithm. Domain knowledge defines design variables (X), variable domains (D), and constraints (R). By making random variations in an objective function, GenerateRandSample creates a random sample of feasible designs, S, where each design is characterized by a set of non-commensurate attributes. RankSample is an interactive procedure for specifying attribute value functions and ranking the elements in the sample. The sample is either totally or partially ordered from pair-wise comparisons between elements in the set. The attribute value functions and preferences among sample elements imply an imprecise value function, v, which measures design value during problem solving. BuildConstraintNetwork generates a constraint network for constraint propagation during search (design) (Mackworth, 1987) . StaticPartitioning examines the constraint connections among design variables, and divides X into sets, P_1, P_2,...,P_i, forming independent sub problems, if possible (see unconstrained variables, Section 3.2). Preference-directed search, PD_Search, finds solutions for each partition by performing a branch-and-bound search of the design space, and limits the complexity of the search by performing constraint satisfaction and pruning domains of unconstrained variables. CombinePartitions integrates partition solutions into the final set of solutions.

Preference-directed evolution

Certain design problems are so large as to precluded the use of an optimal algorithm. In this case the optimization problem becomes one of finding a good solution as opposed to an optimal one. For this classes of problems, we have developed an algorithm, preference-directed evolution, based on evolutionary strategies. We have chosen this approach because of the ease of formulating a configuration design problem as an evolutionary problem, and because of the desirable optimization properties of evolutionary strategies.

An evolutionary algorithm executes for a user specified number of generation, evolving a population of solutions. For a given population size N, the algorithm starts by randomly generating N alternative solutions, any of which may not be feasible. The population is copied to create a set N' of alternatives, and each member in N' is mutated. The two sets, N and N', are then combined, and a fitness functions selects the best N alternatives from the combined set. This process is then repeated, evolving the population with each iteration. An evolutionary algorithm is not trapped by local optimums, since each member of the population represents a solution in a different part of the solution space.

To solve a design problem using an evolutionary strategy, the problem must be mapped into the characteristics of a member and into the fitness function. For preference-directed evolution, a member of the population is defined by a vector of independent design variables. When the algorithm starts, each variable in the vector is randomly assigned to a level in its respective domain. During mutation, at least one design variable is assigned to a different level in its domain. Instead of using a standard weighted evaluation function as the fitness function, constraints and the imprecise value function determine the fitness of member. The population is divided into two classes, feasible and infeasible, and members in a class are sorted based on dominance checks.

3.4 Test Cases

In remaining portion of this section we present a summary of two test cases. Both test case where solved using the preference-directed optimal design algorithm, and one was also solved using the preference-directed evolution algorithm. We performed these test cases to gain some insight on the performance and flexibility of our approach. A summary of the two cases is given in Table 3.3.

	Computer Board	**HWSW Codesign**
Level of Difficulty	Moderate	Moderate
Attributes	Additive	Non-additive
Attr. Value Functions.	Linear	Linear
P-D Design Results	Excellent	Undetermined
P-D Evolution Results	-	Good

Table 3.3: Evaluation of imprecise value functions for four test cases

Computer Board Design

To evaluate the performance of an imprecise value function against generation of the Pareto-optimal set, a suite of test cases based on a computer design problem was developed. The function specification for each test case is identical, but the number of parts (domain size) available to implement the functions ranges from 29 to 463. The imprecise value function was created from linear attribute value functions and ranking a random sample of 10 feasible alternatives. The same problem solver was used for all test cases; the only difference between runs is how evaluations were performed (Pareto preference or imprecise value function). The results are given in Table 3.4.

One important finding from this suite of tests is that for problems of any reasonable size, solving the problem with an imprecise value function requires less time to find a set of nondominated solutions. Each dominance check based on designer's preferences requires the execution of a linear program to determine Min Δv. For trivial problems, the extra time required to solve the linear programs may result in the solver executing longer than one based on Pareto preference. For problems of any significance, the extra pruning capability provided by preference-directed optimal design dramatically reduces execution time.

	J29	J63	J77	J90	J109	J194	J465
# Feasible Alternatives	10^3	10^{11}	10^{12}	10^{14}	10^{16}	10^{21}	10^{28}
Pareto Preference							
# Solutions	28	379	1698	3495	NA	NA	NA
Time (sec.)	0.7	23.4	310.6	1303.4	NA	NA	NA
Imprecise Value Function							
# Solutions	2	2	4	4	4	24	28
Time (sec.)	1.2	2.8	4.4	5.1	5.5	39.3	55.2

Table 3.4: Comparison of dominance checks based on Pareto preference and Preference-Directed Design.

A second finding is that the set of nondominated solutions is drastically reduced by dominance checks based on designer's preferences. For the test case J90, Pareto preference produces 3495 nondominated solutions, while designer's preferences produces only 4. The problem solver could not solve any of the Pareto-preference cases with more than 90 parts, since the memory required to maintain all of the solutions was too large. To solve this problem, attribute bounds would have to be introduced, thus potentially eliminating the optimal solution from the Pareto-optimal set.

Hardware/Software Codesign

To add further evidence of the benefits of preference-directed design over approaches based on Pareto-optimality, we investigated the design of a real-time embedded controller (D'Ambrosio et al., 1993; Hu et al., 1994; D'Ambrosio and Hu, 1994). In this type of a system, interactions between hardware and software must be accounted for. The function specification for this problem included nine software functions to be supported, and required that a computer board with enough RAM, ROM, and CPU throughput be configured. Only branch-and-bound dominance checks were enabled for this test case; the nature of the problem's constraints required that the implementation of the algorithm be extended if constraint propagation and unconstrained variable domain pruning were to be enabled, which we have not accomplished at this time.

The initial results of this test case appeared to be disappointing, in that a Pareto-optimal approach appeared to out-perform preference-directed design. Further investigation showed that the source of the poor performance was due to branch and bound in general, and not the imprecise value function. Disabling the imprecise value function, and performing Pareto-optimal branch-and-bound checks required more time than full generation of the Pareto-optimal set. We have concluded that the performance of branch-and-bound checks can be improved by adding simple heuristics to guide the assignment of design variable, such that the variables which determine most of the level of global attributes are assigned first, as opposed to last.

In addition to the above test, we also solved this problem using our evolutionary algorithm. Our motivation was to compare the results of the evolutionary algorithm to the optimal solution, which was found by the preference-directed design algorithm. The initial population for the evolutionary algorithm is chosen randomly, so by executing the algorithm several times, typical results can be identified. Running the algorithm for fifty generations typically identified most of the nondominated designs found by the optimal algorithm. This result leads us to believe that the evolutionary algorithm will produce good results for larger problems, which cannot be solved optimally.

4.0 ACDS in a Multilevel Environment (ACME)

The theory and algorithms developed in our previous work here focused on resolving conflicting objectives from the viewpoint of a single designer, but in reality, a designer rarely works in isolation. We have chosen to extend preference-directed design to address CE problems. We now describe how the preference model and theory developed for preference-directed design can be combined with an existing CE tool, ACDS, to create ACME, which coordinates a set of agents in a hierarchical organization.

Many decision-making problems are solved by subcontracting or delegating parts of a task. The process of subcontracting, which has already been shown to be viable approach to subtask distribution (Davis and Smith, 1983), creates a hierarchical, decision-making organization. Coordination is accomplished by allowing general contractors, who have a global perspective, to provide direction to subcontractors, who have local expertise.

We believe that it is practical and desirable to direct a subcontractor's actions by explicitly stating a preference structure that the agent must follow. Just as any contract describes a set of constraints that must be met for the contract to be fulfilled, a contract should include a preference structure that must be followed by a subcontractor in order to fulfill its obligations. Oddly enough, our experience with a major automotive company, which we believe to be typical of large engineering companies, shows that *such preferences are not explicitly or implicitly specified.* Specifying a preference structure for a subcontractor does not require the general contractor to reveal all of its preferential knowledge to the subcontractor, only the subset of preferences that rank the implementation domain of the contracted subtask need be specified.

One difficulty in specifying preferences for a subtask is deriving the preference structure for the subtask from the overall preference structure of the general contractor. Given a task to accomplish, a general contractor defines and contracts out a set of subtasks, that taken together, accomplish the task. The preference structure of the general contractor describes the desirability of possible implementation of the task, but to contract out subtasks, the preference structure for each subtask must be derived from the preference structure for the task.

A process similar to isolating variable value functions in preference-directed optimal design can be used to determine the value of a contracted subtask. To accomplish this, an agent value-function/attribute hierarchy is created, expressing global objectives in terms of attributes meaningful to subcontracting agents. Each agent encodes its preferences over the attributes meaningful to it in an imprecise value function. The agent value-function/attribute hierarchy allows the imprecise value function of a general contracting agent to be translated into attributes meaningful for a subcontractor, and the preferences of the two agents are merged appropriately.

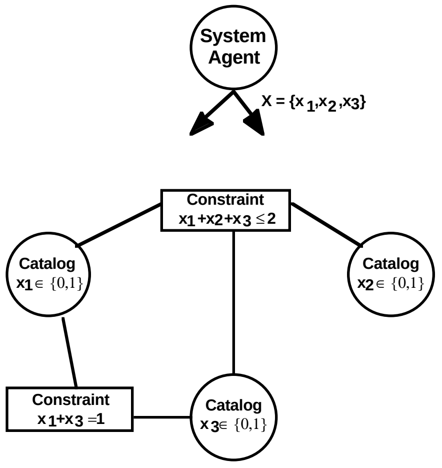
We believe this approach possesses a number of features that are desirable for solving CE problems:

- Formal preference model provides accurate representation of individual preferences.
- Construction of a useful agent preference structure from a subset of design alternatives.
- Ability to provide preferences and constraints to subcontractors.
- Ability of subcontractors to evaluate the effects of local decisions on global value.
- Support for hierarchical agent organizations.
- Problem solving using a variety of agents, each with different objectives.

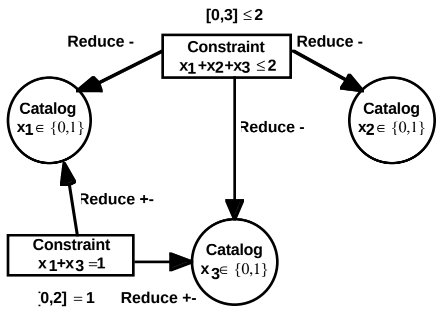
4.1 ACDS Overview

The foundation of ACME is based on an existing CE tool, ACDS, which solves a class of distributed, dynamic, interval constraint-satisfaction problems (DDICSP). We chose ACDS as the foundation for two reasons. First, a constraint-satisfaction formulation is a good match to our approach. Second, we have access to the source code of ACDS, and thus the effort required to construct ACME is minimized.

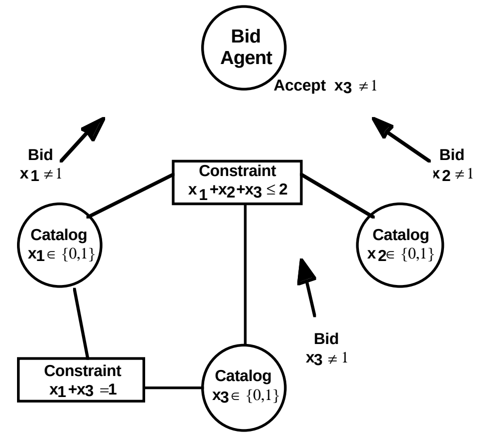
The ACDS system defines a distributed problem-solving algorithm for a network of agents that finds solutions to catalog-based design problems. A network is comprised of a system agent, a bid agent, catalog agents, and constraint agents. The system agent specifies the problem to be solved. Each catalog agent has the responsibility of finding a set of parts (assignments) for a design variable that satisfy constraints. Constraint agents calculate attribute levels and monitor constraint conditions. The bid agent evaluates proposals from catalog agents for moving the network closer to *decomposability,* meaning that all combinations of values from all domains satisfy constraints.



(a) Creating a network.



(b) Detecting constraint violations.



(c) Bidding on assignments to eliminate.

Figure 4.1: Solving an ACDS problem.

Solving a problem proceeds as follows. The system agent specifies to the catalog agents the variables to be assigned, and a catalog agent is selected for each design variable. Constraint agents are then created by the system agent or catalog agents (see Figure 4.1 (a)). The constraint agents check the current range of possible assignments for each catalog, and report to the catalog agent *node- and arc-consistency* violations. The catalogs then prune their domains to achieve node- and arc- consistency. The constraint agents then check to see if any of the attribute intervals violate constraints. If so, the constraint agents instruct the catalog agents to bid a set of assignments to eliminate (see Figure 4.1 (b)), with the goal that once these assignments are eliminated, the degree to which constraints are violated will be reduced. The bid agent evaluates these bids, selecting a set of bids such that progress in reducing each violation is made (see Figure 4.1 (c)). Catalogs whose bids are accepted remove the specified assignments (i.e., parts), and the constraint agents once again check for possible constraint violations. This process is repeated until no constraint violations remain. At this point, a shared value function, specified by the system agent, is applied to select the best assignment in each catalog.

Extending the ACDS algorithm to solve a network of agents with subcontracting ability requires that the concept of a catalog agent and system agent be merged. In terms of the ACDS algorithm, catalog agents gain the ability to specify preferences and to subcontract design variables. In ACDS, this behavior is only possible in the system agent. This eliminates the need for a formal distinction between the system and catalog agents, so we refer to both types of agents simply as design agents. This change only affects the problem-specification phase: the ACDS algorithm for identifying a decomposable network is unchanged.

In addition to modifying the problem-specification phase, the problem-solving phase must be modified to apply preferences during problem solving. The two phases of problem solving that are affected are: achieving a decomposable network and solving a decomposable network. To achieve a decomposable network, coordination methods similar to CSSO (Sobieski, 1990) can be applied. These sensitivity techniques are a good match to our process for calculating Δv_n. Solving a decomposable network has many similarities to the optimal design algorithm given in Section 3, and as a result, we have already defined an algorithm for solving this part of the problem.

We will now describe the operation of ACME by means of an overview of the preference-related aspects of formulating and solving an example problem.

4.2 Example Problem

For purposes of illustration, we will describe our approach using a CE problem focused on hardware-software codesign. The example presented here is a simplified version of the problem described by D'Ambrosio et. al. (1993) and Hu et al. (1994). A system agent wishes to create the embedded controller shown in Figure 4.2. The controller is comprised of both hardware and software, and both must be considered simultaneously. The function specification for this problem includes three software functions to be supported, and requires that a computer board with enough RAM and CPU throughput be configured. Two attributes, cost and feasibility factor, characterize the system. Cost is a summation of hardware component costs, and feasibility factor (D'Ambrosio and Hu, 1994) is a measure of the system's ability to guarantee that all software modules complete execution before their respective deadlines.

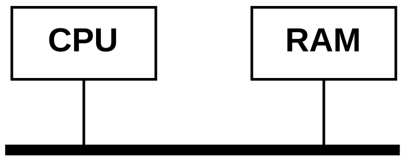
The system agent formulates the problem as shown in Figure 4.3. Design variables x_{CPU} and x_{RAM} are defined for the two hardware modules, as are $x_{ReadSensor}$, x_{Filter}, and $x_{FuelCalc}$ for the three software modules. These five variables are the independent variables for the problem. The attributes of interest, cost and feasibility factor, are represented by two dependent variables, x_{cost} and x_{FF}, respectively. The calculation of these two variables is specified by two constraints, r_{cost} and r_{FF}. The r_{cost} constraint is expressed in terms of functions of independent variables, while the r_{FF} constraint is expressed in terms of both a function of an independent variable, $r_{CPU\,tr}$, and intermediate dependent variables, $x_{tr\text{-}l}$ and $x_{tr\text{-}u}$. In addition to constraints for calculating attribute values, two constraints, r_5 and r_6, determine the feasibility of a solution. r_5 ensures enough RAM exist to meet the needs of the software modules, and r_6 ensures all software modules execute before their deadlines.

To design the embedded controller, the system agent must contract out the design work among the two organizations shown in Figure 4.4. In both organizations, a supervisor is responsible for assigning tasks. The organization headed by the hardware agent is capable of configuring computer boards with CPU, RAM, and hardware I/O. The organization headed by the software agent selects the appropriate software modules.

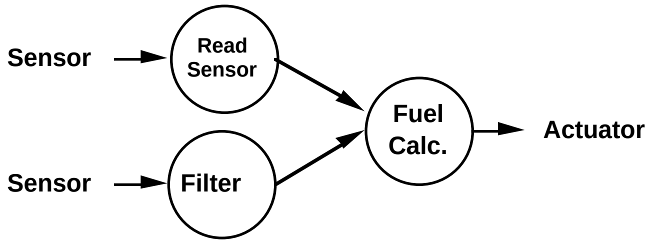
4.3 Establishing a Network of Design Agents

The system agent assigns responsibility for each independent design variable to a design agent and each dependent design variable to a *constraint agent*. Assignment of variables to design agents is performed using an approach similar to the contract-net negotiation process (Davis and Smith, 1983). When a design agent makes an offer for a design variable, it commits to participate in the design process in exchange for the possibility that its assignment for the design variable will be used. In general, a given variable can be assigned to multiple agents, with the agents competing during the design process, and a single agent may bid for multiple design variables. Assignment of variables to constraint agents is discussed in the next section.

Figure 4.5 illustrates the assignment of variables to design agents. The system agent request bids for the following variables: x_{Filter}, $x_{ReadSensor}$, $x_{FuelCalc}$, x_{CPU}, x_{RAM}. The hardware agents bids for x_{CPU}, $x_{RAM,}$ and the software agent bids for x_{Filter}, $x_{ReadSensor}$, $x_{FuelCalc}$. Lower-level agents do not make bids at this point, since they can only bid on offers from their respective supervisors. The system agent accepts the bids from the two agents, and in this case, the two agents subcontract all design variables to their subordinates.



(a) System hardware



(b) System software

Figure 4.2: Embedded controller example.

$X = \{x_{cost}, x_{FF}, x_{tr\text{-}l}, x_{tr\text{-}u}, x_{Filter}, x_{ReadSensor}, x_{FuelCalc}, x_{CPU}, x_{RAM}\}$

$R = \{$

$r_{cost} = r_{CPU\ cost}(x_{CPU}) + r_{RAM\ cost}(x_{RAM}),$

$r_{FF} = f(r_{CPU\ tr}\ (x_{CPU}), r_{tr\text{-}l}, r_{tr\text{-}u}),$

$r_{tr\text{-}l} = f(r_{Filter\ dl}(x_{Filter}), r_{Filter\ per}(x_{Filter}), r_{Filter\ act}(x_{Filter}), r_{ReadSensor\ dl}(x_{ReadSensor}), r_{ReadSensor\ per}(x_{ReadSensor}), r_{ReadSensor\ act}(x_{ReadSensor}), r_{FuelCalc\ dl}(x_{FuelCalc}), r_{FuelCalc\ per}(x_{FuelCalc}), r_{FuelCalc\ act}(x_{FuelCalc})),$

$r_{tr\text{-}u} = f(r_{Filter\ dl}(x_{Filter}), r_{Filter\ per}(x_{Filter}), r_{Filter\ act}(x_{Filter}), r_{ReadSensor\ dl}(x_{ReadSensor}), r_{ReadSensor\ per}(x_{ReadSensor}), r_{ReadSensor\ act}(x_{ReadSensor}), r_{FuelCalc\ dl}(x_{FuelCalc}), r_{FuelCalc\ per}(x_{FuelCalc}), r_{FuelCalc\ act}(x_{FuelCalc})),$

$r_5: r_{CPU\ RAM}(x_{CPU}) + r_{RAM\ RAM}(x_{RAM}) \geq r_{Filter\ RAM}(x_{Filter}) + r_{ReadSensor\ RAM}(x_{ReadSensor}) + r_{FuelCalc\ RAM}(x_{FuelCalc})$

$r_6: r_{FF} \geq 0.0$

$\}$

Figure 4.3: Problem formulation.

4.4 Establishing Constraint Agents

Figure 4.6 shows the constraint agents created for the example problem. In some cases, such as the example problem, all constraint agents are created by the general contractor (system agent), however, this is not a requirement. The set of agents connected by constraints forms the problem-solving network. Other agents, such as the system agent, only indirectly participate in problem solving through the contracting of preferences to active participants in the network.

4.5 Distributing Preferences

With constraints among agents established, the agent value-function/attribute hierarchy can be derived (see Figure 4.7), and agents can proceed to specify preferences. First a random sample of designs, A, is generated and distributed to all agents. The only restriction on the elements in A is that they be feasible. A feasible solution is an assignment of a level to each variable in X such that each constraint in R is satisfied. The process for finding a feasible alternative is identical to solving for the most preferred alternative, except that any set of consistent preferences may be used.

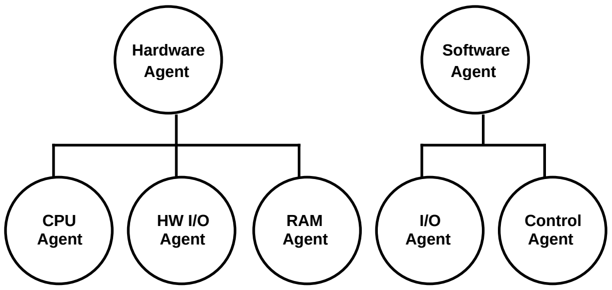


Figure 4.4: Two groups of design agents available to implement the embedded system.

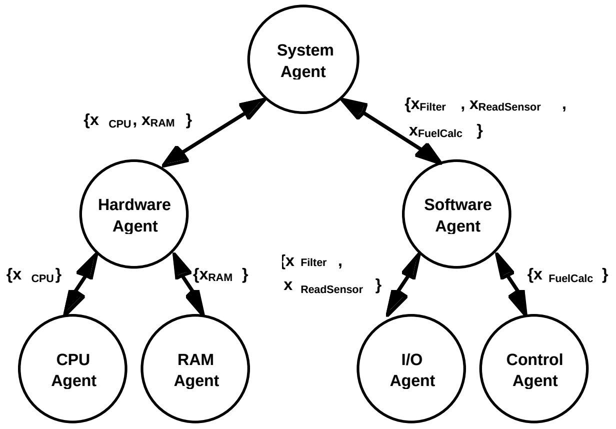


Figure 4.5: Forming a design organization.

Next, the agents hierarchically define preferences by ranking elements in the sample and defining attribute value functions (see Figure 4.8 (a)). The system agent specifies a set of two element tuples, A_{sys}, ranking the alternatives in the sample, and also specifies attribute value functions for each attribute that concerns the agent. A ranking of elements in A is a partial order represented by a set of two element tuples, where the first element in each tuple is an alternative from A that is at least as preferred as the second element in the tuple. This process allows an agent the ability to specify preferences over the attributes that are important to the agent. In the example, the system agent specifies preferences over the cost and feasibility factor attributes, x_{cost} and x_{FF}, the hardware agent over cost and CPU throughput, x_{cost} and $r_{CPU\,tr}$, and the CPU agent over CPU cost and CPU throughput, $r_{CPU\,cost}$ and $r_{CPU\,tr}$. In addition to the constraints specified by the system agent, these preference become part of the problem specification, and therefore, must be followed during the design process.

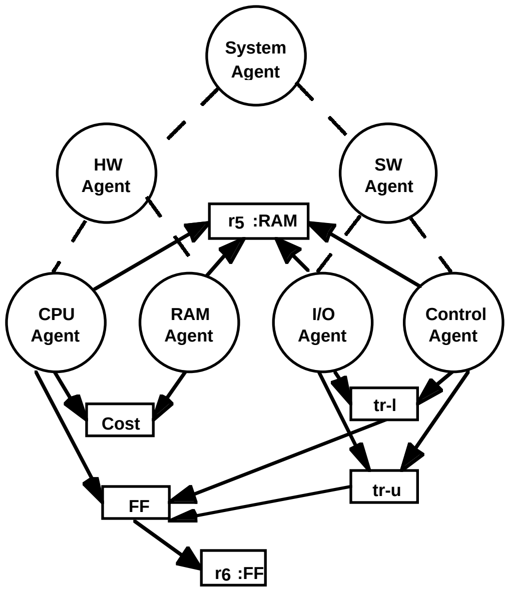


Figure 4.6: Creating constraint agents.

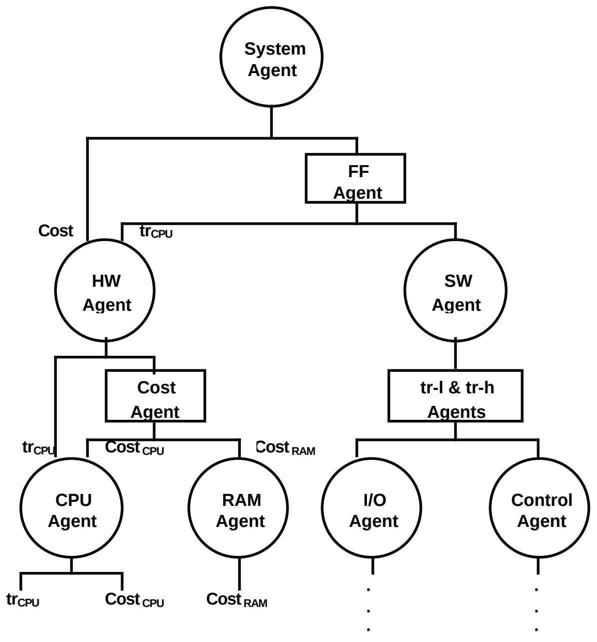
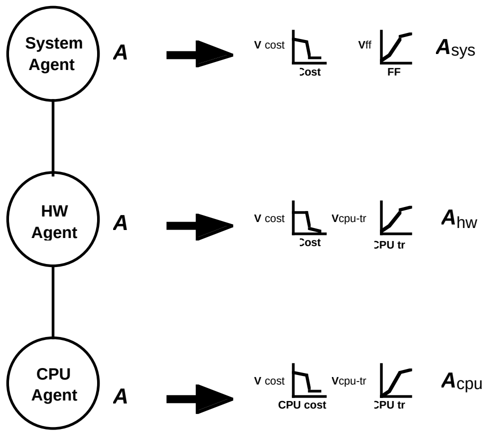
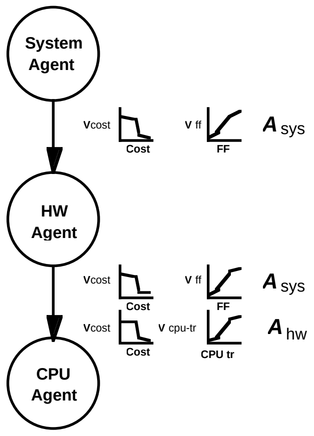


Figure 4.7: Agent value-function/attribute hierarchy.

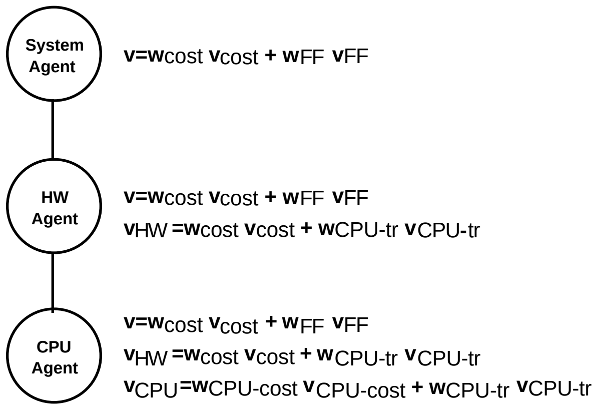
Figure 4.8 (b) shows how preferences are distributed from the system agent down to the CPU agent. Each agent takes the preferences sent to it, and sends these along with its own preferences down to any agents lower in the hierarchy. Each set of preferences implies a value function, resulting in multiple value functions for some agents (see Figure 4.8 (c)).



(a) Agents specify preferences



(b) Agents distribute preferences


System Agent
v=wcost vcost + wFF vFF
HW Agent
v=wcost vcost + wFF vFF
vHW =wcost vcost + wCPU-tr vCPU-tr
CPU Agent
v=wcost vcost + wFF vFF
vHW =wcost vcost + wCPU-tr vCPU-tr
vCPU=wCPU-cost vCPU-cost + wCPU-tr vCPU-tr

(c) Resulting value functions

Figure 4.8: Specification and distribution of preferences.

The set of value functions contained by an agent describes a hierarchy of preferences that must be obeyed. When considering the selection of an alternative, a agent first applies the value function that is highest in the hierarchy, yielding a set of nondominated alternatives. The agent then applies the next function in the hierarchy to the nondominated set, continuing this process until either there is only one nondominated alternative left, or all value functions have been applied. If more than one alternative still remains, then one is chosen at random.

Each value function represents a different view of the design, and requires different sets of attributes. To apply these value functions, an agent must isolate the effect of its local variables on the value of each respective view of the design. This is accomplished using theory developed in Section 3 (see Table 3.2). Thus, for each value function, Δv_n with respect to a design variable of the agent is found. For example, the value functions for the CPU, RAM, and I/O agents become:

CPU agent:

$$\Delta v = w_{\cos t}\Omega\left(v_{\cos t}, r_{\cos t}\right)\Delta r_{CPU\cos t} + w_{FF}\Omega\left(v_{FF}, r_{FF}\right)\Omega\left(r_{FF}, r_{CPUtr}, r_{tr-l}, r_{tr-u}\right)\Delta r_{CPU-tr}$$
$$\Delta v_{hw} = w_{hw\cos t}\Omega\left(v_{hw\cos t}, r_{\cos t}\right)\Delta r_{CPU\cos t} + w_{hwCPU-tr}\Omega\left(v_{hwCPU-tr}, r_{CPUtr}\right)\Delta r_{CPU-tr}$$
$$\Delta v_{CPU} = w_{CPU\cos t}\Omega\left(v_{CPU\cos t}, r_{CPU\cos t}\right)\Delta r_{CPU\cos t} + w_{CPUCPU-tr}\Omega\left(v_{CPUtr}, r_{CPUtr}\right)\Delta r_{CPU-tr}$$

RAM agent:

$$\Delta v = w_{\cos t}\,\Omega\left(v_{\cos t}, r_{\cos t}\right)\Delta r_{RAM\cos t}$$

$$\Delta v_{hw} = w_{hw\cos t}\,\Omega\left(v_{hw\cos t}, r_{\cos t}\right)\Delta r_{RAM\cos t}$$

$$\Delta v_{RAM} = w_{RAM\cos t}\,\Omega\left(v_{RAM\cos t}, r_{RAM\cos t}\right)\Delta r_{RAM\cos t}$$

I/O agent (x_{Filter}):

$$\Delta v = w_{FF}\,\Omega\left(v_{FF}, r_{FF}\right)\begin{bmatrix} \Omega\left(r_{FF}, r_{tr-l}, r_{CPU\,tr}, r_{tr-u}\right)\begin{bmatrix}\Omega\left(r_{tr-l}, r_{Filter\,dl}, \mathrm{K}\right)\Delta r_{Filter\,dl} + \\ \Omega\left(r_{tr-l}, r_{Filter\,per}, \mathrm{K}\right)\Delta r_{Filter\,per} + \\ \Omega\left(r_{tr-l}, r_{Filter\,act}, \mathrm{K}\right)\Delta r_{Filter\,act}\end{bmatrix} + \\ \Omega\left(r_{FF}, r_{tr-u}, r_{CPU\,tr}, r_{tr-l}\right)\begin{bmatrix}\Omega\left(r_{tr-l}, r_{Filter\,dl}, \mathrm{K}\right)\Delta r_{Filter\,dl} + \\ \Omega\left(r_{tr-l}, r_{Filter\,per}, \mathrm{K}\right)\Delta r_{Filter\,per} + \\ \Omega\left(r_{tr-l}, r_{Filter\,act}, \mathrm{K}\right)\Delta r_{Filter\,act}\end{bmatrix}\end{bmatrix}$$

$$\Delta v_{sw} = w_{sw\,tr-l}\,\Omega\left(v_{sw\,tr-l}, r_{tr-l}\right)\begin{bmatrix}\Omega\left(r_{tr-l}, r_{Filter\,dl}, \mathrm{K}\right)\Delta r_{Filter\,dl} + \\ \Omega\left(r_{tr-l}, r_{Filter\,per}, \mathrm{K}\right)\Delta r_{Filter\,per} + \\ \Omega\left(r_{tr-l}, r_{Filter\,act}, \mathrm{K}\right)\Delta r_{Filter\,act}\end{bmatrix} +$$

$$w_{sw\,tr-u}\,\Omega\left(v_{sw\,tr-u}, r_{tr-u}\right)\begin{bmatrix}\Omega\left(r_{tr-u}, r_{Filter\,dl}, \mathrm{K}\right)\Delta r_{Filter\,dl} + \\ \Omega\left(r_{tr-u}, r_{Filter\,per}, \mathrm{K}\right)\Delta r_{Filter\,per} + \\ \Omega\left(r_{tr-u}, r_{Filter\,act}, \mathrm{K}\right)\Delta r_{Filter\,act}\end{bmatrix}$$

$$\begin{aligned}\Delta v_{I/O} = {} & w_{I/O\,Filter-dl}\,\Omega\left(v_{I/O\,Filter-dl}, r_{Filter\,dl}\right)\Delta r_{Filter-dl} + \\ & w_{I/O\,Filter-per}\,\Omega\left(v_{I/O\,Filter-per}, r_{Filter\,per}\right)\Delta r_{Filter-per} + \\ & w_{I/O\,Filter-act}\,\Omega\left(v_{I/O\,Filter-act}, r_{Filter\,act}\right)\Delta r_{Filter-act}\end{aligned}$$

In the above equations, the w_k are constrained variables, with the range of possible assignments for a w_k dependent the ranges of the other w_k. The Ω_{kl} functions are also constrained variables, with the range of possible assignments for an Ω_{kl} dependent on the possible slopes for its corresponding v_k. The Δr are constants for any given evaluation of Δv_n. Although solving for Δv_n requires the solution of mathematical program with a polynomial objective function, the mathematical program can be simplified to a linear program by noting that values for the Ω_{kl} can be determined in isolation, before solving for Δv_n. In fact, the Ω_{kl} are often required by several agents, and the calculation of these values can be assigned to constraint agents, thus only one calculation is necessary, and the results can be shared. The assignment of Ω functions to constraint agents is shown in Figure 4.9. Note that when solving an ACDS network, only the agents responsible for assigning design variables participate in the solution process. With the delta value functions now isolated, agents have the preference structures necessary for decision making.

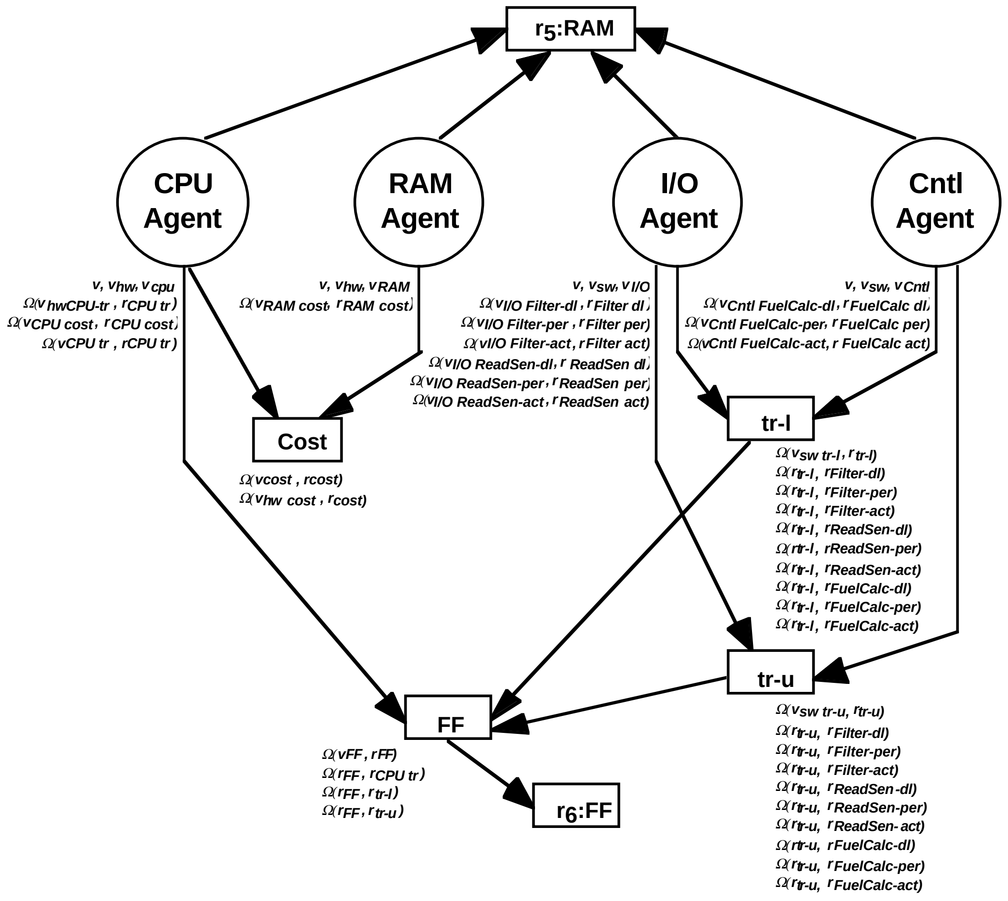


Figure 4.9: Assignment of Ω functions to constraint agents.

4.6 Decision Making

While achieving a decomposable network, or solving a decomposed network, an agent must communicate with other agents to apply preferences. For example, to choose a CPU, the CPU agent must first evaluate Δv for various pairs of alternative CPUs. To perform this evaluation, the CPU agents receives $\Omega(v_{cost}, r_{cost})$ from the cost constraint agent and $\Omega(v_{FF}, r_{FF})$ and $\Omega(r_{FF}, r_{CPU\,tr})$ from the feasibility factor constraint agent. With this information, the CPU agents solves a linear program to evaluate Δv for each pair of alternatives to be compared. If the agent is unable to identify a single nondominated alternative, the agent requests $\Omega(v_{hw\,cost}, r_{cost})$ from the cost constraint agent, and then locally evaluates $\Omega(v_{hwCPU\text{-}tr}, r_{CPU\,tr})$. With this information, the CPU agent evaluates Δv_{hw} for various pairs of the nondominated alternatives found by Δv. If more than one nondominated alternative still remains, then the CPU agent applies Δv_{CPU} in a similar manner.

We believe that calculating Δv_n will not introduce excessive communication overhead. If attribute value functions are nonlinear, or if attributes are nonadditive, then the calculation of Δv_n requires input from other agents. The worst-case scenario is that to calculate Δv_n, a design agent would have to receive information that requires input from every other design agent. We believe that for most cases the amount of information (*e.g.*, the number of attributes) required to find Δv_n will be less than the number required to calculate v. This belief follows from the fact that Δv_n is found by taking the derivative of v, which often results in the elimination of some terms in the equation, especially when v has an additive form. *The same level of global coordination can be achieved by evaluating either Δv_n or v, but* *evaluating Δv_n requires less communication*. In addition, some of the information required to determine Δv_n for one agent may also be required by other agents, and this information can be calculated once and shared among agents.

In this section, we have presented a basic outline of a new tool ACME, which facilitates hierarchical CE. Agents contract out subtasks, with their preferences and constraints forming the terms of the contract. The hierarchical preference structured created by subcontracting provides hierarchical control to a network of agents. Agents interact to solve a problem by means of distributed constraint satisfaction.

5.0 Summary and Discussion

In this paper, we have identified the need for a framework to solve CE problems in a hierarchical organization. Hierarchical organizations arise from hierarchical problem decomposition, subcontracting of subtasks, and supervisor and subordinate relationships. Hierarchical control provides global coordination to domain experts, who must utilize local expertise to solve a CE problem. Existing approaches focus on either solving a decomposed, single monolithic problem or peer-to-peer problem solving. What is needed is a combined approach that provides both global coordination and exploitation of local expertise.

In our preference-directed design research, we identified a preference model that facilitates multiattribute design evaluation. In addition, we developed a technique for isolating the value of a design variable, which is needed when partitioning is performed. The value of a design variable reflects the influence of that variable on the total value of the design. Finally, we developed and coded two design algorithms. Preference-directed optimal design provides optimal solutions for moderately sized problems, and preference-directed evolution provides good solutions to problems of any size.

To fill the gap in the existing CE research, and thus solve the problems related to hierarchical CE, we discussed a new tool, ACME, that builds upon our preference model and theory for isolating design-variable value functions. Each agent has the ability to contract out subtasks, and both the constraints and preferences of the agent are included in the contract. The preferences for a subtask are found by applying our theory for isolating the value of a design variable. Contracting of preferences creates a hierarchical preference structure, and provides hierarchical control of the network of agents. Agents with local expertise are free to apply their own preferences as long as they do not violate contracted preferences. Peer-to-peer problem solving is accomplished by distributed constraint satisfaction. In addition to software coding and experiments, several theoretical issues must still be resolved.

The main contributions this paper are the concept of hierarchical CE, the development of a technique for isolating subcontractor value functions from a general contractors function, and generation of methods for applying these functions during problem solving. This approach provides the benefits of global coordination in a decentralized problem solving environment.

At this time we are in the process of coding ACME. In addition to code development, a firm definition of the optimal solution for a hierarchical organization of agents must still be identified, as well as conditions when such a solution can be found. This may involve research from the area of mathematical programming, group decision making, and negotiation. Propagation of intervals through nonmonotonic functions may be required, and techniques based on the work Faltings (1994) and Hyvonen (1992) may prove useful. In addition, decomposition formulas, similar to those given in Table 3.2, are required to support a wider range of attribute types. Specifically, a complete set of formulas is needed for attribute functions that depend on multiple functions of a design variable. This is the case for the formulation of the hardware/software codesign problem given in Section 4, which required extensions to the formulas given in Table 3.2.

References

Azarm, S. and Li, W-C. (1988). A two-level decomposition method for design optimization, *Engineering Optimization*, Vol. 13, pp. 211-224.

Barthelemy, J-F. (1983). *Development of a Multilevel Optimization Approach to the Design of Modern Engineering Systems*, Ph. D. Dissertation, Virginia Polytechnic Institute and State University, University Microfilms International, Ann Arbor, Michigan.

Birmingham W.P., Gupta A.P., and Siewiorek D.P. (1992). *Automating the Design of Computer Systems*, Jones and Bartlett, Boston.

Bloebaum, C.L., Hajela, P., and Sobieszczanski-Sobieski, J. (1992). Non-hierarchic system decomposition in structural optimization, *Engineering Optimization*, Vol. 19, pp. 171-186.

Bradley, S.R., and Agogino, A.M. (1993). Computer-assisted catalog selection with multiple objectives, *ASME Design Theory and Methodology*, DE-Vol. 53, pp. 139-147.

Corkill, D.D., and Lesser, V.R. (1983). The use of meta-level control for coordination in a distributed problem solving network, Proceedings of the Eighth IJCAI, Karlsruhe, FRG., Morgain-Kaufmann, San Mateo, Calif., pp. 748-756.

Cutkosky, M.R., et al. (1993). PACT: an experiment in integrating concurrent engineering systems, *IEEE Computer*, January, pp. 28-37.

D'Ambrosio, J.G., et al. (1993). The role of analysis in hardware/software codesign, Presented *at The Second International Workshop on Hardware-Software Co-Design*.

D'Ambrosio, J.G., and Hu, X. (1994). Configuration-level hardware/software partitioning for real-time embedded systems, Presented at *The Third International Workshop on Hardware-Software Codesign*, Sept.

D'Ambrosio, J.G., and Birmingham, W.P. (1995). Preference-directed design, *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AI EDAM)*, Vol. 9, pp. 219-230.

Dantzig, G.B., and Wolfe, P. (1961). The decomposition algorithm for linear programming, *Econometrica*, Vol. 9, No. 4.

Darr, T. and Birmingham, W.P. (1994). Automated design for concurrent engineering, *IEEE Expert*, October.

Davis, D., and Smith, R.G. (1983). Negotiation as a metaphor for distributed problem solving, *Artificial Intelligence*, Vol. 20, pp. 63-109.

Davis, E. (1987). Constraint propagation with interval labels, *Artificial Intelligence*, Vol. 32, pp. 32-44.

Dechter, R., and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems, *Artificial Intelligence*, Vol. 34, No. 1, pp. 1-38.

Dean, T. and Boddy, M. (1988). An analysis of time-dependent planning, *Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Minneapolis, MN, August, pp. 49-54.

Durfee, E.H. and Montgomery, T.A. (1991). Coordination as distributed search in a hierarchical behavior space, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No 6., pp. 1363-1378.

Ephrati, E. and Rosenschein, J.S. (1992). Planning to please: planning while constrained by a master agent, *Working Papers of The Eleventh International Workshop on Distributed Artificial Intelligence*, Glen Arbor, Michigan, pp. 77-94.

Faltings, B. (1994). Arc-consistency for continuous variables, *Artificial Intelligence*, Vol. 65, pp. 363-379.

Fox, M.S. (1981). An organizational view of distributed systems, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 1, January, pp. 70-80.

Gebotys, C.H., and Elmasry, M.I. (1993). Global optimization approach for architectural synthesis, Computer-*Aided Design of Integrated Circuits and Systems*, Vol. 12, Sept., pp. 1266-1278.

Gorman, W.M. (1968). The structure of utility functions, *Review of Economic Studies*, Vol. 35, pp. 367-390.

Grossmann, I.E. (1990). Mixed-integer nonlinear programming techniques for synthesis of engineering systems, *Research in Engineering Design*, Vol. I, pp. 205-228.

Haimes, Y.Y. (1981). Hierarchical holographic modeling, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 9, Sept., pp. 606-617.

Haimes, Y.Y. and Hall, W.A. (1974). Multiobjectives in water resources systems analysis: the surrogate worth trade-off methods, *Water Resources Research*, Vol. 10, No. 4, pp. 615-624.

Haimes, Y.Y., et al. (1990). *Hierarchical Multiobjective Analysis of Large-Scale Systems*, Hemisphere Publishing Corp., New York.

Haworth, M.S., and Birmingham, W.P. (1993). Towards optimal system-level design, *Proceedings of 30th Design Automation Conference*, June, pp. 434-438.

Horvitz, E.J. (1988). Reasoning under varying and uncertain resource constraints, *Proceedings AAAI-88 Seventh National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Minneapolis, MN, August, pp. 111-116.

Hu, X., D'Ambrosio, J.G., Murray, B.T., and Tang, D. (1994). Codesign of architectures for automotive powertrain modules, *IEEE Micro*, August, pp. 17-25.

Huang, G.Q. and Brandon, J.A. (1993). Agents for cooperating expert systems in concurrent engineering design, , *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, Vol. 7, No. 3, pp. 145-158.

Huyn, P.N., Genesereth, M.R., and Letsinger, L. (1993). Automated concurrent engineering in DesignWorld, *IEEE Computer*, January, pp. 74-76.

Hyvonen, E. (1992). Constraint reasoning based on interval arithmetic: the tolerance propagation approach, *Artificial Intelligence*, Vol. 58, pp. 71-112.

Ishida, T. (1992). The tower of babel: towards organization-centered problem solving, *Working Papers of The Eleventh International Workshop on Distributed Artificial Intelligence*, Glen Arbor, Michigan, pp. 141-153.

Keeney, R.L., and Raiffa, H. (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, Wiley and Sons, New York.

Krantz, D.H., Luce, R.D., Suppes, P., and Tversky, A. (1971). *Foundations of Measurement*, Academic, New York, Vol. 1.

Li, W-C. and Azarm, S. (1989). Parameter sensitivity analysis in two-level design optimization, Advances in Design Automation, ASME Publication DE-Vol. 19-2, Design Optimization, B. Ravani (ed.), pp. 31-38, 15th Design Automation Conference, Montreal, Quebec, Canada, September.

Macko, D. and Haimes, Y.Y. (1978). Overlapping coordination of hierarchical structures, IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-8, No. 10, pp. 745-751.

Mackworth, A.K. (1987). Constraint satisfaction, In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, John Wiley & Sons: New York.

Mesarovic, M.D., Macko, D., and Takahara, Y. (1970). Theory of hierarchical multilevel, systems, *Mathematics in Science and Engineering*, Vol. 68, Academic Press, New York.

Michelena, N., and Papalambros, P.Y. (1994). A network reliability approach to optimal decomposition of design problems, *Advances in Design Automation-1994*, B.J. Gilmore (ed.), Vol. 2, ASME, New York, pp. 195-204.

Michelena, N., and Papalammbros, P.Y. (1995). A hypergraph framework for optimal model-based decomposition of design problems, *Univ. of Michigan Technical Report UM-MEAM 95-02*.

Mittal, S. and Falkenhainer, B. (1990). Dynamic constraint satisfaction problems, *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90).*

Mittal, S., and Frayman, F. (1987). COSSACK: a constraints-based expert system for configuration tasks, *Proc. 2nd Int'l Conf. on Applications of AI to Engineering*.

Montgomery, T.A. and Durfee, E.H. (1993). Search reduction in hierarchical distributed problem solving, *Group Decision and Negotiation*, Vol. 2, pp. 301-317.

Murty, K.G. (1983). *Linear Programming*, John Wiley & Sons: New York.

Nemhauser, G.L., *et al*. (1989). *Handbooks in Operations Research and Management Science: Optimization*, Vol. 1, North-Holland, New York.

Pan, J.Y.C. and Tenenbaum, J.M. (1991). An intelligent agent framework for enterprise integration, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No. 6, pp. 1391-1408.

Papalambros, P. and Wilde, D.J. (1988). *Principles of Optimal Design: Modeling and Computation*, Cambridge University Press.

Park, H., et al. (1994). An agent-based approach to concurrent cable harness design, *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, Vol. 8, pp. 45-61.

Pekny, J.F. (1992). Combinatorial optimization in engineering systems: exploiting problem structure and parallelism, *NSF Design and Manufacturing Systems Conference*.

Renaud, J.E. and Gabriele, G.A. (1993). Improved coordination in nonhierarchic system optimization, *AIAA Journal*, Vol. 31, No. 12, December, pp. 2367-205.

Schoeffler, J.D. (1971). Static multilevel systems, *Optimization Methods for Large-Scale Systems ...with Applications*, D.A. Wismer (ed.), McGraw-Hill Book Company, New York, pp. 1-46.

Seo, F., and Sakawa, M. (1982). Mathematical optimization for multiobjective decision making, *Multiobjective and Stochastic Optimization*, M. Grauer, A. Lewandowski, and A.P. Wierzbicki (eds.), IIASA Collaborative Proceedings Series, CP-82-S12, IIASA, Austria, pp. 169-302.

Sobiezczanski-Sobieski, J. (1982). A linear decomposition method for large optimization problems - blueprint for development, *NASA TM 83248,* Feb.

Sobieszczanski-Sobieski, J. (1990). Sensitivity of complex, internally coupled systems, *AIAA Journal*, Vol. 28, No. 1, January, pp. 153-160.

Sykes, E.A., and White, C.C. (1991). Multiobjective intelligent computer-aided design, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 21, No. 6, November/December, pp. 1498-1511.

Thurston, D.L. (1991). A formal method for subjective design evaluation with multiple attributes, *Research in Engineering Design*, Vol. 3, pp. 105-122.

Tarvainen, K., and Haimes, Y.Y. (1981). Hierarchical-multiobjective framework for energy storage systems, *Organizations: Multiple Agents with Multiple Criteria*, Lecture Notes in Economics and Mathematical Systems, M. Beckman and H.P. Kunzi (eds.), Springer-Verlag, New York, pp. 424-446.

Vanderplaats, G.N., and Sugimoto, H. (1985). Numerical optimization techniques for mechanical design, *Design and Synthesis*, Elsevier Science Publishers B. V (North-Holland).

Wagner, T.C., and Papalammbros, P.Y., (1993). A general framework for decomposition analysis in optimal design, *Advances in Design Automation*, B.J. Gilmore (ed.), Vol. 2, ASME, New York, pp. 315-325.

Wagner, T.C., and Papalammbros, P.Y., (1993). Implementation of decomposition analysis in optimal design, *Advances in Design Automation*, B.J. Gilmore (ed.), Vol. 2, ASME, New York, pp. 327-335.

Werkman, K.J. (1992). Multiple agent cooperative design evaluation using negotiation, *Artificial Intelligence in Design '92*, Kluwer Academic Publishers.

White, C.C., *et al*. (1984). A model of multi-attribute decision making and trade-off weight determination under uncertainty, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-14, No. 2, March/April, pp. 223-229.

Wood, K.L., and Antonsson, E.K. (1989). Computations with imprecise parameters in engineering design: background and theory, *Journal of Mechanisms, Transmissions and Automation in Design*, Vol. 111, Dec., pp. 616-625.

Wood, K.L., Antonsson, E.K., and Beck, J.L. (1990). Representing imprecision in engineering design: comparing fuzzy analysis and probability calculus, *Research in Engineering Design*, Vol. 1., pp. 187-203.

Wu, J., *et al*. (1990). A model-based expert system for digital system design, *IEEE Design and Test of Computers*.

Wujek, B.A., Renaud, J.E., Batill, S.M., and Brockman, J.B. (1995). Concurrent subspace optimization using design variable sharing in a distributed computing environment, Accepted for publication in *ASME 21st Design Automation Conference*, Boston, MA, September.

Yokoo, M., Ishida, T., and Kuwabara, K. (1990). Distributed constraint satisfaction for DAI problems, *Proceedings of the 1990 distributed AI workshop*, Bandara, Texas, October.

Yokoo, M. and Durfee, E.H. (1991). Distributed constraint optimization as a formal model of partially adversarial cooperation, *The University of Michigan Technical Report CSE-TR-101-91*, Ann Arbor, Michigan.