Concurrent Engineering:

An Automated Design-Space Exploration Approach

Timothy P. Darr, William P. Birmingham

Abstract

Concurrent engineering (CE) aims to speed the design process by considering *downstream* concerns, such as reliability and manufacturability, during the design process. Today's CE approaches focus on creating and coordinating product-development teams of human designers. Little attention has been paid to automating the CE process. We present an automated synthesis approach that generates a single space of all possible designs that simultaneously considers all concerns. This approach significantly reduces the time required to create a design.

1. Introduction

Concurrent engineering (CE) is an important area of research. Many industries believe that their survival depends on incorporating CE methodologies. CE has been applied with success to manufacturing industries as diverse as commercial aircraft, construction equipment and computers. CE has been shown to significantly reduce time to introduce new products into the market, improve the quality of designs and reduce the amount of defects once the product gets to market [17] [18].

Much of the current research in CE has focused on the issues involved in allowing groups of workers from a variety of perspectives (design, testability, manufacturability, finance, etc.) to effectively work together in designing some artifact. Creating this diverse group allows a company to bring together personnel as early as possible having a wide range of experiences from design, development, marketing, manufacturing, service, and sales [17]. CE has been developed to deal with many of the problems associated with bringing this geographically and functionally distributed "virtual team" together to eliminate the problems that have arisen in the traditional sequential approach to product development. Issues that have received attention in the CE literature include: collocation and communication [4][16], information sharing [6][8][9], coordination [2][5][11], and developing intelligent agents to support a particular design task or activity [15]. No work to date has addressed the issue of how CE concepts can be applied to automating the actual design process.

This paper presents an Automated Catalog-Design Service (ACDS) which incorporates many of the concepts found in CE to the task of configuration design. In configuration design, an artifact is designed by selecting parts from a catalog. In particular, this paper discusses ACDS's representation and synthesis process. Section 2 is an overview of ACDS. In Section 3, we discuss an interval-based representation scheme. Section 4 presents the synthesis approach and a brief example. Section 5 discusses our results. Section 6 concludes the paper.

2. Overview

ACDS performs configuration design, a type of design in which a set of functions specified by the designer is implemented by selecting components from a library of parts [1][14]. ACDS incorporates the CE principles of concurrency, geographic distribution of knowledge and integration of a variety of design perspectives into the specification of the desired artifact.

To use ACDS, system designers provide a high-level description of the desired design, including the functions to be performed, their interconnections and performance specifications, as shown in Figure 1 (see Appendix).

ACDS is a collection of loosely-coupled, autonomous agents that self-organize based on design specifications. In Figure 2 (see Appendix), ACDS is shown generating the design specified in Figure 1. The network consists of catalog agents, system agents and constraint agents, as described in this section.

2.1 Catalog Agents

Each catalog agent, denoted CA_i, represents a set of physical parts. Catalogs are composed of part models, plus information abstracted from the part models to describe the catalog as a whole (discussed more fully later). A part model includes a set of attributes (e.g., access time), and statements about the set of values associated with the attribute (for example, that a memory's access time is somewhere in the interval [100 120] ns).

ACDS is designed to support thousands of catalog agents, where each agent will represent the product line of a component manufacturer. As such, catalog agents have the freedom to choose whether to participate in a design, so the catalog agents will change from design to design.

2.2 System Agent

The system agent provides a graphical interface that allows the user to specify the design for presentation to the ACDS network. The system agent is responsible for translating the high-level design specifications into the network's representation and broadcasting them to all relevant agents.

The system agent is also responsible for configuring the network to the current design. This is done by identifying which of the catalog agents that are currently participating in the network need to be notified of the current design. When a catalog agent initially joins the network, it sends its internet address to the system agent. The system agent keeps a record of each catalog agent and its location. When a design is initially specified, the system agent sends an invitation to participate in the design to each of the catalog agents. This invitation includes the design constraints and the functions to implement. The catalog agent determines whether or not it can implement the given functions within the specified constraints and makes a decision whether or not to participate in the design. If a catalog agent accepts the invitation, then an acceptance message is sent to the system agent. The system agent also uses the design specifications to create any constraint agents that are needed in the design.

2.3 Constraint Agents

The constraint agents, denoted C_j, maintain consistency throughout the network by enforcing constraints. The constraints are formed over part attributes (e.g., the sum of power consumption of all parts ≤ 15 W), and can be of any arity. Each constraint agent ensures that the evolving design space conforms to the constraint it represents. Thus, downstream concerns (e.g., failure rate) are easily represented as a constraint and are monitored in the same fashion as more typical constraints, such as power dissipation.

In addition to maintaining consistency among constraints, constraint agents direct the pruning of part catalogs to satisfy any violated constraints. As explained in the following sections, the constraint agents evaluate proposed catalogs from the catalog agents and accept new catalogs that move closer toward satisfying the constraints.

3. Representation

ACDS is based on a design-space approach, where the entire design space is reduced through a series of pruning operations until a set of feasible designs results. The design space is formed over all the catalogs participating in a design. The advantage of the design-space approach is that all potential designs are covered during the refinement process, so that iterations through the design space, in general, are not required.

3.1 Intervals

Catalogs are represented as sets of intervals [3][19]. An interval is defined for each design attribute in the catalog. The set of all attributes in a design is given by $A = \{A_1, ..., A_m\}$. For example, attributes for a memory device will include power dissipation, cost, area, and a number of timing parameters, such as CS valid to data out. An individual part's attributes may be intervals. An example catalog representation is shown in Figure 3.

In general, each catalog agent CA_i represents a set of parts $P_i = \{p_{i1}, p_{i2}, ..., p_{ik}\}$, and each part p_{ij} has a value of each of the attributes A, $A_{ji} = \{a_{ji1}, a_{ji2}, ..., a_{jim}\}$. Each catalog agent defines a set of intervals for each design attribute A_j, $[a_{ji\,1}\ a_{jih}]$, where $a_{ji\,1} = \min(A_{ji})$ and $a_{jih} = \max(A_{ji})$. Each constraint agent represents an n-ary constraint for some attribute A_p over all of the catalog agents. These constraints specify that the value of the constraint must lie within some interval $[C_{j\,1}\ C_{jh}]$ and are of the form $\sum_{i=1}^{n}(a_{jih}) <= C_{jh}$ $\sum_{i=1}^{n}(a_{ji1}) >= C_{j\,1}$.

ACDS also has a utility function defined over the set of attributes. The form of the utility function is the following: $U(A) = w_1 * A_1^{-1} + ... w_m * A_m^{-1}$, where the w_i's are a set of attribute weights and $\Sigma w_i = 1$. This utility function is shared by all catalog agents. From this function, each catalog has a set of utilities $U_i = \{u_{i1}, ..., u_{ik}\}$ that correspond to each of its parts and an interval that represents the range of utilities $[u_{i\,1}\ u_{ih}]$, where $u_{i\,1} = \min(U_i)$ and $u_{ih} = \max(U_i)$. These utilities are used to order parts in a catalog; the part with the highest utility value is considered the best part. Figure 4 shows the utility values calculated for our example catalogs.

A = {Cost, Area, Power}
CA_1 = CPU Catalog
P_1 = {80186, 8086, 8086_5}
A_{Cost1}= {20.0, 15.0, 10.0}
A_{Power1} = {3.00, 2.75, 2.50}
A_{Area1} = {1.210. 1.300, 1.367}

Part	Cost ($)	Power (W)	Area (sq. in.)
80186	20.00	3.00	1.210
8086	15.0	2.75	1.300
8086_5	10.00	2.50	1.367
Catalog	[10.0 20.0]	[2.50 3.00]	[1.210 1.367]

CA_2 = Address Decoder Catalog
P_2 = {PAL18P8L, PAL18P8B*, PAL18P8A*}
A_{Cost2} = {3.00, 3.12, 3.02}
A_{Power2} = {0.05, 0.05, 0.05}
A_{Area2} = {0.345, 0.345, 0.345}

Part	Cost ($)	Power (W)	Area (sq. in.)
PAL18P8L	3.00	0.05	0.345
PAL18P8B*	3.12	0.05	0.345
PAL18P8A*	3.02	0.05	0.345
Catalog	[3.00 3.12]	[0.05 0.05]	[0.345 0.345]

Figure 3: An example part catalog.

$$U(Cost, Power, Area) = 0.5 * Cost^{-1} + 0.3 * Power^{-1} + 0.2 * Area^{-1}$$

Part	Cost ($)	Power (W)	Area (sq. in.)	Utility
80186	20.00	3.00	1.210	0.290
8086	15.0	2.75	1.300	0.296
8086_5	10.00	2.50	1.367	0.316
Catalog	[10.0 20.0]	[2.50 3.00]	[1.210 1.367]	[0.290 0.316]

Part	Cost ($)	Power (W)	Area (sq. in.)	Utility
PAL18P8L	3.00	0.05	0.345	6.750
PAL18P8B*	3.12	0.05	0.345	6.740
PAL18P8A*	3.02	0.05	0.345	6.745
Catalog	[3.00 3.12]	[0.05 0.05]	[0.345 0.345]	[6.74 6.75]

Figure 4: An example part catalog with utilities.

The objective of ACDS is to find a set of values for the variables that is consistent with the constraint set, and maximizes the utility function.

In addition to catalog intervals, ACDS has state variables. State variables are not associated with a part, but with the design as a whole. An example state variable is total design cost. State variables are also intervals.

3.2 Dynamic, Distributed Constraint-Satisfaction Problem Formulation

The computational model used by ACDS is the dynamic, distributed constraint satisfaction problem (DDCSP), which is an amalgamation of the distributed constraint-satisfaction problem [20], and the dynamic constraint-satisfaction problem [13].

The DDCSP is represented as a graph, where variables are nodes and constraints are arcs. Each variable has a domain, which is the set of values that the variable can assume. A constraint has a predicate indicating when the constraint is active. Thus, the constraints in a design can change as the design progresses. In our formulation, catalog agents are variables having several domains, each corresponding to an attribute. Constraints are formed over the attributes. A simple DDCSP network is given in Figure 5 for the two catalogs from Figure 3 from the perspective of the power attribute. Note that the constraint is always applicable, as its predicate is True.

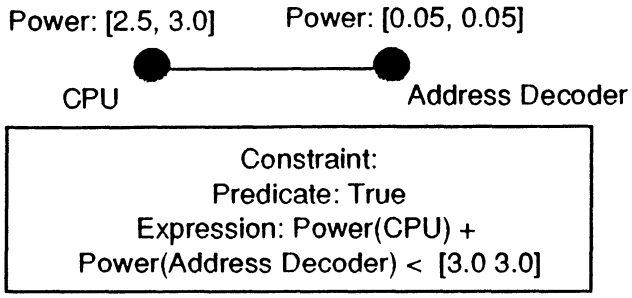

Figure 5: Simple DDCSP network.

As the DDCSP is a derivative of the CSP, all consistency operations are applicable [10][12]. Specifically, ACDS uses node and arc consistency[1] to relax the network. A bidding technique, as described in the next section, is used to generate solutions. The notation introduced in this section is summarized in Table 1.

4. ACDS Behavior

In generating a design, ACDS first initializes the network by sending out invitations to all catalog agents that have joined the network and instantiates the constraints agents. Once all catalog agents have accepted or declined the invitation to participate in the current design, the catalog agents send bids of their catalogs to the constraint agents so that infeasible parts can be removed using a form of arc-consistency. Once infeasible parts have been identified and removed, the catalog agents bid their catalogs to the constraint agents and the constraint agents evaluate the bids to determine if the constraints are violated. If the constraints are violated, then the catalog agents propose new, pruned catalogs to satisfy the violated constraints. Catalogs are pruned by removing parts. The constraint agents evaluate the pruned catalogs and either reject or accept the pruned catalog based on feasibility and utility considerations. Once a pruned catalog is accepted, the part used to form the pruned catalog is removed and the process of removing infeasibles, bidding and pruning continues until all constraints are satisfied or a determination is made that no solution exists. Each step is described in greater detail in the remainder of this section.

[1]These are also know as 0- and 1- consistency, respectively.

4.1 Initialization

ACDS begins by determining which CA_i's to incorporate into the network, as described in Section 2.2. At this point, the network consists of a set of catalog agents CA, each catalog agent CA_i representing some set of parts P_i.

Notation	Meaning
A	Set of all design attributes
A_p	Attribute p
CA	Set of all catalog agents
C	Set of all constraint agents
CA_i	Catalog agent i
C_j	Constraint agent j
C_{j1}	Lower bound on C_j
C_{jh}	Upper bound on C_j
P_i	Set of parts in CA_i
p_{ij}	Part j in catalog CA_i
A_{ji}	Set of attribute values for p_{ij}
a_{jik}	Value of A_j for p_{ik}
a_{ij1}	Minimum value in the set A_{ij}
a_{ijh}	Maximum value in the set A_{ij}
U(A)	Utility Function
w_p	Weight assigned to A_p
U_i	Set of utility values in CA_i
u_{ij}	Utility value for p_{ij}
u_{i1}	Minimum value in the set U_i
u_{ih}	Maximum value in the set U_i

Table 1 Summary of Notation

The system agent instantiates a set of constraint agents C that correspond to each constraint by creating a process for each C_j. Once all constraint agents have reported to the system agent, the system agent notifies each catalog agent of the constraint agents and vice versa.

The system agent next sends messages to each catalog agent, indicating the function(s) that are to be implemented. After the functions are broadcast, the system agent broadcasts the design specifications (constraints) and utility function to each of the catalog agents.

4.2 Pruning Infeasibles

Once the catalog agents have received design parameters from the system agent, they select parts to satisfy their set of functions. Each catalog agent removes any parts that are provably infeasible through arc-consistency. To do this, each CA_i bids an interval for an attribute A_j to the constraint agent C_j. C_j collects the bids, and computes a new interval for each catalog agent that will eliminate any provably infeasible parts. A part p_{ij} from catalog agent CA_i is provably infeasible if there are no parts that can be selected from any of the other catalog agents CA_k, $k \neq i$, such that those parts, in combination with pij satisfy the constraint C_j.

This operation is expressed in the following equation: New interval $CA_i = [a_{ji\,1}{}^{new}$ $a_{ji\,l}{}^{new}]$, where $a_{ji\,1}{}^{new} = \sum_{\substack{k=1 \\ k\neq i}}^{n} a_{jk\,h} - C_{j\,1}$, and $a_{ji\,l}{}^{new} = C_{j\,l} - \sum_{\substack{k=1 \\ k\neq i}}^{n} a_{jk\,1}$.

C_j sends this new interval to the catalog agent CA_i. If the new lower bound $a_{ji\,1}{}^{new}$ is greater than the old lower bound $a_{ji\,1}$, then any part p_{ik} such that $a_{jik} < a_{ji\,1}{}^{new}$ is removed, since there is no way that it can be a part of any solution. Similarly, if the new upper bound $a_{ji\,l}{}^{new}$ is less than the old upper bound $a_{ji\,l}$, then any part p_{ik} such that $a_{jik} > a_{ji\,l}{}^{new}$ is removed. Each catalog agent CA_i now has an interval for each attribute A_j of the form: $[\max(a_{ji\,1}, a_{ji\,1}{}^{new})\ \min(a_{ji\,l}, a_{ji\,l}{}^{new})]$

4.3 Design-Space Formulation

Each CA_i next "bids its catalog", which is the key element of the design-space approach. This means that each catalog agent sends, or bids, its intervals to the corresponding constraint agents for evaluation. The constraint agents collect the bids from all the catalog agents, and determine whether or not a constraint has been violated. If a constraint is violated, then the catalog agents prune their catalog to satisfy the violated constraints. This new catalog is a subset of the original catalog with the property that one or more of the intervals has been relaxed. The process of proposing a new catalog will be described in the next section.

C_j evaluates the bids by creating an interval over the whole design of the form $[\sum_{i=1}^{n}(a_{ji1})$ $\sum_{i=1}^{n}(a_{jih})]$. If $\sum_{i=1}^{n}(a_{ji1}) < C_{j\,1}$ or $\sum_{i=1}^{n}(a_{jih}) > C_{j\,l}$, then the constraint is violated. If a constraint violation is detected, a message is sent to each of the CA_i's indicating whether the violation occurred at the upper or lower bound of the constraint. This step is shown in Figure 6 for the catalogs defined in Figure 3.

Cost <= 23.00
Power <= 3.00
Area <= 2.00

Cost			
CPU [10.0 20.0]	Address Decoder [3.00 3.12]	Cost Interval [13.0 23.12]	Violated
Power			
CPU [2.50 3.00]	Address Decoder [0.05 0.05]	Power Interval [2.55 3.05]	Violated
Area			
CPU [1.210 1.367]	Address Decoder [0.345 0.345]	Area Interval [1.555 1.712]	Satisfied

Figure 6 Catalog Bidding

4.4 Pruning

When a constraint has been violated, each CA_i must prune its catalog to satisfy the violated constraints. Catalogs are pruned by proposing to remove some parts such that the resultant interval will lead to resolving the constraint violation. Since there are likely to be multiple constraints, any of which may be violated, we need some way of assessing the effect that the removal of a part will have on the catalog as a whole. Assuming that an upper-bound violation was detected and that parts with lower attribute values are preferred, the removal of a part that reduces the upper bound on some interval is a good, since it relaxes a constraint and also removes a less preferable part. Conversely, removal of a part that increases the lower bound on some interval is a bad, since it removes a part with the lowest value for some attribute. The argument is analogous if a lower bound violation was detected.

Keeping these rules in mind, we can assign each part a "rating" that equals the marginal utility ($\frac{\partial A_P}{\partial U}$) of removing that part from the catalog. For each part in the catalog, we consider the effect that removing that part would have on the intervals for all the attributes. Ratings are assigned to parts based on the following three cases:

Case 1: p_{ij} is the part with the highest value for some attribute A_p. Let p_{ik} be the part with the next highest value for A_p. Removal of part p_{ij} from the catalog would relax the upper bound on the interval for A_p by the amount $a_{pij} - a_{pik}$. In this case, the rating assigned to pij with respect to Ap is given by $\frac{\partial A_P}{\partial U} = w_p * (a_{pij} - a_{pik})$.

Case 2: p_{ij} is the part with the lowest value for some attribute A_p. Let p_{ik} be the part with the next highest value for A_p. Removal of part p_{ij} from the catalog would increase the lower bound on the interval for A_p by the amount $a_{pik} - a_{pij}$. In this case, the rating assigned to p_{ij} with respect to Ap is given by $\frac{\partial A_P}{\partial U} = - w_p * (a_{pik} - a_{pij})$.

Case 3: p_{ij} is neither the part with the lowest value nor the part with the highest value for some attribute A_p. In this case, the rating assigned to p_{ij} with respect to A_p is $\frac{\partial A_P}{\partial U} = 0$.

The rating for each part p_{ij} is given by $\sum_{A_P \in A} \frac{\partial A_P}{\partial U}$. After all parts have been assigned a rating, the parts are fully ordered. Parts with the highest positive rating are the best parts to remove, since they relax the most intervals. The catalog that will be proposed to the C_j's is the catalog that would result from the removal of the highest rated part. Figure 7 shows the ratings computed for the CPU example catalog.

U(A) = 0.5 * Cost + 0.3 * Power + 0.2 * Area
A = { Cost, Power, Area}

Part	Cost Rating	Power Rating	Area Rating	Total Rating
80186	0.5 * (20.0 -15.0)	0.3 * (3.00 - 2.75)	-0.2 * (1.367 - 1.300)	2.5616
8086	0.0	0.0	0.0	0.0
8086_5	-0.5 * (15.0 - 10.0)	-0.3 * (3.00 -2.75)	0.2 * (1.367 - 1.300)	-2.5616

Figure 7 Part Ratings for CPU Catalog

After determining the best part to remove, each CA_i sends the proposed catalog to the C_j's along with an interval that represents the change in the utility interval U_i that would result if the proposal is accepted. This utility interval is denoted $\Delta U_i = [\Delta u_{i1}\ \Delta u_{ih}]$, where Δu_{i1} is the change in u_{i1} for the proposed catalog and Δu_{ih} is the change in u_{ih} for the proposed catalog.

In evaluating proposals, the goal of the constraint agents is to accept a proposal that possibly contains at least one solution. Let $CP_i = \{CP_{i1}, ..., CP_{ik}\}$ be the set of proposals from catalog agent CA_i. Let $CP = \{CP_i\}$ be a set of proposed catalogs, where $CP_i = [a_{ji1}\ a_{jih}]$ is a proposed interval from catalog agent CA_i for the constraint C_j. Again assuming that an upper bound constraint violation was detected, the set CP can be accepted if the following condition holds for any CP_i in CP: $a_{ji1} + \sum_{\substack{k=1 \\ k \neq i}}^{n} a_{jkh} <= C_{jh}$. The reason that this set can be accepted is that this set of proposals possibly contains at least one solution that satisfies C_j, namely, the part with the lowest value for A_j from CA_i in combination with the parts with the highest value for A_j from all other catalogs CA_k, $k \neq i$. There might be other solutions in this set, and the solution mentioned might violate some other constraint C_g, but these are not relevant to the decision made by C_j. If there is more than one acceptable set of proposals, then the proposal in which $\sum_{i=1}^{n} \Delta u_{ih}$ is the smallest is selected. Figure 8 shows this process for our example catalogs.

Once a set of feasible proposals has been identified, a message is sent to each CA_i indicating the proposal that has been accepted. If none of the proposals is acceptable, then a message is sent to each CA_i requesting a new proposal. If a CA_i cannot propose a new catalog because there are no more parts to remove, then the catalog agent sends a message to the requesting C_j indicating that no proposal can be made.

The C_j's continue to request proposals until a set of feasible catalogs can be identified. Once an identification has been made, a message is sent to the catalog agents indicating which catalog has been accepted. When a proposal is accepted, each CA_i prunes its catalog by removing the part(s) corresponding to the proposal. The process of removing infeasibles, bidding and pruning continues until all constraints are satisfied or it is determined that no solution exists.

CPU proposes removing 80186, producing the new intervals:
Cost [10.0 15.0] Power [2.5 2.75] Area [1.300 1.367]
Change in utility interval: [+0.006 0.0], highest utility part remains

Address Decoder proposes removing PAL18P8B*, producing the new intervals
Cost [3.00 3.02] Power [0.05 0.05] Area [0.345 0.345]
Change in utility interval: [+0.01 0.0], highest utility part remains

Consider accepting Address Decoder proposal only
Cost {[10.0 20.0], [3.00 3.12]} is acceptable
Power {[2.50 3.00], [0.05 0.05]} is unacceptable

Consider accepting CPU proposal only
Cost: {[10.0, 15.0], [3.00. 3.12]} is acceptable
Power: {[2.5 2.75], [0.05, 0.05]} is acceptable t
Area: {[1.300, 1.367], [0.345 0.345]} is acceptable

Consider accepting both CPU and Address Decoder proposals
Cost : {[10.0 15.0], [3.00 3.02]} is acceptable
Power: {[2.5 2.75], [0.05 0.05]} is acceptable
Area: {[1.300 1.367], [0.345 0.345]} is acceptable

Both the CPU and Address Decoder proposals are accepted since they are feasible proposals and neither removes the highest utility part.

Figure 8 Evaluating Proposals

4.5 Solution

Once all constraints are satisfied, each catalog sends its highest utility part to the system agent. The system agent verifies that all functions have been covered and displays the solution. In our example, the CPU catalog would send the part 8086_5 and the Address Decoder catalog would send the part PAL18P8L.

5. Discussion

Our bidding algorithm achieves parallelism in the way that the bidding and pruning is accomplished between the catalog agents and the constraint agents. Bids and proposals are made synchronously to each constraint agent. Each constraint agent collects bids and proposals from the catalog agents and performs their evaluation independently of the other constraint agents. It is not necessary for the constraint agents to communicate among themselves because any dependence between constraints is taken into account when the catalog agents rate their parts and prune their catalogs.

Modeling the problem in this way may result in a loss of optimality. By not allowing the constraint agents to communicate with one another, it is possible that the optimal solution could be overlooked. A network configuration in which constraint agents could communicate would allow the constraints to select the best proposal or set of proposals from the perspective of all constraints, rather than just a single constraint. The cost of such a configuration is an increase in the number of messages and communication paths required to make the decisions.

Our algorithm facilitates CE activities in that it is based on a geographically distributed model of the design space. Any part vendor can create a catalog by supplying information to a knowledge acquisition tool now under development. This tool creates a part model

which includes the necessary intervals for communication with the rest of the network, and any constraints that may be required for correct operation. A user of ACDS can also create constraint agents to represent any downstream design concerns, such as reliability or manufacturability.

The time required to produce a solution is dominated by the sending of messages from catalog agents to constraint agents, the complexity is measured in the number of messages sent; communication time is on the order of seconds, while computation time is on the order of tenths of seconds. The worst case occurs when a constraint agent or set of constraint agents is unable to find a feasible set of proposed catalogs, and must continue to request new proposals from the catalog agents. In this case, each catalog agent would have to send proposals to eventually remove each part. The number of messages in this case would be 2 * max(Number of Parts in any catalog) * Number of constraints, giving a complexity of $\Theta(P*C)$, where P is the maximum number of parts in any catalog and C is the number of constraints. If the network configuration is such that the constraint agents are allowed to communicate with each other, then the number of messages would be 2 * (max(Number of Parts in any catalog) * Number of constraints) * Number of messages between constraint agents. The number of messages between constraint agents to determine the best proposal in this case would be (Number of constraints)^2, so the complexity is $\Theta(P*C^3)$. From this analysis, it is hard to imagine that the benefits of finding the optimal solution is worth the cost of the number of messages required.

6. Summary and Conclusion

ACDS is a fully operation system that uses a distributed-computing paradigm, having created more than 20 designs. Each agent is a separate (UNIX) process, and these processes are distributed throughout the network. At present, ACDS runs on nodes in Ann Arbor, MI, Pittsburgh, PA, and Worcester, MA. Using ACDS, we have created designs consisting of electronic components (CPU, Clock, Address Decoder, etc...) and have also designed an adaptive-process-control model consisting of a milling machine, thermal model, sensor configuration and control and compensation algorithms.

ACDS is based on the notion of shrinking the design space through application of network consistency and bidding algorithms. When performed in a decentralized fashion, the design process is fast, although some degree of optimality is lost. This approach, however, provides a natural framework to support the CE process, where design teams are often in different locations.

Acknowledgments

We would like to thank Dan Atkins, Chip White, Al Ward and Ed Durfee, for their help in formulating many of the ideas in this paper.

7. References

[1] Birmingham W.P., Gupta A.P., Siewiorek D.P., Automating the Design of Computer Systems, Jones and Bartlett Publishers, 1992.

[2] Bowen J., Bahler D., "A Constraint-Based Approach to Networked Colocation in Concurrent Engineering", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering., 1992.

[3] Davis E., "Constraint Propagation with Interval Labels", Artificial Intelligence 32(1987).

[4] Dewan P., "Enabling Technologies for Concurrent Engineering: A Position Statement", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering. 1992

[5] Durfee E., "Coordination Among, and Through, Intelligent Computer Systems", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering, 1992

[7] Finin T., Fritzson R., McKay D, "A Language Protocol to Support Intelligent Agent Interoperability", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering, 1992.

[8] Huhns M., "Integrating Information Semantics for Concurrent Engineering", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering, 1992.

[9] Karinthi R., Jagannathan V., Montan V., Petro J., Raman R., Trapp G., "Integrating Heterogeneous Information Repositories in a Concurrent Engineering Environment", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering, 1992.

[10] Kumar V., "Algorithms for Constraint-Satisfaction Problems: A Survey", AI Magazine, Spring 1992.

[11] Londono F, Cleetus K. J., Nichols D. M., Iyer S., Karandikar H. M., Reddy S. M., Potnis S. M., Massey B., Reddy A., Ganti V., "Managing Chaos: Coordinating a Virtual Team", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering, 1992.

[12] Mackworth A. K., "Consistency in Networks of Relations", Artificial Intelligence 8(1977).

[13] Mittal S., Falkenhainer B., "Dynamic Constraint Satisfaction Problems", Proc. AAAI 1990.

[14] Mittal S., Frayman F., "COSSACK: A Constraints-Based Expert System for Configuration Tasks", Proceedings 2nd Int'l Conf. on Applications of AI to Engineering, 1987.

[15] Pan J., Tenenbaum J., "An Intelligent Agent Framework for Enterprise Integration", IEEE Transactions on Systems, Man and Cybernetics, November/December 1991.

[16] Rangan P., Vin H., "A Unified Framework for Modeling Synchronous and Asynchronous Multimedia Collaborations", Proceedings First Workshop on Enabling Technologies for Concurrent Engineering. 1992

[17] Shina S., "New rules for world-class companies", IEEE Spectrum, July 1991.

[18] Turino J., "Making it work calls for input from everyone", IEEE Spectrum, July 1991.

[19] Ward A., Lozono-Perez T., Seering W., 1990, "Extending the Constraint Propagation of Intervals", AI EDAM 1990 4(1).

[20] Yokoo M., Durfee E., Ishida T., Kuwabara K., "Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving", Dept. of EECS, University of Michigan Technical Report No. CSE-TR-102-91 1991.

Appendix
Figure 1: High Level Design Description

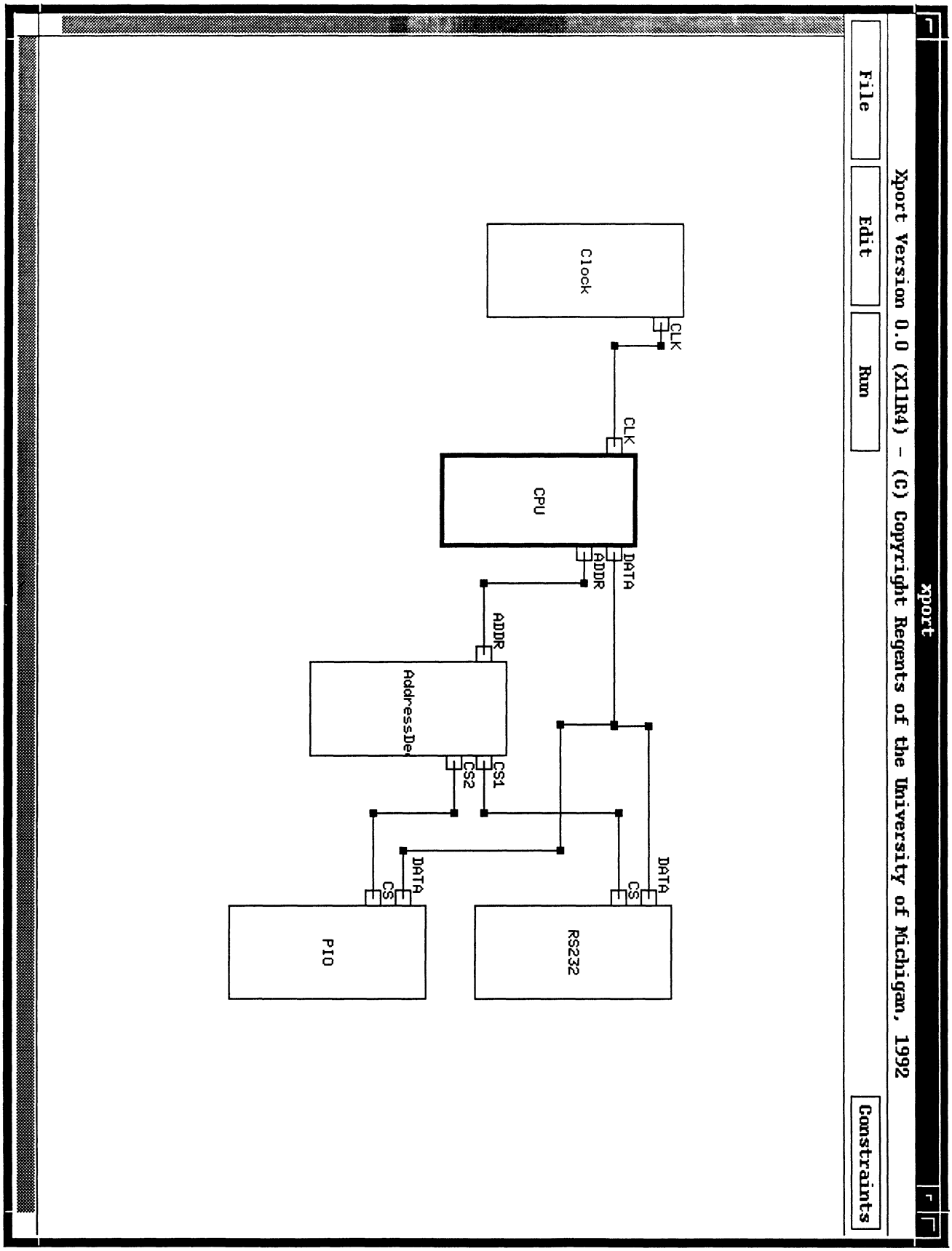

Figure 2: Design Generation