CONCURRENT ENGINEERING: Research and Applications

Hierarchical Concurrent Engineering in a Multiagent Framework

Joseph D'Ambrosio, Timothy Darr and William Birmingham*

Artificial Intelligence Lab, EECS Department, The University of Michigan, Ann Arbor, MI 48109

Received 11 August 1995; accepted in revised form 12 January 1996

Abstract: Our experience indicates coordination in concurrent engineering (CE) requires support for two types of relationships among decision makers: supervisor/subordinate and peer-to-peer. Supervisor/subordinate relationships are created by the standard hierarchical decomposition process that is required to solve any large design problem. Peer-to-peer relationships arise when teams of decision makers must interact, without direct guidance, to achieve individual and common goals. In this paper, we describe a general decision-making methodology, which we call hierarchical CE. The emphasis of hierarchical CE is to provide support for both supervisor/subordinate and peer-to-peer relationships. In addition to the concept of hierarchical CE, we present a supporting agent-based framework in which the preferences and constraints of a design supervisor are distributed to design subordinates, who are expected to exploit their local expertise within the context provided by this global information. A distinct separation between feasibility and value facilitates optimal decision-making by design agents, since the bounds on feasibility do not include arbitrary statements about value. This distinction may prove useful for other problem domains as well.

Key Words: concurrent engineering, value function, distributed constraint satisfaction, multilevel optimization, agents, hierarchical decomposition.

1. Introduction

The core constituents of (future) CE organizations will be, in our view, *autonomous* agents that represent various facets of the design organization. These agents will come from both inside and outside an enterprise. These agents transcend the typical notion of agents, as they have models of and can reason about their capabilities. They are truly autonomous in the sense that an agent cannot be compelled to act in particular ways by other agents (e.g., to join teams or render particular services), but rather chooses to participate or not in endeavors strictly because it is in its own interests to do so. Thus, there are few truly central controls or global, shared goals imposed on agents. Instead, organizations may influence the behavior of their constituents indirectly through incentives, which might be more or less tangible, and more or less compelling in varying circumstances.

Uncontrolled autonomy is not necessarily a desirable goal in concurrent engineering for the following reason. Suppose that each member of a design team (which may include thousands of participants) is represented by an agent with the capability to communicate instantaneously with other agents throughout the design process, and all agents share a common representation and ontology. In other words, the traditional CE goals of fast communication and shared representations have been met [1,2]. While these goals are important, it is easy to see that chaos will result without a mechanism to enforce coordination among the team members for resolving conflicts.

Coordination of agents therefore becomes the central issue in CE. It is necessary to ensure that all constituents in a design organization are committed to overall organizational objectives (e.g., increase market share), while allowing agents to make their own decisions in the appropriate way.

Our experience indicates coordination in CE requires support for two types of relationships among decision makers: supervisor/subordinate and peer-to-peer. Supervisor/subordinate relationships are created by the standard hierarchical decomposition process that is required to solve any large design problem [see Figure 1(a)]. A subordinate receives specifications from a supervisor, and is required to implement a subsystem within the limits defined in the specification. Peer-to-peer relationships arise when teams of decision makers must interact, without direct guidance, to achieve individual and common goals. For example, life-cycle engineering requires interaction among domain experts, each with their own preferences and expertise [see Figure 1(b)]. We refer to CE that supports both types of relationships as *hierarchical CE.*

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 a viable approach to subtask distribution [3], creates a hierarchical, decison-making organization. Coordination is accomplished by allowing general contractors, who have a global

*Author to whom correspondence should be addressed.

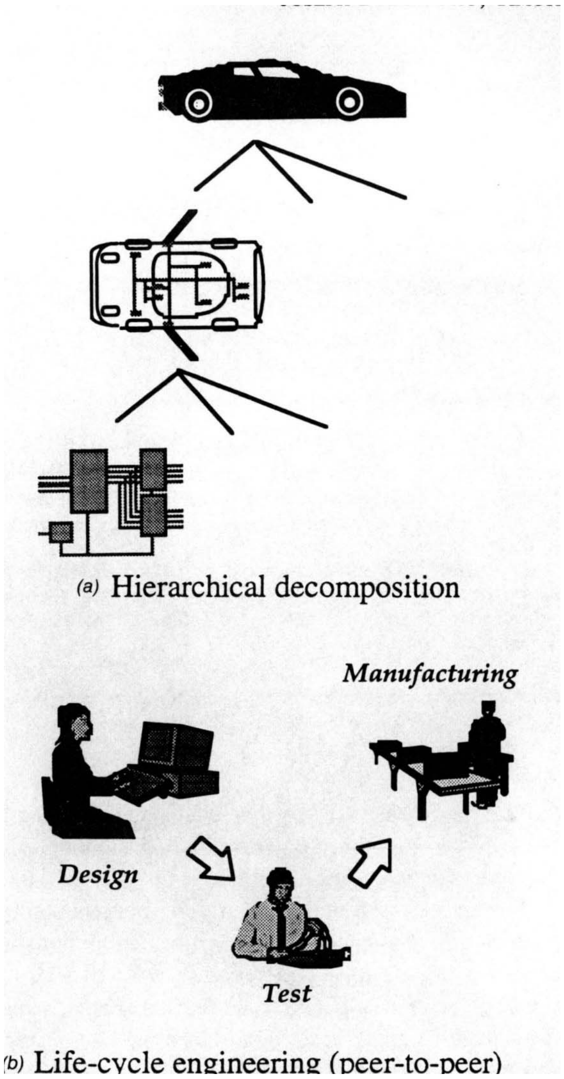

(a) Hierarchical decomposition
Manufacturing
Design
Test
(b) Life-cycle engineering (peer-to-peer)

Figure 1. Sources of relationships among decision makers.

perspective, to provide direction to subcontractors, who have local expertise.

Figure 2 illustrates our view of a typical hierarchical CE problem. A supervisor agent contracts out a CE problem to three domain agents, DA_1, DA_2, and DA_3. DA_1 in turn subcontracts its responsibilities to two other agents, DA_4 and DA_5. Problem solving takes place through the interaction of agents DA_2, DA_3, DA_4, and DA_5, with coordination provided by the supervisor agent and DA_1. 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.

We believe that it is both 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.

In this paper, we describe a general decision-making methodology and agent architecture for solving hierarchical CE problems. In our approach, the preferences and constraints of a supervisor are distributed to subordinates, who are expected to exploit their local expertise within the context provided by this global information. Thus, global coordination is achieved and the preferences of all decision makers play a role in problem solving.

To support hierarchical CE, we have defined an agent architecture based on a clear, *theoretical* distinction between decisions about *feasibility* and decisions about *preference* or value. By separating decisions about feasibility and value, we can achieve optimal decisions, since the bounds on feasibility do not include arbitrary statements about value. This is in contrast to other approaches that attempt to capture preferences by introducing artificial bounds on feasibility, which has the negative consequence of possibly eliminating the best solutions. For example, specifying constraints such as total cost $\leq \$100$ and total weight ≤ 10 lbs. fails to take into account the possibility that the designer is willing to accept 10.1 lbs. in total weight in order to save \$10 in total cost.

In the remainder of this paper, we first describe how our research relates to previous work. Next, we provide a formal model of the problem to be solved, and describe our agent framework. We then describe how an existing tool, The Automated Configuration-Design Service (ACDS), can be extended to create a new tool, ACME, which meets the needs of hierarchical CE. Finally, we provide a discussion of our current results.

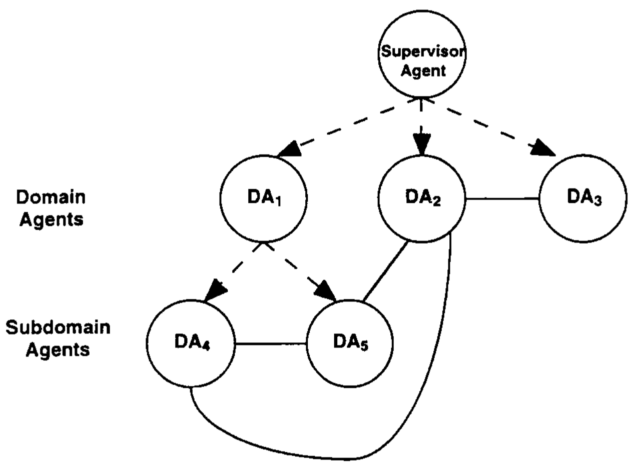


Figure 2. A hierarchical agent organization. Solid lines indicate problem-solving networks, arrows indicate control relationships.

2. Previous Work

Currently, there exist two primary approaches for solving CE problems: multilevel optimization and agent-based systems. The multilevel optimization approach is based on mathematical programming techniques for decomposing and optimizing large systems [4]. 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. Examples of this approach include the work of Macko and Haimes [5], Haimes [6], Azarm and Li [7], and Sobieski [8].

The work of Sobieski 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 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. An *optimal* solution is obtained by restricting the definition of optimality to the preferences of the global coordinator, and ignoring the preferences of domain (discipline) experts. Although Haimes' hierarchical holographic modeling provides a method for capturing and applying domain preferences, it only supports peer agents, and as such, does not provide a framework for an agent hierarchy. This approach is similar to agent-based systems, which will be discussed next.

Several agent-based systems focus on the needs of cooperative problem solving among domain experts. PACT [1] is a testbed for building large-scale, distributed CE systems, where agents are grouped by discipline and communicate through facilitators [9]. First-Link [10] emphasizes collaboration among specialists and the development of hierarchical design representations. DesignWorld [11] is an automated engineering environment for the design and manufacturing of digital circuits, which is built upon the concept of facilitators. 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 systems have attempted to address the need for hierarchical control and preferences. DFI [12] 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 specifiying 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 [13,14], a system agent distributes its preferences to catalog agents, who use these preferences to select components. In this system, there is no ability for catalog agents to apply their own expertise (preferences). AGENTS [15] is an 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 multilevel optimization nor agent-based systems provide a general framework for hierarchical preferences. What is needed is an approach that provides global coordination, similar to multilevel optimization, and exploits local expertise, similar to agent-based systems. In the next section, we formally define a specific design problem to solve. We then describe how our agent framework can provide a hierarchical CE environment to solve this type of design problem.

3. Problem Definition

In this paper, we restrict the problem domain to a common class of design problems characterized by the selection of parts (components) from catalogs. For example, given a catalog containing CPUs such as 68040 and Intel P6, the 68040 might be selected to implement the function CPU. The selected parts must also satisfy any specified constraints and be optimal in some sense. We define the problem formally below.

Given:

- A set of parts, possibly distributed among several catalogs, that implement the desired functions.

$$P = \{p_{11}, \ldots, p_{1j}, \ldots, p_{nj}\}$$

where

p_{nj} = the jth part that implements function n

$p_{nj}.a_k$ = the level[1] of attribute k for p_{nj} (e.g., 68040.cost = \$100)

- A set of discrete design variables that represent the functions to be implemented. The domain of each design variable is the corresponding set of parts that implement the function.

$$X = \{x_1, \ldots, x_n\}$$

$$x_n = \{p_{nj} | p_{nj} \text{ is a part that implements function } x_n\}$$

$$x_n.a_k = p_{nj}.a_k \text{ if } x_n, p_{nj}, \text{ for all } k$$

[1]To avoid confusion in this paper, we restrict the use of the term *value* to the context of formal value functions (see Section 4.2), and we use the term *level* when discussing raw data values (e.g., power = 3W) or the value of a variable assignment (e.g., $x = 2$).

- A set of intermediate variables for specifying attributes and constraints.

$I = \{i_1, \ldots, i_y\}$

$i_y = f(X), i_y \in R$

- A set of design attributes that directly contribute to evaluating the optimality of a design.

$A = \{a_1, a_2, \ldots, a_k\}$

$a_k = f(X,I), a_k \in R$

- A multiattribute value function v that captures designer preferences [16].

$$v = \sum_k w_k \, v_k \, (a_k)$$

where

w_k = a relative attribute weight
$v_k(a_k)$ = an attribute value function (see Section 4.2 for a definition of value function).

- A set of constraints that define feasibility and interoperability.

$C = \{c_1, c_2, \ldots, c_m\}$

$c_m = f(X, I, A).$

Find:

- A set of parts $S_{opt} = \{p_{ij}, \ldots, p_{nj}\}$, such that:
  - S_{opt} is a feasible solution (it satisfies C)
  - $v(S_{opt}) \geq v(S)$ for every feasible S identified.

Although the above problem definition is targeted toward simple part selection problems, it can be easily extended to address other types of configuration design problems, such as those that must support multifunction parts and different possible component configurations.

In the next section, we describe our agent framework, with emphasis placed on modeling of preferences. Our agent preference model provides a key component for supporting hierarchical CE.

4. Agent Decision Making

Once an agent contracts its services to participate in a design, we view the agent as making two classes of decisions relating to the design process: those about feasibility and those about value of feasible designs. In this section, we provide details of these two aspects of decision making.

4.1 Modeling Feasibility

We model the task of determining the feasibility of a design as solving a distributed, dynamic, interval constraint-satisfaction problem (DDICSP) [13]. A constraint satisfaction problem (CSP) is characterized by a set of variables to assign and a set of constraints that restrict the possible assignments to the variables. In relation to the previous problem definition, the variables are those contained in X, I, and A, and the constraints are those in C. With a distributed CSP, variables and constraints are distributed among agents, and a distributed problem-solving algorithm must be applied to derive variable assignments. A distributed problem-solving approach is desirable since solving hierarchical CE problems involves the interaction of many independent decision makers. The DDICSP model has several desirable properties, including providing an efficient technique for managing the problem's inherent exponential complexity, and a structure that facilitates identification of heuristics suitable for particular design domains.

As presented in the problem definition, CSP constraints are used to capture interoperability and feasibility relationships. In other words, they are restricted to describing what is physically possible, rather than what is desirable from the perspective of the design organization. Modeling this sort of preference information is the topic of the next section.

4.2 Modeling Preferences

Many design problems are characterized by multiple conflicting attributes, and formal models of preferences [17,18] have shown promise in resolving conflicting design objectives. A formal model, e.g., a *multiattribute value 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 solutions produced is difficult to evaluate.

Although there are many forms of value functions, an additive value function is a simple form that is applicable to many problems. An additive value function is a weighted sum of attribute values, such that the value of an alternative S with k attributes is given by:

$$v(S) = \sum_k w_k \, v_k(a_k) \qquad (1)$$

where

w_k = the tradeoff weight for attribute a_k
$v_k \, (a_k)$ = the value produced by the value function of attribute a_k

Keeney and Raiffa [16] describe the process for determining the attribute value functions and trade-off weights.

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) [19] creates a partial order based on preference relationships among a subset of alternatives. ISMAUT uses a weighted sum of attribute values [see Equation (1)], where all weights must be positive and their sum must be unity:

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

$$\sum_k w_k = 1$$

A preference statement of the form "S_i is preferred to S_j" means that $v(S_i) \geq v(S_j)$, and implies an inequality in the space of possible weights according to the following relation:

$$\Delta v = v(S_i) - v(S_j) = \sum_k w_k\,[v_k(a_{ik}) - v_k(a_{jk})] \geq 0$$

According to this interpretation, the statement that S_i is preferred to S_j means that the trade-off weights are such that the total weighted value of S_i is at least as great as that of S_j. 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 $v(S)$ can order pairs of alternatives other than those specified by the designer: S_i is preferred to S_j if, for every possible vector of weights $< w_1, w_2, \ldots, w_k >$ within W', the value of S_i is greater than the value of S_j, for example,

$$\text{Min}\ \Delta v = \text{Min} \sum_k w_k\,[v_k(a_{ik}) - v_k(a_{jk})] \geq 0,\ w_k \in W'$$

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 by ranking a *sample* of alternatives create a partial order over *all* design alternatives, and this partial order can identify the *nondominated* set of design alternatives. The nondominated set contains only those alternatives for which no other preferred alternative exists based on specified preferences. The nondominated set of alternatives is guaranteed to contain the optimal one [18].

In the next section, we describe how agents interact to perform hierarchical CE. As will be seen, the agent-preference model plays a key role in applying local expertise and achieving overall coordination.

5. Agent Coordination in a Multilevel Environment (ACME)

We now describe how an existing CE tool, ACDS [13,14] can be extended to meet the needs of hierarchical CE. We call the resulting tool ACME, which provides support for both hierarchical and peer-to-peer problem solving. When subcontracting a task in the ACME environment, an agent specifies both the constraints and preferences for the subtask. By agreeing to obey the contractor's preferences, subcontractors collaborate in a manner consistent with global preferences.

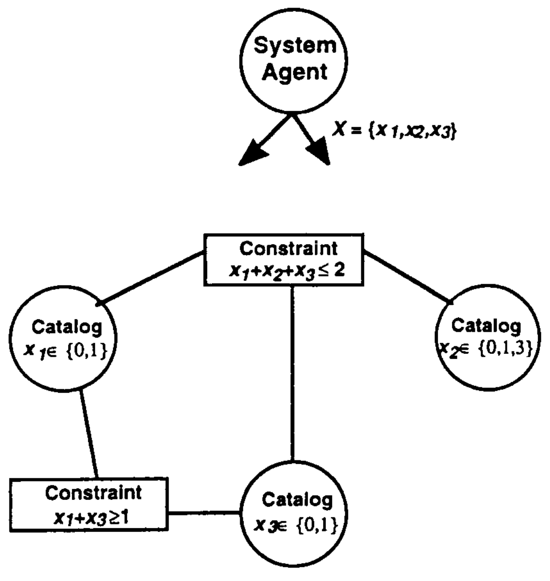
5.1 ACDS Overview

ACDS solves a class of DDICSPs and has the ability to model design feasibility. In terms of the previously provided problem definition, ACDS can achieve all criteria except that of the multiattribute value function; the ACDS cost function is too restrictive to model the preferences necessary for hierarchical CE. In addition, ACDS does not provide the ability to subcontract tasks.

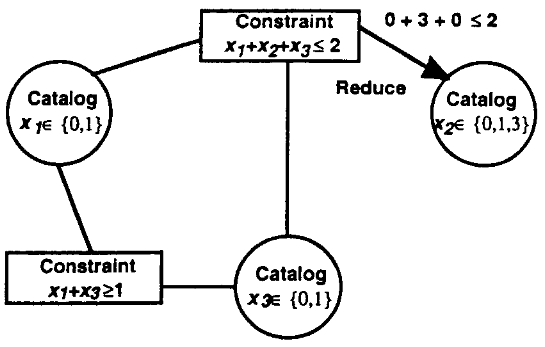
A DDICSP, implemented as a network of agents, can solve catalog-based design problems. A network is comprised of a system agent, catalog agents, and constraint agents. The system agent specifies the problem to be solved. Each catalog agent has the responsibility of finding a set of parts for the design variables it can assign such that constraints are satisfied. Constraint agents calculate attribute levels and monitor constraint conditions.

Solving a problem proceeds as follows. The system agent broadcasts to a network of catalog agents the functions to implement (variables to be assigned). Catalog agents respond with a desire to participate in the design. Currently, the catalog agents participate in the design if their catalog contains parts that implement the specified functions. No other measure of worth from the perspective of the catalog agent is taken into consideration. Constraint agents are then created by the system agent or catalog agents. In the example given in Figure 3(a), three catalog agents are available to implement the three variables, x_1, x_2, and x_3, broadcast by the system agent, and two constraint agents have been created. In this simple example, the design problem involves assigning integer levels to the three variables, as opposed to assigning parts. A true part assignment example is presented later in this paper.

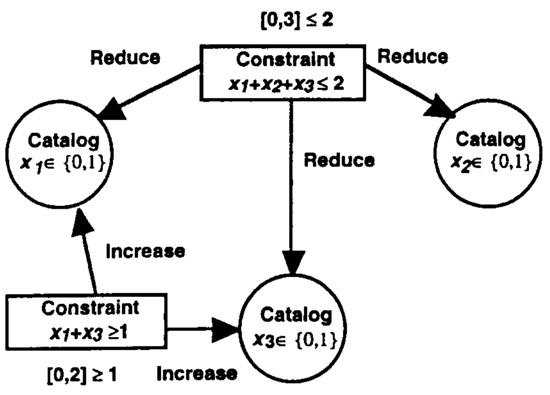
Once created, the constraint agents request and check the current range of possible assignments for each variable, and report *consistency* violations to the catalog agents. A consistency violation exists for a constraint if there is a possible assignment of one variable contained in the constraint for which no possible assignment of the remaining variables satisfies the constraint. In Figure 3(b), a constraint agent detects a consistency violation, since if $x_2 = 3$, there is no possible assignment to x_1 and x_3 that will satisfy the con-



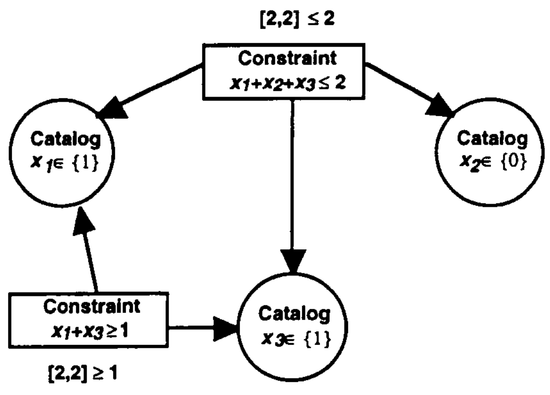
(a) Creating a network.



(b) Detecting consistency violations.



(c) Removing constraint violations.



(d) A decomposed, solved network

Figure 3. Solving an ACDS problem.

straint. The catalogs then prune their domains to eliminate consistency violations.

After a consistent network is achieved, the constraint agents then check to see if any of the attribute intervals violate constraints. A constraint violation exists if any possible set of variable assignments violates a constraint. If a violation exists, the constraint agents direct the catalog agents to modify previous assignments to reduce the constraint violations, backtracking, if necessary. In Figure 3(c), the top constraint agent detects that the sum of the three variables may be greater than two, so the maximum level of the variables must be reduced. This process is repeated until no constraint violations remain [see Figure 3(d)].

A network where no constraint violations are present is called a *decomposable* network. Once a network is decomposable, any possible assignment of variables results in a feasible design. At this point, a shared cost function, specified by the system agent, is applied to select the best assignment in each catalog. For the example given in Figure 3, there is only one possible assignment for each of the variables, so there is no need to perform this step. Note that in general there may be a large number of possible assignments remaining for each variable.

5.2 ACME

Extending the ACDS algorithm to include a network of agents with their own preferences and subcontracting abilities 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.

In addition, preferences must be applied 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, the current ACDS constraint-satisfaction approach can be augmented with search methods similar to those applied by Sobieski [8]. These methods focus not only on feasibility to direct the distributed search (see Figure 3), but value as well. For example, when considering which parts to bid to reduce a set of constraint violations, an ACDS catalog agent bids a set of parts such that each constraint violation is reduced. An alternative approach would be for a catalog agent to bid the single part that minimizes value loss, while still reducing at least one of the constraint violations. Heuristics, such as the one just discussed, help guide the distributed search to identify design alternatives that both satisfy constraints and maximize value.

Since ACME supports a wider range of value functions, and thus can represent a wider range of preferential knowledge, the difficulty related to solving a decomposable network is increased. Once a decomposable network is achieved in ACDS, catalog agents concurrently assign variables using a shared cost function. The ACME multiattribute value function may be constructed based on nonlinear attribute value functions, which couple variable assignments in relation to determining value similar to the way constraints couple variable assignments in relation to determining feasibility. As a result, variable assignments cannot be made concurrently, but must be done in a manner similar to achieving a decomposable network. Specifically, just as consistency checks eliminate possible variable assignments while attempting to achieve a decomposable network, dominance checks (see Section 4.2) eliminate possible assignments while solving a decomposable network. To perform distributed dominance checks, attribute ranges must be propagated through the equations defining the multiattribute value function. Thus, in addition to checking constraint equations, constraint agents must also propagate attribute levels to design agents, who can then perform dominance checks.

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.

5.2.1 EXAMPLE PROBLEM

For purposes of illustration, we will describe our approach using a part selection CE problem focused on hardware-software codesign. In this application, the parts to be selected include both hardware and software components. The example presented here is a simplified version of the problem described by Hu et al. [20]. A system agent wishes to create the embedded controller shown in Figure 4. 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 computerboard 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,

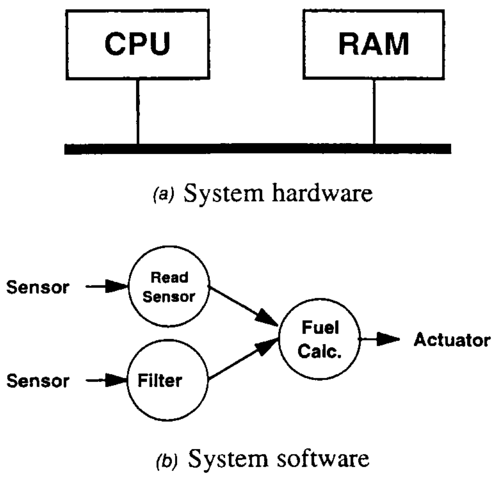


Figure 4. Embedded controller example.

and feasibility factor [21] 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 5. 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 (a_{cost}) and feasibility factor (a_{FF}), are determined by two constraints, c_{cost} and c_{FF}, respectively. The c_{cost} constraint is expressed in terms of functions of independent variables, while the c_{FF} constraint is expressed in terms of both a function of an independent variable, x_{CPU}, and intermediate dependent variables, $i_{tr\text{-}l}$ and $i_{tr\text{-}u}$. These intermediate variables provide a lower and upper bound of the amount of processing capacity required by the software. In

$$
\begin{aligned}
&X = \{\, x_{Filter}, x_{ReadSensor}, x_{FuelCalc}, x_{CPU}, x_{RAM} \}\\
&I = \{i_{tr\text{-}l}, i_{tr\text{-}h}\}\\
&A = \{a_{cost}, a_{FF}\}\\
&v = w_{cost} v_{cost} + w_{FF} v_{FF}\\
&\qquad v_{cost} = f(a_{cost})\\
&\qquad v_{FF} = f(a_{FF})\\
&C = \{\\
&\qquad c_{cost} : a_{cost} = f(x_{CPU}, x_{RAM}),\\
&\qquad c_{FF} : a_{FF} = f(x_{CPU}, i_{tr\text{-}l}, i_{tr\text{-}u}),\\
&\qquad c_{tr\text{-}l} : i_{tr\text{-}l} = f(x_{Filter}, x_{ReadSensor}, x_{FuelCalc}),\\
&\qquad c_{tr\text{-}u} : i_{tr\text{-}u} = f(x_{Filter}, x_{ReadSensor}, x_{FuelCalc}),\\
&\qquad c_5 : f(x_{CPU}, x_{RAM}) \geq f(x_{Filter}, x_{ReadSensor}, x_{FuelCalc}),\\
&\qquad c_6 : a_{FF} \geq 0.0\\
&\}
\end{aligned}
$$

Figure 5. Problem formulation.

addition to constraints for calculating attribute values, two constraints, c_5 and c_6, determine the feasibility of a solution. c_5 ensures enough RAM exists to meet the needs of the software modules, and c_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 6. 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 input/output (I/O). The organization headed by the software agent selects the appropriate software modules, including modules for low-level I/O operations as well as high-level control algorithms.

5.2.2 ESTABLISHING A NETWORK OF DESIGN AGENTS

The system agent assigns responsibility for each independent design variable to a design agent and each constraint to a constraint agent. Assignment of variables to design agents is performed using an approach similar to the contract-net negotiation process [3]. 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.

Figure 7 illustrates the assignment of variables to design agents. The system agent requests bids for the following variables: x_{Filter}, $x_{ReadSensor}$, $x_{FuelCalc}$, x_{CPU}, x_{RAM}. The hardware agents bid 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.

5.2.3 ESTABLISHING CONSTRAINT AGENTS

Figure 8 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

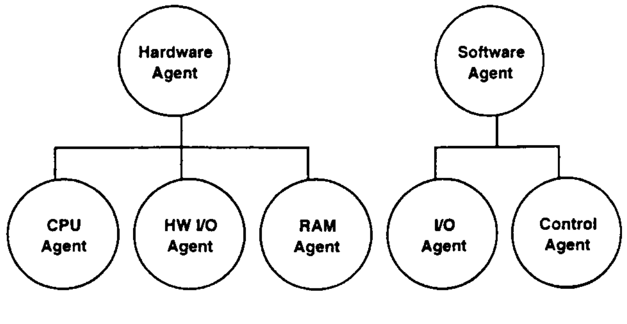


Figure 6. Two groups of design agents available to implement the embedded system.

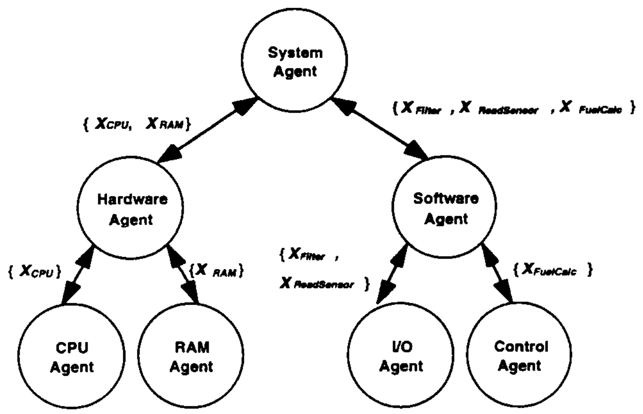


Figure 7. Forming a design organization.

(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.

5.2.4 DISTRIBUTING PREFERENCES

With constraints among agents established, agents can proceed to specify preferences. First a random sample of designs, $S^* = \{S_1, \ldots, S_i\}$, is generated and distributed to all agents. The only restriction on the elements in S^* is that they must be feasible. The process for finding a feasible alternative is identical to solving for the most preferred alternative, except that any set of preferences may be used.

Next, the agents hierarchically define preferences by ranking elements in the sample, S^*, and defining attribute value functions [see Figure 9(a)]. The system agent specifies

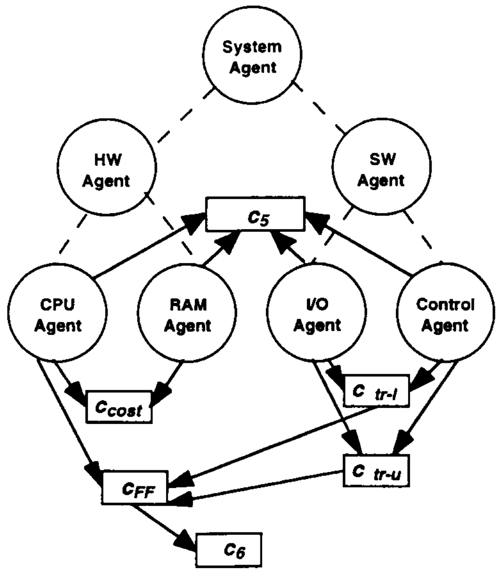


Figure 8. Creating constraint agents.

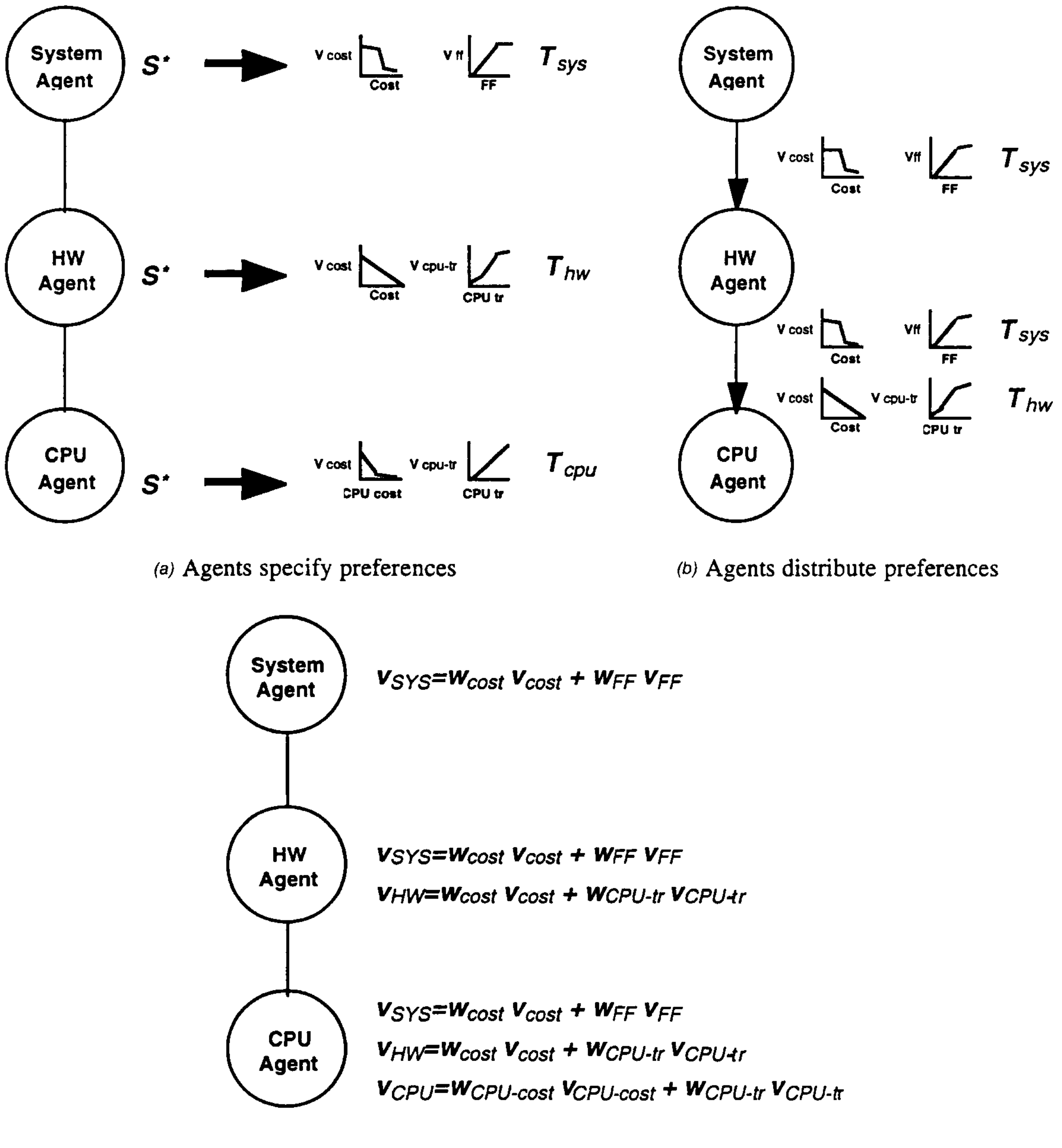

System Agent
HW Agent
CPU Agent
S*
V cost
Cost
V ff
FF
T sys
V cpu-tr
CPU tr
T hw
CPU cost
T cpu
(a) Agents specify preferences
(b) Agents distribute preferences
VSYS=Wcost Vcost + WFF VFF
VHW=Wcost Vcost + WCPU-tr VCPU-tr
VCPU=WCPU-cost VCPU-cost + WCPU-tr VCPU-tr
(c) Resulting value functions

Figure 9. Specification and distribution of preferences.

a set, $T_{SYS} = \{t_1, \ldots, t_z\}$, of two element tuples, $t_z = (S_i, S_j)$, S_i, $S_j \in S^*$, ranking the alternatives in the sample. A ranking of elements in S^* is a partial order represented by the set of two element tuples, where the first element in each tuple, S_i, is at least as preferred as the second element in the tuple, S_j. This ranking is based on the attributes of interest to the system agent, for which the agent also specifies attribute value functions, v_k. 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, a_{cost} and a_{FF}, the hardware agent over cost and CPU throughput, a_{cost} and $x_{CPU}.tr$, and the CPU agent over CPU cost and CPU throughput, $x_{CPU}.cost$ and $x_{CPU}.tr$. In addition to the constraints specified by the system agent, these preferences become part of the problem specification, and therefore, must be followed during the design process.

Figure 9(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 preference down to any agents lower in the hierarchy. Each set of preferences (e.g., v_{cost}, v_{FF}, and T_{SYS} for the system agent) provides all of the information required to construct an imprecise value function (e.g., v_{SYS} for the system agent). Recall that an imprecise value function is defined by specifying attribute value functions (e.g., v_{cost}, v_{FF}) and ordering a sample of alternatives, (e.g., T_{SYS}). The weights (e.g., w_{cost}, w_{FF}, for the system agent) in the imprecise value function are constrained by the ranking and attribute value functions. Each set of preferences implies a value function, resulting in multiple value functions for some agents [see Figure 9(c)].

5.2.5 DECISION MAKING

Identifying the best design is accomplished by first achieving network consistency, then achieving decomposability, and finally identifying a nondominated alternative. Achieving network consistency is identical to the process followed in ACDS, in that all decisions made at this stage are purely related to design feasibility. Once a consistent network is achieved, ACME design agents perform decision making based on both feasibility and value analysis in an attempt to achieve a decomposable network. Analysis results are combined as described at the beginning of Section 5.2, and the appropriate parts are eliminated. In this phase, feasibility analysis is identical to ACDS, but value analysis, as described in the following, is based on the hierarchical preferences that must be obeyed. Finally, agents perform decision making based entirely on value to identify a nondominated alternative. In this last phase, a distributed search algorithm explores possible part combinations, pruning search paths based on dominance checks.

When considering the selection of an alternative based on value, an 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. This process is guaranteed to produce at least one nondominated alternative based on the attribute information that is currently available. If more than one alternative still remains, then one is chosen at random.

For example, to choose a CPU, the CPU agent must first evaluate the change in value, Δv_{SYS}, for various pairs of alternative CPUs. Alternatives that are dominated are eliminated, and the remaining alternatives are the nondominated set based on the system agents preferences. If the agent is unable to identify a single, nondominated alternative, the agent evaluates Δv_{hw} for various pairs of the nondominated alternatives found by Δv_{SYS}. If more than one nondominated alternative still remains, then the CPU agent applies Δv_{CPU} in a similar manner. Note that agent value functions are never directly combined, since in general it is very difficult to ensure that all agents use the same value scale.

6. 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.

To fill the gap in the existing CE research, and thus solve the problems related to hierarchical CE, we discussed a new tool, ACME. Each autonomous ACME design agent has the ability to independently contract for tasks and contract out subtasks. A contract between a supervisor and subordinate includes *both* the constraints and preferences of the supervisor. 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. Agents interact to solve a problem by means of distributed constraint satisfaction, during which search heuristics attempt to identify the best design based on the specified hierarchical preference structure.

At this time, we are completing code development for the ACME system. Value decisions involve the repeated calculation of Δv, which can require extensive communication among agents. We are attempting to reduce these costs by isolating the impact of a change in a design variable assignment to Δv. Initial test results related to isolating the impact of variable changes on value have been encouraging [22]. Isolating this impact may require propagation of intervals through nonmonotonic functions if a wide class of problems is to be supported, and techniques based on the work of Faltings [23] and Hyvönen [24] may prove useful. 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.

References

1. Cutkosky, M. R., et al. 1993. "PACT: an Experiment in Integrating Concurrent Engineering Systems," *IEEE Computer*, pp. 28-37, January.
2. Darr, T. P. and W. P. Birmingham. 1995. "A Case Study in Design Service Integration," in *Proceedings of the Second International Conference on Concurrent Engineering: Research and Applications (CE95)*, pp. 409-415.
3. Davis, D. and R. G. Smith. 1983. "Negotiation as a Metaphor for Distributed Problem Solving," *Artificial Intelligence*, 20:63-109.
4. Dantzig, G. B. and P. Wolfe. 1961. "The Decomposition Algorithm for Linear Programming," *Econometrica*, 9(4).
5. Macko, D. and Y. Y. Haimes. 1978. "Overlapping Coordination of Hierarchical Structures," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-8(10):745-751.
6. Haimes, Y. Y. 1981. "Hierarchical Holographic Modeling," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(9):606-617.
7. Azarm, S. and W.-C. Li. 1988. "A Two-Level Decomposition Method for Design Optimization," *Engineering Optimization*, 13:211-224.

8. Sobieszczanski-Sobieski, J. 1990. "Sensitivity of Complex, Internally Coupled Systems," *AIAA Journal*, 28(1):153-160.
9. Kuokka, D. R. and L. T. Harada. 1995. "Communication Infrastructure for Concurrent Engineering," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 9(4):283-297.
10. Park, H. et al. 1994. "An Agent-Based Approach to Concurrent Cable Harness Design," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 8:45-61.
11. Huyn, P. N., M. R. Genesereth and R. Letsinger. 1993. "Automated Concurrent Engineering in DesignWorld," *IEEE Computer*, pp. 74-76, January.
12. Werkman, K. J. 1992. "Multiple Agent Cooperative Design Evaluation Using Negotiation," *Artificial Intelligence in Design '92*, Kluwer Academic Publishers.
13. Darr, T. P. and W. P. Birmingham. 1994. "Automated Design for Concurrent Engineering," *IEEE Expert*, 9(5):35-42.
14. Darr, T. P. and W. P. Birmingham. 1996. "An Attribute-Space Representation and Algorithm for Concurrent Engineering," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, in press.
15. Huang, G. Q. and J. A. Brandon. 1993. "Agents for Cooperating Expert Systems in Concurrent Engineering Design," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 7(3):145-158.
16. Keeney, R. L. and H. Raiffa. 1976. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, New York: Wiley and Sons.
17. Thurston, D. L. 1991. "A Formal Method for Subjective Design Evaluation with Multiple Attributes," *Research in Engineering Design*, 3:105-122.
18. D'Ambrosio, J. D. and W. P. Birmingham. 1995. "Preference-Directed Design," *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (AI EDAM)*, 9:219-230.
19. 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*, SMC-14(2):223-229.
20. Hu, X., J. G. D'Ambrosio, B. T. Murray and D. Tang. 1994. "Codesign of Architectures for Automotive Powertrain Modules," *IEEE Micro*, 14(4):17-25.
21. D'Ambrosio, J. G. and X. Hu. 1994. "Configuration-Level Hardware/Software Partitioning for Real-Time Embedded Systems," presented at *The Third International Workshop on Hardware-Software Codesign*, September.
22. D'Ambrosio, J. G. and W. P. Birmingham. 1995. "Hierarchical Concurrent Engineering: Supporting Hierarchical Decomposition and Peer-to-Peer Problem Solving," University of Michigan Technical Report CSE-TR-259-95, September.
23. Faltings, B. 1994. "Arc-Consistency for Continuous Variables," *Artificial Intelligence*, 65(2):363-379.
24. Hyvönen, E. 1992."Constraint Reasoning Based on Interval Arithmetic: The Tolerance Propagation Approach," *Artificial Intelligence*, 58:71-112.

William P. Birmingham

William P. Birmingham is an associate professor in the Electrical Engineering and Computer Science Department at the University of Michigan, Ann Arbor. He is also a member of the University's Collaborate for Research on Electronic Work (CREW). Birmingham's research interests are in the area of large, distributed information systems. This includes such areas as distributed optimization and design, concurrent engineering, and digital libraries. Birmingham is the lead system architect on the University of Michigan's Digital Library Project (UMDL), funded by an NSF, ARPA, and NASA joint initiative. Common threads in these areas are coordination mechanisms for a large number of agents, interoperability, and creation and maintenance of large knowledge bases.

Birmingham is an NSF PYI. He is on the editorial board of *Artificial Intelligence and Engineering Design, Analysis, and Manufacture*, and IEEE Expert. He is on the technical program committee of a number of conferences and workshops. He has published over fifty technical papers and one book on digital libraries, design, concurrent engineering, and related topics.

Birmingham has also worked in industry. He spent summers at United Technologies Microelectronics Center in Colorado Springs, CO in 1981 and 1984. He spent a summer at Mentor Graphics Corp. in Portland, OR. He also worked at Carnegie Group in Pittsburgh, PA, building AI-based design tools. He started, with three others, Trimeter Technologies Corp. in Pittsburgh, PA. At Trimeter, Birmingham led the development of several AI-based design tools. He has also consulted with a number of companies over the past few years.

Timothy P. Darr

Timothy P. Darr is a graduate student research assistant in the Electrical Engineering and Computer Science Department at the University of Michigan, where he is a member of the Artificial Intelligence (AI) Laboratory. He is currently pursuing a Ph.D. degree in computer science and engineering. His research interests include concurrent engineering, electronic commerce, distributed design, and autonomous-agent systems. He received a B.S.E. degree in computer engineering in 1988, and an M.S.E. degree in computer science and engineering in 1992, both from the University of Michigan.

Joseph G. D'Ambrosio

Joseph G. D'Ambrosio is a graduate student in the Electrical Engineering and Computer Science Department at the University of Michigan, where he is a member of the Artificial Intelligence (AI) Laboratory. He is currently pursuing a Ph.D. degree in computer science and engineering. He is also a staff research engineer at General Motors Research and Development Center. His research interests include concurrent engineering, agent-based systems, and hardware/software codesign for embedded systems. D'Ambrosio received a B.S.E.E. from Michigan State University and an M.S.E.E. from Purdue University. He is a member of the IEEE, the IEEE Computer Society, and Sigma Xi, and is a licensed professional engineer.