The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems

H. Van Dyke Parunak (ERIM)[1], Allen Ward (Ward Synthesis), John Sauter (ERIM)

Abstract

MarCon (Market-based Constraints) applies market-based control to distributed constraint problems. It offers a new approach to distributing constraint problems that avoids challenges faced by current approaches in some problem domains, and it provides a systematic method for applying markets to a wide array of problems. Constraint agents interact with one another via the variable agents in which they have a common interest, using expressions of their preferences over sets of assignments. Each variable integrates this information from the constraints interested in it and provides feedback that enables the constraints to shrink their sets of assignments until they converge on a solution. MarCon originated in a system for supporting human product designers, and its mechanisms are particularly useful for applications integrating human and machine intelligence to explore implicit constraints. MarCon has been tested in the domain of mechanical design, in which its set-narrowing process is particularly useful.

1. Introduction

MarCon (Market-based Constraints) bridges two important areas of research in multi-agent systems: distributed constraint optimization and market-based programming.

The classic constraint satisfaction problem (CSP) seeks assignments to a set of variables $X = \{x_1, x_2, \ldots, x_m\}$ from a set of corresponding domains, one per variable, $D = \{d_1, d_2, \ldots, d_m\}$, that satisfy a set $C = \{c_1, c_2, \ldots, c_n\}$ of relations over subsets of the Cartesian space spanned by D. CSP is a binary problem. A set of assignments to the variables X either satisfies the constraints or it does not. Many applications permit varying degrees of satisfaction, leading to the constraint optimization problem (COP). CSP and COP can be distributed across computational agents in a number of ways, leading to DCSP and DCOP, as discussed further in Section 7.

Distribution of a single computation across multiple agents raises problems of coordination. In natural systems, agents often manage distributed constraints by sensing the gradients of dissipative fields [17]. MarCon applies the dissipative forces of a marketplace to address such ill-structured constraint problems.

There is considerable interest in applying market techniques to engineered environments [5] (e.g., manufacturing scheduling [13] and product design [25]). Until now the design of a marketplace to support a particular problem has been more of an art than a science. MarCon offers a systematic way to apply markets to a large subset of constraint optimization problems. It builds on the mapping established by [25] between market entities (consumers, producers with technologies, goods) and the components of a distributed constraint problem (agents, constraints, variables). In MarCon, individual constraint agents (humans in the current implementation) buy

[1] Correspondence to first author at vparunak@erim.org, CEC, ERIM, 2901 Hubbard Rd., Ann Arbor, MI 48105-2467 USA, +1 734-769-4049 (voice), +1 734-213-3405 (fax)

and sell assignments to variables among themselves, and market dynamics drive variable agents to assignments that reflect the utilities of the various constraints.

Section 2 describes the kind of problems for which MarCon is particularly suited, and shows how MarCon is configured for a specific problem. Section 3 outlines the behaviors of the constraint and variable agents as the network runs. Section 4 defines qualitative market representations for both ordered and unordered variable domains that support these agent interactions. Section 5 offers a formal analysis of how local myopic decisions yield a globally coherent solution. Section 6 reports experiments with the system. Section 7 discusses the relations of MarCon and RAPPID with other research, and Section 8 summarizes the significance of the approach.

2. Problem Set-Up and Execution

Given an appropriate problem, MarCon identifies variable agents, constraint agents, and distinguished subsets of constraint agents that ground the network's utility and cost functions, and then defines which constraint agents buy and sell which variable agents.

2.1. Problem Characteristics

MarCon generalizes techniques developed in the RAPPID project for distributed component-centered design [18-21]. Some characteristics of this domain are *desirable* for the application of MarCon, and if they are lacking, one might prefer alternative distributed constraint mechanisms. Other characteristics reflect *features* of MarCon over other approaches.

Realized Constraints (Desirable).–In design, a system component constitutes an implicit constraint among a subset of the design variables. MarCon is particularly appropriate for domains in which a constraint is not just an abstract bundle of mathematical relations, but a naturally bounded chunk of reality. Many conventional constraint problems (such as 3-SAT) are formulated at a level of abstraction far removed from underlying domain specifics. MarCon is most naturally mapped to entities in the underlying real-world problems.

Causality (Desirable).–The components in a design that are interested in the assignments to any given variable (say, a motor and a transmission interested on the torque on the shaft between them) can be divided into those that causally affect the assignment (the motor) and those that do not (the transmission). When causality is present, MarCon uses it in defining which constraints initially sell a given variable and which ones buy it. This notion of causality is intuitive rather than formal. It helps human observers and participants understand the markets more easily.

Implicit Constraints (Feature).–Many design and planning domains do not provide explicit analytic forms for constraints, and gathering empirical utility data may be expensive and time-consuming. MarCon does not require explicit advance knowledge of the form of a constraint or the associated utility values. MarCon can begin with only the knowledge of which variables are involved in a constraint and very approximate knowledge of their individual utilities. During operation, it discovers regions of the search space where more detailed exploration is worth the time and expense.

Sparse Connectivity (Desirable).–Most variables in design are of interest to only a small subset of the components, and each component is interested in only a small subset of the variables. This sparseness limits the communication necessary through the constraint network, and permits

effective distribution of work among agents. MarCon has less of an advantage in constraint problems that do not exhibit this sparseness, such as bin packing.

Set-Based (Feature).–Most designers consider a series of point solutions, and must periodically backtrack to escape dead-ends, making convergence difficult. MarCon is set based. Variables' domains shrink monotonically over time from their initial full extents, collapsing on the solution to the problem. The disciplined convergence thus guaranteed is more valuable in many industrial contexts than the theoretical optimum that, in some contrived problems, may occasionally evade it.

Carbon-Silicon Hybrids (Feature).–Much of the creative work of design must be done by humans. Thus the network of agents that MarCon supports includes both carbon (human) and silicon (computerized) agents. It is easier to integrate human participation into a MarCon-based system than into many fully automated constraint-based mechanisms.

2.2. Kinds of Agents

A MarCon configuration is a network of alternating variables and constraints (Figure 1). Each variable and each constraint is a separate agent. A constraint agent is *interested in* the assignments to those variable agents to which it is adjacent. Constraint agents bid among themselves to set the assignments of variable agents in which they are interested. Where there is no danger of confusion, we use "constraint" for "constraint agent" and "variable" for "variable agent." Potential assignments to a variable need not be numerical or even ordered.

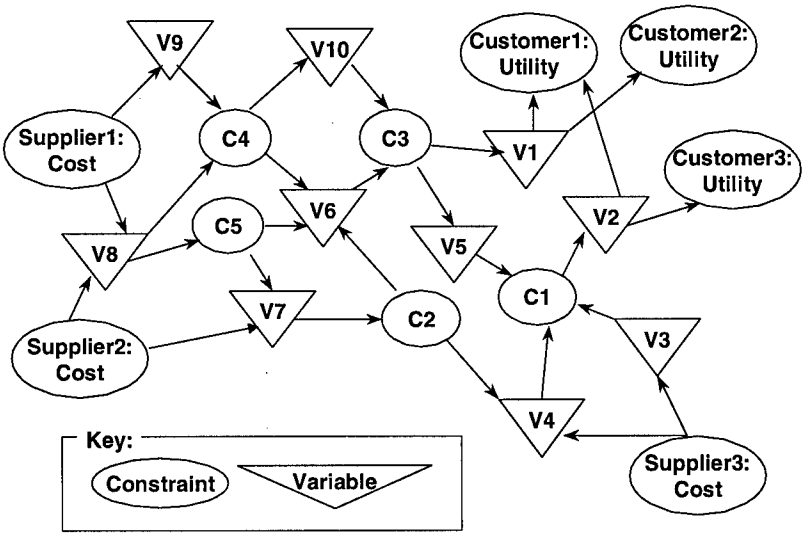

Figure 1: An Example MarCon Network

Each constraint is a function from some subset of the variables to a utility. Given two sets of assignments to the variables of interest to a constraint, the constraint prefers the set yielding the higher utility. The dependency of a constraint's utility on its variables may be represented either explicitly (by closed analytic form or enumeration) or implicitly (to be discovered through search, or in the expertise of a human decision-maker). In design, most constraint agents are humans, while most variable agents are computerized. A MarCon constraint estimates its utility

as the difference between what it receives for variables that it sells and what it must pay for variables that it buys.

A fundamental question in market-based programming is how to define the initial allocation of resources. MarCon's answer is to ground both utilities and costs in distinguished constraints, known as "customers" and "suppliers," respectively.

One or more customers have an interest in the overall outcome of the constraint optimization problem. The distinguishing feature of a customer is that its utility does not depend on the utility that other agents in the system place on variables to which the customer can make assignments. Thus the customers ground the utility of the network. If there is only one customer, that agent possesses a global utility function for the network, but the possibility of multiple customers means that the system does not require such a global utility function. Even if there is only a single customer, in a sparse network the market structure divides the task of estimating the global utility among several other constraints.

One or more suppliers can cause some or all of the variables to assume specific assignments. Each supplier owns (either explicitly or implicitly) a cost function over the possible values of the variables to which it can make assignments. The distinguishing feature of a supplier is that this cost function does not depend on the costs that other agents in the system attach to variables on which the supplier may depend. Thus the suppliers ground the costs in the network.

As a MarCon network operates, utility flows from customers toward suppliers, and cost flows from suppliers toward customers. Each constraint seeks to maximize the difference between the utility it offers to its buyers and the cost it incurs from its sellers.

2.3. Identifying Buyers and Sellers

The MarCon network (Figure 1) must meet certain conditions. In many domains, a causal heuristic helps identify configurations that meet these conditions.

The roles of agents as buyers or sellers for particular variables are fairly fluid, subject to the following restrictions:

- Each variable must be of interest to at least two constraints, at least one that is a potential buyer and at least one that is a potential seller. (Variables of interest only to a single constraint are internal to that constraint and hidden from the market.)
- Each constraint must be either a buyer or a seller for each variable that it constrains. This role may change over time.
- The initial buying and selling roles must be assigned so that if arrows are drawn from seller to buyer as in Figure 1, every variable lies on a directed path from a supplier to a customer, ensuring that each variable can be influenced both by the costs grounded in suppliers and by the utilities grounded in customers.

It is intuitively helpful (but not necessary) to identify causal links between some system elements and the assignments to some variables. For example, a motor and a transmission both have an interest in the RPM and torque on the shaft between them, but the motor causes those assignments, while the transmission only constrains them. In this case, the motor sells torque and RPM, while the transmission buys assignments to these variables. Once clearly causal links are identified, a stochastic search via percolation or spreading activation, beginning with customers

and suppliers and spreading into the network, can identify paths and label buyers and sellers as needed to permit flow along those paths.

3. Agent Behaviors

Constraint agents offer to buy or sell ranges of assignments to variables of interest to them, by posting bids with the variables. Each variable compares the bids it receives from its constraints and recommends how those constraints should shrink their ranges in converging toward a solution.

3.1. What do Constraint Agents do?

A constraint expresses its interest in a variable by telling that variable:

1. Whether it wants to buy or sell the variable;
2. A range of assignments to the variable that it can accept;
3. A range of prices that it would be willing to pay (as buyer) or accept (as seller) for different assignments in this range;
4. A description of the function of price over range of assignments;
5. Whether its offer is good for "any" or only for "one" of the possible assignments to the variable.

Items 2 and 3 define a region of price/assignment space. The constraint that is issuing a bid indicates how its price varies with assignment (item 4). In some domains (for example, catalog-based design), this price function is known in great detail. In others, MarCon supports qualitative indications of price shapes (as in Table 1).

Table 1: Price/Assignment Shape Indicators

/	Price increases with assignment
\	Price decreases as assignment increases
^	Price is maximum between extreme assignments
v	Price is minimum between extreme assignments
-	Price is relatively flat
~	Price/assignment relation is unspecified

Ordinal variables such as weight or torque, whose potential assignments can be ordered, support price/assignment ranges and qualitative price shapes. Another class of variables ("nominal variables") does not support a natural ordering of potential assignments (e.g., the material out of which a product should be made). Shape indicators are not meaningful for such variables. MarCon uses directed acyclic graphs (DAG's) to record a constraint's preferences for nominal variables. Section 4 discusses the definition and use of both price shapes and preference DAG's.

In Section 5, we show that if constraints compute their prices according to particular rules, their local behavior yields a global maximum of utility over cost. The essential point of the rules is that two underlying heuristics yield global coherence:

- In buying, each agent passes on as much of the utility it receives as it can and still break even, based on the buy bids it is offered.

- In selling, each agent demands the lowest price that lets it break even, based on the sell bids it is offered.

In some cases a constraint can accept any assignment within its bid range, for a price within the range of prices it originally quoted. In other cases, the ranges reflect the bidding constraint's uncertainty as to exactly what assignments it can accept and what the associated prices are, and it can guarantee its price range only if it makes the final choice of assignment. These two classes of bids are called "any of" and "one of," respectively. All constraints begin bidding in "one-of" mode, and switch to "any-of" as soon as ranges have narrowed sufficiently that they can make a meaningful bid on "any-of" terms. A constraint may decline subsets of its bid proposed by its associates on transactions in which it is in "one-of" mode, but once it moves to "any-of," it must accommodate their requests.

3.2. What do Variable Agents Do?

A variable agent maintains a market in the assignments to that variable. Its basic functions guide its constraints in narrowing the ranges of their bids.

Recommend Trimming.–Because the variable knows the assignment ranges, price ranges, and shapes of the bids submitted by interested constraints, it can recommend how those constraints should trim their assignment ranges, thus narrowing in on an assignment that is maximally beneficial to all of them. MarCon defines two trimming heuristics, one for ordinal variables and another for variables with no natural ordering, as discussed in Section 4 below.

Enforce Protocol.–The variable knows the "one-of/any-of" status of all interested constraints. It enforces the rule that a constraint can move from "one-of" to "any-of" but not back again. In addition, an "any-of" bidder can accept a more drastic change in its bid range than can a "one-of" bidder, and the variable uses its knowledge of each bidder's status to tune bidding recommendations.

Recommend Direct Negotiation.–Sometimes the variable's heuristics and algorithms cannot make progress. For example, price or assignment ranges in buy bids may not overlap those in sell bids, or bid shapes may not support strong recommendations by the variable. In these cases, the variable recommends that the constraints interact directly with one another to resolve the stalemate. For automated constraint agents, this interaction might take the form of a richer negotiation protocol. In our hybrid carbon/silicon environment, it consists of a direct communication between human designers.

Other Services.–In addition to its basic function of guiding constraint agents in narrowing their bid ranges, a variable agent provides a number of other services that cannot be discussed in detail for lack of space.

- At any moment in the process, a given constraint is considering a number of different variables and their impact on the satisfaction of the constraint. Each variable can compute various "figures of merit" that its constraints can use to compare the state of problem-solving with respect to the different variables and select the most important one to consider next. For example, one variable may offer a much wider price range than another.
- Sometimes a group of variables needs to be traded together. For example, in a design problem, the torque and RPM of a shaft must both be purchased from the same source, the

supplier of the (single) shaft. The associated variables can be grouped together to help the constraints deal with them as a unit.

- Some variables are of interest to multiple buyers, multiple sellers, or both. MarCon defines methods to integrate the bids from these multiple participants.

3.3. Souq vs. Exchange

Participants in natural markets can communicate costs and utilities in two ways. Markets with many participants can close frequently on the basis of simple, direct bids, as in a stock exchange or commodities exchange. When there are fewer participants, they may engage in extended negotiation (as in a middle eastern *souq*) before reaching a deal. The current MarCon protocols follow the latter pattern. They do not require an actual flow of currency, just the propagation of a cost field (grounded in the suppliers) and a utility field (grounded in the customers) through the network of designers. Thus there is no need for an initial allocation of currency. Constraints communicate over sets of possible assignments, and their bids change as the various assignment ranges shrink. The design is complete when all assignments have converged. If money were to change hands, it would do so at this point. Changes in the net worth of individual designers as the result of such "closing dynamics" may help organize the behavior of a design team across multiple design projects, but we have not exploited it in our current scenarios.

4. Qualitative Representations

Both variables and constraints depend on representations of how prices vary over a set of alternative assignments to variables. MarCon uses two representations: price shapes for ordinal variables (those over which a natural total ordering is defined) and price DAG's for nominal variables (those over which there is no natural ordering).

4.1. Qualitative Trading of Ordinal Variables

Where quantitative details are not available, a constraint communicates its utility (as a buyer) or its cost (as a seller) for an ordinal variable in terms of a range of possible assignments to the variable, a range of prices over that range of assignments, and a curve shape indicator (Table 1) that estimates the qualitative relation between the two. The variable agent computes the intersection of the assignment ranges of interest to the constraints. Then, using simple linear models of the curve shapes, it estimates the region in which the buyer's price exceeds the seller's, and recommends that the participating constraints trim their assignment ranges to maximize that difference.

For example, one constraint bids to buy a given variable with assignment range 2-7, price range 5-20, and curve shape '/', and another constraint offers to sell the same variable with assignment range 5-8, price range 5-25, and curve shape '^'. Figure 2 illustrates graphically the range of

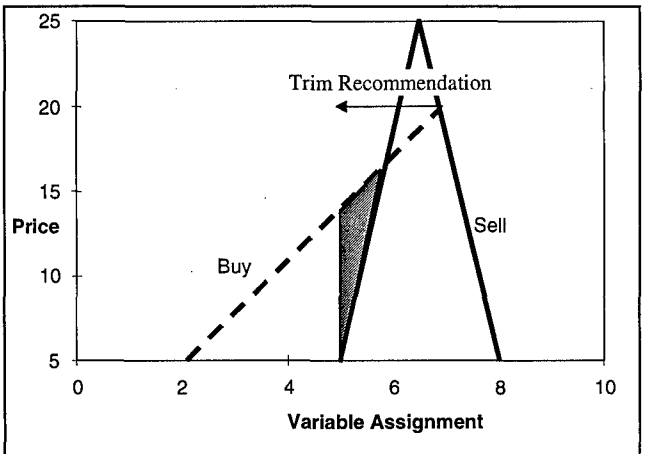

Figure 2: Resolving Qualitative Bids

assignments of common interest to the constraints (5-7), simple linear models of utility and cost over this range, and the shaded subregion over which utility is likely to exceed cost (5-6). The variable recommends that the participating constraints trim their ranges from the top of this subregion.

This mechanism is heuristic. One can contrive bid configurations in which the participants' price or assignment ranges do not intersect, or where (as in Figure 3) no single trim recommendation can be made. In these cases, the variable recommends direct negotiation between the constraints. Furthermore, linear stereotypes of bid shapes only approximate the actual (and often yet unknown) shape of the price/assignment dependency. In practice, the approximations enable constraints (and in our domain, the designers responsible for them) to narrow in on the most promising regions of the search space.

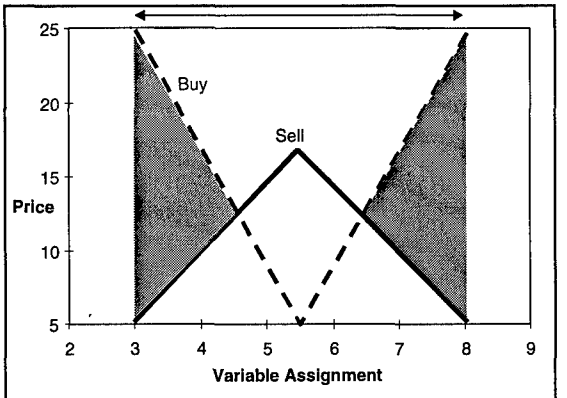

Figure 3: A configuration that does not support automatic trimming

4.2. Qualitative Trading of Nominal Variables

For nominal variables, the lack of a natural ordering precludes the use of a price curve. An alternative data structure, the price DAG, enables nominal variables to make trimming recommendations and identify the need for direct interaction.

4.2.1. Price DAG's

Each constraint agent interested in a nominal variable generates a DAG that reflects its preferences, and specifies an overall price range. Each edge in this "price DAG" originates at the alternative with lower price and terminates at the alternative with higher price. The constraint agent need not establish a preference between every possible pair of possible assignments, but can begin by specifying only those preferences that it clearly knows initially. In the example in Figure 4, the constraint (a buyer) has the highest utility for Wood (not more than $500) and the lowest for Ceramic (not less than $300). It prefers Wood to either of Aluminum or Steel, and both Aluminum and Steel to Ceramic, but has not determined the relative benefit between Aluminum and Steel, or between Plastic and any of the other materials.

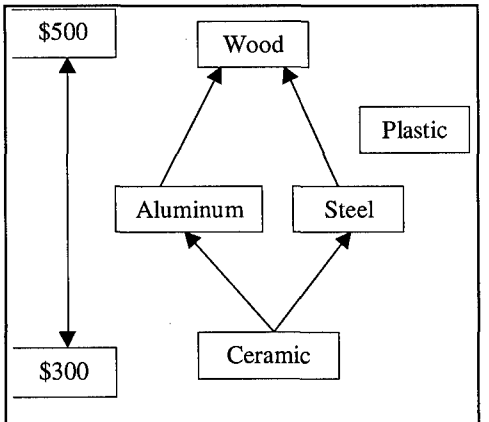

Figure 4: A Buy DAG

A Constraint Agent offering Material for sale constructs a similar DAG (Figure 5).

Computationally, price DAGs are represented as adjacency matrices. Table 2 shows the adjacency matrix ***B*** for the buy DAG in the example. The sell adjacency matrix is ***S***.

4.2.2. Support for Trimming

A nominal variable uses the price DAG's to recommend which possible assignments may be excluded from further consideration, by executing the following steps.

1. Close each adjacency matrix by summing all its powers up to the fixed point and replacing non-zero entries with 1. The resulting *dominance matrix* shows which possible assignments dominate which other ones either directly or indirectly. B^* and S^* are the buy and sell dominance matrices, respectively.
2. Transpose S^* to invert the sell DAG, putting low prices at the top and high prices at the bottom. If two alternatives are in the same order in B^* and $(S^*)^T$, the one with higher utility also has lower cost, and is thus clearly preferred to the alternative.
3. The non-zero cells in $J = AND(B^*, (S^*)^T)$, where the logical *AND* is cell-by-cell, identify orderings that are true of both buyer and seller, and define a joint DAG (Figure 6) that reflects the excess of utility over cost. On the basis of this particular DAG, the variable recommends that its constraints determine their relative preferences for Plastic and Wood, and trim from the bottom of the joint DAG.

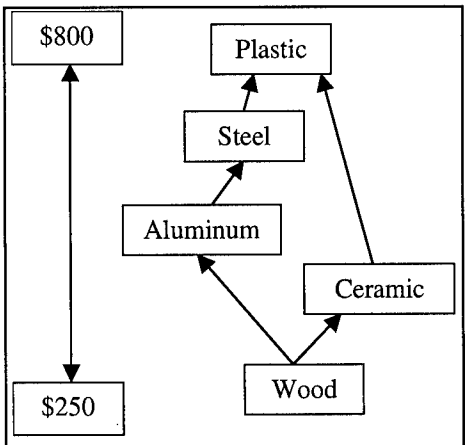

Figure 5: A Sell DAG

Table 2: Adjacency Matrix *B* for Buy DAG

	Aluminum	Ceramic	Plastic	Steel	Wood
Aluminum	0	1	0	0	0
Ceramic	0	0	0	0	0
Plastic	0	0	0	0	0
Steel	0	1	0	0	0
Wood	1	0	0	1	0

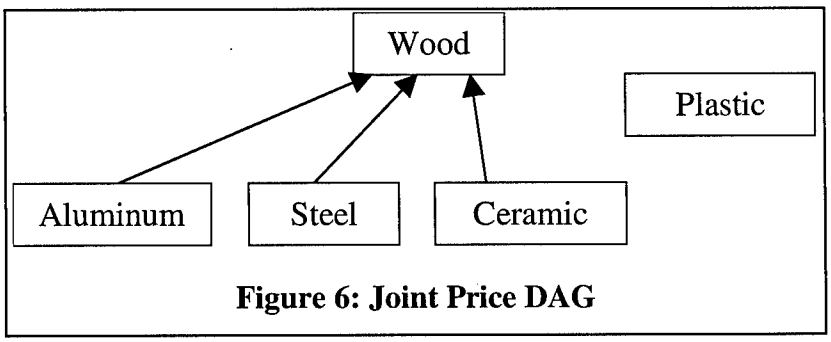

Figure 6: Joint Price DAG

4.2.3. Support for Direct Interaction

Price DAG's also enable nominal variables to identify a non-overlap condition and provide information to the constraints that may help them close the gap. A non-overlap can be detected either because the price ranges specified by the constraints do not overlap, or because the sets of alternatives included in their price DAG's are disjoint.

Some alternatives may not be directly comparable in the price DAG from one or another of the negotiating constraint agents. These cases are the non-zero cells in $NOT(AND(OR(S^*, (S^*)^T), B^*))$. When one constraint provides a preference and another does not, this mismatch indicates to the second constraint which preferences it should explore further to advance the overall relaxation.

The constraints may have opposing preferences for different alternatives. For example, the buyer may assign a higher utility to wood than to plastic, while the seller assigns a higher cost to wood than to plastic. Thus the high utility alternative is not the low-cost alternative, and the ordering information in a joint DAG is not sufficient to identify the alternative with the highest utility in excess of cost. ***AND(B*, S*)*** identifies these cases.

5. Emergence of Global Coherence from Local Decisions

In making their individual decisions, constraint agents seek to maximize the difference between the utility they realize and the costs they incur, thus providing overall system value of the sum of individual maxima. The acceptability of the overall solution depends on maximizing the difference between the sum of the utilities of individual agents and the sum of their costs, a computation that in general requires central selection of the individual agent choices. This section shows that subject to appropriate structural restrictions and with specified rules for agent behaviors, the two approaches give the same result. It also discusses some limitations of the current formal analysis, which emphasize the importance of the experimental results in Section 6.

5.1. Basic Definitions and Assumptions

We consider the situation when all bids have collapsed to points, so that only the base packages have prices associated with them, and those prices are single points. Assume that the network forms a tree, with a single customer, with constraints numbered depth first as shown in Figure 7. The customer is number 0, and each variable has only one buyer. (We relax this tree assumption below.) For the purposes of this analysis, the variables serve only to establish the connectivity of the constraints, and are not in focus in the remainder of this discussion. Thus we will speak of one constraint selling to (buying from) another, when strictly we should say that one sells (buys) a variable that another buys (sells).

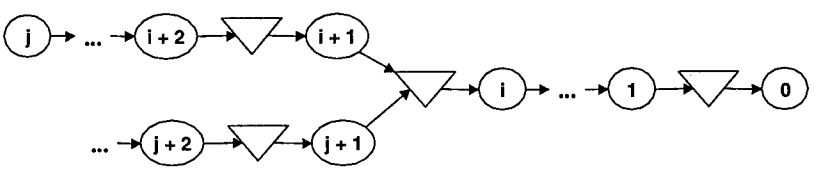

Figure 7: Numbering the Constraints

Table 3 shows how we map our market vocabulary onto standard tree terminology. The tree terms "descendant" and "ancestor" are used in their usual senses as iterations of "child" and "parent," respectively.

A node with two or more children is a *branch point.*

Table 3: Tree and MarCon Terminology

Tree Terminology	MarCon Vocabulary
Node	Constraint
Root	Customer
Leaf	Supplier
Node A is a child of node B	Constraint A sells a variable that constraint B buys (loosely, Constraint A sells to constraint B)
Node A is a parent of child B	Constraint A buys a variable that constraint B sells (loosely, Constraint A buys from constraint B)

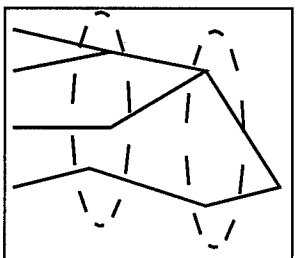

Figure 8: Cuts

We will refer to any set of contiguously numbered nodes that does not include both a branch-point and its child as a *limb.* A *cut* (Figure 8) is a set of nodes such that any path between the root and any leaf includes only and only one member of the cut.

C_i is the internal costs experienced by constraint i (that is, costs experienced by the constraint but not visible as purchases in the network being modeled), and U_i its internal utility, so that its net internal cost is C_i - U_i. All of a supplier's costs, and all of a customer's utilities, are internal. The total cost $C = \sum_{i>0}(C_i - U_i)$. (This and other indexed operations are over i > 0, excluding the customer, which permits sales to be attributed to every indexed node.) Let S be price at which constraint 1 sells the system to the customer (constraint 0). Then, system profit $P = S - C = S - \sum(C_i - U_i)$. At the system level, we want to maximize this difference.

Let $B_{i,o}$ be the sum of the prices of offers to buy make by the ith constraint (the o standing for "offer" or "out".) Let $B_{i,r}$ be the sum of the prices of the received bids to sell to constraint i, that is, the total amount that constraint i would pay if it bought at the prices offered to it. Similarly, let $S_{i,o}$ be the sum of the prices of constraint i's offers to sell, and $S_{i,r}$ be the sum of the prices offered to constraint i by its buyer. (Even with the tree restriction, there may be more than one item being traded between a node and its unique parent.) Clearly, if node i and node $(i+1)$ are members of the same limb, then $S_{i+1,o} = B_{i,r}$, and $B_{i,o} = S_{i+1,r}$. If D is the set of immediate children of node i, $B_{i,o} = \sum_{j \in D}(S_{j,r})$ and $B_{i,r} = \sum_{j \in D}(S_{j,o})$.

We impose the following bidding rules on the constraints:

Rule 1: $B_{i,o} \leftarrow S_{i,r} - C_i + U_i$

Rule 2: $S_{i,o} \leftarrow B_{i,r} + C_i - U_i$.

That is, in formulating buy bids, each constraint passes on as much as it can of the money it receives and still break even, based on the buy bids it is offered. In formulating sell bids, it sells for the lowest price it can and still break even, based on the sell bids it is offered.

Finally, we define value added for constraint i to be $V_i = S_{i,r} - (B_{i,r} + C_i - U_i)$. We want to determine the relationship between the sum of the value added across all constraints, and the system profit.

5.2. The Basic Result

First we need:

Lemma 1: $V_i = S_{i,r} - S_{i,o} = B_{i,o} - B_{i,r}$.

Proof: The equalities are obtained by substituting Rule 1 and Rule 2 into the definition of value added. □

Lemma 2: $S_{i,o} = \sum_j (C_j - U_j)$, for j a descendent of i (including i).

Proof: Clearly this is true for leaf nodes (suppliers). Suppose it is true for all the descendants of node i. Then, node i arrives at $S_{i,o}$ by adding its internal costs and subtracting its internal utilities, so it is true for i. □

Now, we can state:

Theorem 1: The sum of the values added for the nodes in any cut is equal to the system profit.

Proof (for the special case of trees): This is clearly true for the cut consisting only of constraint 1, since for this node (by Lemma 2) the offering sales price is the sum of the costs, and the received sales price is what the customer will pay. Now suppose it is true for a given cut. Move the cut "closer to the leaves" by replacing one member, say j, of the cut with its child or children. If there is only one child, the value added is the same, by Lemma 1. If there are multiple children, let us call them the set D. Then, $\sum_{j \in D} V_j = \sum_{j \in D} (S_{j,r} - S_{j,o}) = B_{i,o} - B_{i,r} = V_i$. □

5.3. Relaxing the Tree Restriction

In real constraint problems, it is often the case that the constraints do not fall naturally into a tree. We relax our tree assumption in two steps. First, we convert the tree into a semi-lattice by allowing any constraint to sell to another constraint ($i > 0$) iff the second constraint is not "farther from the customer." That is, there can be no loops in the digraph, but all other connections are allowed. Later, we will add loops.

5.3.1. Allowing a Semilattice

The semi-lattice has a unique least upper bound (LUB, node 1), but will normally not have a lower bound. The restriction $i > 0$ implies that the customer buys only from constraint 1, and is imposed so that $S_{i,r}$ and $S_{i,o}$ are defined uniformly for all nodes at or below the LUB.

Converting to a semi-lattice has two effects. First, there may be nodes that cannot be part of any cut, for example node 9 in Figure 9, because any cut that severs the paths through the direct arc from 10 to 8 must also sever every path through 9. A similar situation arises when multiple leaf nodes differ in the number of steps that separate them from their LUB. We correct such situations by introducing "virtual nodes," such as 9a in Figure 10. Virtual nodes have no costs associated with them, but otherwise behave like real nodes; their only function is ensure that every node can be part of a cut, and therefore that the idea of "moving a cut one step" is well defined.

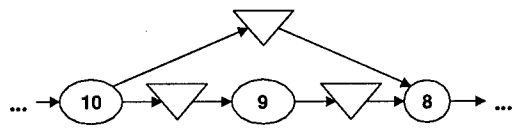

Figure 9: A Semi-Lattice with No Cut

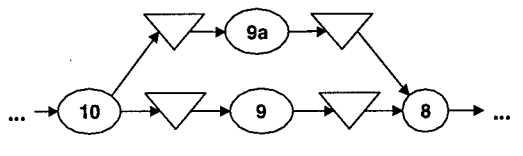

Figure 10: Inserting a Virtual Node

The other problem is that Lemma 2 is no longer valid, since not all the costs of the descendents (or inferiors, to switch to lattice terminology)

need be represented in a given ancestor's sales price. However, we do have a slightly more complex equivalent:

Lemma 3: For any cut E with collective descendants F, $E \subset F$, $\sum_{j \in E} S_{j,o} = \sum_{k \in F} (C_k - U_k)$.

Proof: This is clearly true for the cut consisting of all the leaf nodes. Now, assume it is true for a given cut. Move that cut one step toward node 1. This move may replace a single node with a single ancestor, a single node with several ancestral nodes, or several nodes with a single ancestor, the ancestors being in each case immediate. Let the replaced nodes be G and the new nodes H. In each case, $\sum_{j \in H} S_{j,o} = \sum_{k \in G} S_{k,o} + \sum_{j \in H} (C_j - U_j)$. Hence, the statement will remain true, and is true of every cut. □

In particular, Lemma 3 is true of the cut consisting of node 1. Hence, we have a starting point for reproving Theorem 1, and we can move the cut toward the leaves stepwise as in Lemma 3 to reprove Theorem 1 for the case of lattices.

5.3.2. Allowing Loops

Now, consider what happens when we add loops: a buyer (in our design domain, often but not necessarily node 1) sells to a node farther from node 0 than itself. A simple insight shows that this case is isomorphic to the previous one (the semi-lattice). That is: a sale from A to B at price P is the same as a sale from B to A at a price –P. From a structural perspective, any structure with loops can be reduced to a semi-lattice by permitting some sales to have negative prices, and the proof schema used for trees and semi-lattices can then be applied.

5.4. Practical Implications

The formal development outlined in the previous sections has two limitations, both based on the observation that the critical variables in the discussion ($B_{i,o}$, $B_{i,r}$, $S_{i,o}$, $S_{i,r}$, C_i, and U_i) are functions over the set of variables in which the agent has an interest, evaluated at the point when the agents' ranges on variables have collapsed.

First, while we have proven that the prices will yield a system optimum when the ranges collapse, we have not proven that the ranges can in fact collapse with Rules 1 and 2 in force. That is, we have not shown that it will always be possible for constraints to bid consistently with these rules. Formal exploration of the dynamical behavior of the algorithm in general and its convergence properties in particular remains for further work. In practice, we have found that the system does converge on the examples to which we have applied it.

Second, agents do not know in advance the point to which their variable ranges will collapse, and so cannot apply Rules 1 and 2 exactly. In describing the behavior of constraint agents in Section 3, we summarized two heuristics that approximate these rules:

1. In buying, each agent passes on as much of the utility it receives as it can and still break even, based on the buy bids it is offered.
2. In selling, each agent demands the lowest price that lets it break even, based on the sell bids it is offered.

Again, our response is an empirical one: these approximations to the formally verified rules yield reasonable results in the problems to which we have applied them.

6. Experimental Results

One motivation for MarCon is the need to support a mix of human and artificial agents. To date, the system has only been implemented in such settings, with human constraint agents and computerized variable agents. The wide range of decision-making algorithms used by humans in such a setting makes systematic experimentation difficult and reduces the significance of comparison with fully automated constraint optimization methods. One of our experiments does illustrate the potential of the approach. A team of five designers designed a power transmission mechanism, a problem that is often used as a class assignment in university design curricula. Human agents represent the electric motor that provides rotational power (constrains 7 of the 23 system variables), the transmission that reduces that power to the customer's requirements (10 variables), a box to hold the motor and transmission (11 variables), the customer's demands for the overall system (7 variables), and overall system weight (4 variables). Twenty-three computerized markets represent the system variables.

Figure 11 illustrates the convergence of one system variable (horsepower between the motor and the transmission) during the course of the experiment. The motor is able to narrow its initial sell bid without an offer to buy from the transmission, on the basis of bids in other markets. The transmission's offer to buy cuts the range to about a third of its original size. The subsequent sell bids by the motor and the final collapse to a single assignment appear unilateral, but in fact are in response to buy bids from the transmission in other related markets (RPM and torque). The graph shows the convergence that MarCon supports, and the lack of alternation between buying and selling in this particular market emphasizes the interaction of different variables that the constraints couple together.

The motor and transmission designers select from predefined catalogs of components with 23 and 22 alternatives, respectively, while the box designer selects one of three materials and defines each of the box's three dimensions as a real number. Thus, even ignoring the real-valued box dimensions, the design space includes 23*22*3 = 1518 possible independent choices. In a brute-force distributed implementation, one might broadcast all of these possible configurations to the design team for review and comment, resulting in a total communicative load on the order of (1518 configurations + 5*1518 comments ≈ 9000) messages and a need for each agent to process 5*1518 ≈ 7500 messages (the original configuration and the responses of the other four agents). If we assume something smarter than brute-force distribution, these estimates would decrease, but if we include the real-valued variables of box dimension and weight, they would greatly increase, so they provide a useful rough benchmark.

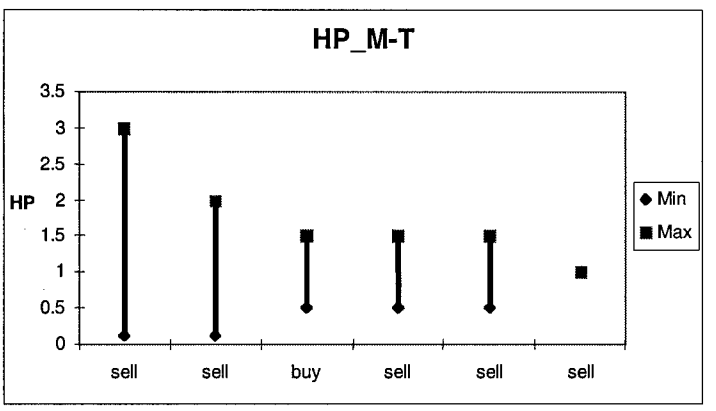

Figure 11: A sample history of MarCon bids

The team converged on a solution after posting only 145 separate bids. In this example, all bids were between only two agents. Thus MarCon's communication requirements are on the order of 2% of the benchmark, and each agent has to process only an average of 2*145/5 = 58 messages, or less than 1% of the benchmark. Although we do not claim that MarCon is formally optimal, in this case subsequent exhaustive search of the design space showed that the team did in fact converge to the global optimum. What is perhaps most important in our domain is that MarCon realizes these efficiencies while supporting implicit constraints and set-based reasoning in a hybrid carbon-silicon community as discussed in Section 2.1 above.

<<<Note to editor and reviewers: On Oct. 23, a group of graduate students in UM's NAME dept. will be applying RAPPID to a more complex design problem, an exercise in ship design. This experiment is being designed to yield further quantitative information on the convergence properties of MarCon, and this section can easily be extended to incorporate those results when they are available.>>>

7. Relation to Other Research

MarCon bears comparison with other research in three areas: agent-based design support, distributed constraint optimization, and market-based programming.

7.1. Agent-Based Design Support

MarCon itself is not a design system, but it arose in the context of the RAPPID system for distributed design support, and design continues to be a natural domain to which to apply MarCon. The comparisons with other research in this section are in terms of the parent RAPPID system.

It is convenient to summarize research in agent-based design environments by how each system decomposes the problem into agents. Five approaches are common: agents may be mapped onto features, parts, designers, design tools, or design functions. We first review these categories, and then compare RAPPID with the systems that are most similar to it to make clear the contributions it offers. The projects classified here are representative rather than exclusive, but do include the projects most directly comparable with RAPPID.

The vision of systems instantiating *features* (such as a hole or a wall) as agents is that a designer can identify the features required on a part, and those features will negotiate among themselves to discover a mutual organization that satisfies functional and manufacturing constraints. Such systems (e.g., [3, 9]) attempt to apply agent technologies to the long tradition of research into feature-based design (e.g., [16]).

Systems that represent individual *parts* as agents (e.g., [8, 25, 26] recognize the growing popularity of the Internet as a medium over which product vendors offer their goods, and develop mechanisms to help designers quickly locate and evaluate components in on-line catalogs that will satisfy their requirements. ActiveCat [26] focuses on attaching semantic information and simulation models to catalog entries, so that designers can "try before they buy." ACME [8] propagates constraints among individual catalogs (each represent a set of alternative parts) and among different hierarchical levels of product structure. Both catalogs and constraints are agents in this system. The preferences of ACME become an actual market model in [25], in which the various components bid among themselves for the assignments they can make to their attributes.

Many design problems cannot be solved by configuring pre-existing parts from catalogs, but require designers to create the necessary components and their interfaces. In this situation, it is natural to support the individual (human) *designers* or design teams with (computerized) agents. The decomposition is similar to the part-based approach, since each designer or design team is typically responsible for a specific component, but the inferencing task is more difficult. Instead of checking the consistency among existing members of a discrete set of components, the system must guide designers in creating new components, whose attributes may be drawn from an infinite set (e.g., rational numbers) or even a non-ordered set (e.g., material). This decomposition is the one addressed by RAPPID, which uses a set-narrowing heuristic to avoid conflicts. [10, 11] also supports designers as agents, using conflict resolution heuristics to deal with conflicts after they arise.

In many problem domains, the design process is dominated by complex *tools* (such as finite-element codes for mechanical design, or circuit emulators for microprocessor design). These tools are typically configured for stand-alone use, and coordinating several of them on a multi-disciplinary project can be challenging. Systems such as PACT [7], SBD [6], and the blackboard support for RRM [12] treat design tools as the fundamental agents and use multi-agent techniques to coordinate their use.

The philosophy of *functional* decomposition, common in traditional monolithic software design, survives in some agent architectures for design. DICETalk [22] recursively divides design tasks into subtasks and assigns them to general problem-solving agents with access to task-specific knowledge bases. SiFA [4] is a variety of the A-Team architecture [24], with different species of agents for such functions as selecting a possible assignment to a design parameter, evaluating an assignment, identifying conflicts among assignments, and recommending resolutions to conflicts.

RAPPID is motivated by the challenge of distributing design across physically and organizationally separated design teams, each responsible for some component or subsystem of the overall product. This requirement strongly recommends an architecture that decomposes the problem into agents representing designers or the parts that they design. Each agent can then be local to a specific organization, managing that organization's communications with the outside world and encapsulating the high bandwidth communications within the organization. This decomposition means that RAPPID bears most immediate comparison with the part-centric and designer-centric systems identified above. Systems focused on features, design tools, and design functions are largely orthogonal to RAPPID's objectives.

RAPPID's set-based dynamics offer a fundamental contrast to Klein's approach [10, 11]. Klein sees conflict as central to design, and so provides mechanisms primarily for the resolution of conflicts after they arise. RAPPID recognizes conflicts as a symptom of a defective design philosophy, one that jumps prematurely to point solutions. Its mechanisms focus on ways for designers to balance preferences before they make conflicting decisions, rather than resolving conflicting desires afterwards. RAPPID recognizes that conflicts are sometimes unavoidable, and helps its participants identify them so that they can resolve them through SLOWh mechanisms (which might in some cases include systems embodying Klein's mechanisms). However, it devotes its major effort to avoiding such conflicts in the first place.

Among the part-based systems, ActiveCat [26] is complementary to RAPPID rather than competitive with it. When a designer in RAPPID is dealing with off-the-shelf parts, ActiveCat is a natural mechanism for storing and consulting the specifications of those parts, and could

provide much of the functionality that RAPPID prototypes in its spreadsheet interface. In turn, RAPPID's market mechanisms have no counterpart in ActiveCat. The preference functions of ACME [8] and the market formalization of [25] bear much more similarity to RAPPID. However, these systems are not set-based, but presume that the components of the design are defined in advance, a luxury that is not available in many industrial design scenarios. They are both fully automated systems that require formal models of components and subsystems, another requirement that is difficult to realize in many industries. Where RAPPID treats a component as the instantiation of a constraint among design attributes, ACME distinguishes constraints from components. Thus RAPPID can begin work as soon as the components and their attributes are enumerated and attributes of interest to more than one component are identified. ACME requires an additional level of knowledge engineering to specify the constraints among components. This task often requires solving the most vexing design problems, and so asks the system analyst to do much of the designers' work in advance.

7.2. Distributed Constraint Optimization

Early work in distributed constraint management focuses on distributed CSP, and assigns subsets of the variables to each agent [23, 28]. Each agent monitors all the constraints in which its variables are involved. The assumption that variables are the most natural mapping for agents persists as recently as [1, 27]. In many domains it makes sense to assign computational ability to constraints rather than to variables, an approach taken in [14], where subsets of related constraints are assigned agenthood. In both approaches, each agent represents one or more variables (or alternatively, constraints), uses traditional centralized techniques among those elements, and coordinates with other agents to identify inconsistencies and backtrack as necessary.

Liu and Sycara [15] extend the application of constraint agents from constraint satisfaction to constraint optimization. They classify constraints as either hard and soft and relax soft constraints as necessary to maximize some overall objective function. Monitoring a global objective function is difficult in a distributed system, but in some problems the constraints vary widely in the degree to which they impact the value of that function. In such domains, high *disparity metrics* identify anchor constraints whose local costs are a reasonable estimator of the global costs. A showcase example is shop floor scheduling, in which utilization levels at bottleneck workstations have much higher impact on manufacturing costs than do utilization levels at other workstations. The agent representing an anchor constraint takes a leadership role among other agents interested in the same variables, and uses its local costs to guide its decisions (and thus theirs as well). Armstrong and Durfee [1] are one example of recent work with variable agents (in this case, in DCSP) that similarly seeks ways to establish the relative priority of the agents to guide the search process.

In an approach inspired by naturally occurring ecologies with multiple interacting species of agents [17], MarCon elevates both variables and constraints to the status of autonomous agents. Many distributed problems offer neither natural priorities among agents, easy access to a global objective function, nor distinguished anchor constraints. MarCon copes with such situations by using a marketplace to propagate information asynchronously and omnidirectionally among agents.

7.3. Market-Based Programming

Most work in market-based programming, including Wellman's pioneering work on configuration-based design [25], is based on an economic model defined by the nineteenth-century economist Leon Walras. In Walrasian economies, a centralized auctioneer collects bids from the participants, computes the price at which global supply equals global demand, and then reports this price to the participants. In such an economy, no trades take place until the auctioneer derives an equilibrium price. For economies with multiple commodities, Walras defined an incremental mechanism, *tatonnement*, in which the auctioneer moves from one commodity to another, adjusting the price so that supply equals demand. Because the price of one commodity may affect the demand for others, the auctioneer typically must cycle through the list of commodities several times before prices converge.

The Walrasian model is analytically tractable, and so has been the subject of considerable theoretical study. However, it has several disadvantages. Its centralization is contrary to the distributed spirit of agent-based architectures and the practical requirements of many modern business environments. It implies universal information about prices and demands, which is unrealistic. Furthermore, in an economy involving multiple commodities, computation of a Walrasian equilibrium is NP-complete.

These shortcomings have led to a renewed interest in an alternative economic model known as "bilateral exchange" or (after its nineteenth-century economic proponent) "Edgeworth barter" [2]. In this approach, individual pairs of agents meet, and trade if they can find a price at which both participants improve their individual utility. There is no need for a central auctioneer. Agents do not require global information, and trade can take place before prices have stabilized globally. As agents engage in trade with different partners, price information propagates through the community. While bilateral exchange has less of an analytical literature than Walrasian auction, Lyapunov analysis shows the existence and stability of convergence, and under manageable assumptions the result is Pareto optimal. Both the equilibrium location and the final distribution of wealth over agents depend on the actual trajectory followed by the system, but the convergence can be achieved in time linear in the number of agents and quadratic in the number of commodities.

MarCon was not developed explicitly from either of these models, but shares intuitions from each of them, and sometimes deviates from them both. As in a Walrasian model, constraint agents do not trade pairwise, but through a variable agent, who may be viewed as an auctioneer. However, this auctioneer manages only a single commodity, and so cannot engage in *tatonnement*. Instead, it seeks convergence by guiding the incremental shrinkage of price ranges, a dynamic not explicit in either Walrasian or Edgeworth models. As in pairwise exchange, constraint agents can make local decisions without waiting for the system to reach a global equilibrium. MarCon's deferral of any actual transfer of funds until the system converges has a decidedly Walrasian flavor, but in fact by the time such a transfer takes place, the market mechanisms have already served their purpose of coordinating the actions of the different agents. In MarCon the market is used more for its side effects than for the actual exchange of currency for commodity (its main function in both Edgeworth and Walrasian models). Over a time horizon longer than a single problem, for instance among design teams involved in multiple projects, the movement of currency among constraint agents may very well have an important integrating effect. But our analysis and examples give us no information on this dynamic.

8. Conclusion

MarCon advances both market-oriented programming and constraint programming.

- It offers a systematic way to apply market mechanisms to a wide class of constraint problems. Thus it moves the application of these mechanisms from a research problem toward an engineering discipline.
- It provides a truly distributed way to solve constraint problems, and one that does not require *a priori* knowledge of anchor constraints that dominate the problem.

MarCon exploits the physical structure of a problem, including realized constraints and causal pressures. It is particularly attractive in domains that require close integration between carbon and silicon intelligence, and in applications where evaluation of the utilities faced by individual constraints is costly and time-consuming and must be justified on a step-by-step basis as problem-solving progresses.

Acknowledgements

Portions of this research were funded by the DARPA RaDEO program under contract # F33615-96-C-5611, under Program Manager Kevin Lyons and COTR James Poindexter. The RAPPID project team includes the authors, Mike Davis, Bob Matthews, Mitch Fleischer, and Mike Wellman. MarCon is the subject of US and International patents pending.

References

[1] A. Armstrong and E. Durfee. Dynamic Prioritization of Complex Agents in Distributed Constraint Satisfaction Problems. In *Proceedings of Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 620-625, 1997.

[2] R. Axtell and J. Epstein. Distributed Computation of Economic Equilibria via Bilateral Exchange. Brookings Institution, Washington, DC, 1997.

[3] S. Balasubramanian and D. H. Norrie. A Multiagent Architecture for Concurrent Design, Process Planning, Routing, and Scheduling. *Concurrent Engineering: Research and Applications*, 4(1):7-16, 1996.

[4] I. Berker and D. C. Brown. Conflicts and Negotiation in Single Function Agent Based Design Systems. *Concurrent Engineering: Research and Applications*, 4(1):17-33, 1996.

[5] S. H. Clearwater, Editor. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. Singapore, World Scientific, 1996.

[6] A. Cox, E. Bouchard, and D. Drew. SBD System Design. *Concurrent Engineering: Research and Applications*, 4(1):35-46, 1996.

[7] M. R. Cutkosky, R. S. Englemore, R. E. Fikes, T. R. Gruber, M. R. Genesereth, W. S. Mark, J. M. Tenenbaum, and J. C. Weber. PACT: An Experiment in Integrating Concurrent Engineering Systems. *IEEE Computer*, 26 (January)(1):28-37, 1993.

[8] J. D'Ambrosio, T. Darr, and W. Birmingham. Hierarchical Concurrent Engineering in a Multiagent Framework. *Concurrent Engineering: Research and Applications*, 4(1):47-57, 1996.

[9] D. Jacquel and J. C. Salmon. Solving the 3D Localization Problem for Feature Agents. In *Proceedings of The 9TH SYMPOSIUM of the International Federation of Automatic Control on INformation COntrol in Manufacturing systems (INCOM'98)*, 1998.

[10] M. Klein. Supporting Conflict Management in Cooperative Design Teams. In *Proceedings of 11th Inter. Workshop on DAI*, pages 155-81, 1992.

[11] M. Klein. Supporting Conflict Management in Cooperative Design Teams. *Journal on Group Decision and Negotiation*, 2:259-278, 1993.

[12] S. E. Lander, D. D. Corkill, and S. M. Staley. Designing Integrated Engineering Environments: Blackboard-Based Integration of Design and Analysis Tools. *Concurrent Engineering: Research and Applications*, 4(1):59-71, 1996.

[13] G. Y.-J. Lin. *A Distributed Production Control for Intelligent Manufacturing Systems.* Ph.D. thesis, Purdue University, School of Industrial Engineering, 1993.

[14] J. Liu and K. Sycara. Emergent Constraint Satisfaction Through Multi-Agent Coordinated Interaction. In *Proceedings of Reaction to Cognition: 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'93 Neuchatel, Switzerland*, pages 107-121, Springer, 1995.

[15] J. Liu and K. Sycara. Exploiting Problem Structure for Distributed Constraint Optimization. In *Proceedings of First International Conference on Multi-Agent Systems*, pages 246-253, AAAI, 1995.

[16] D. Navinchandra. *Exploration and Innovation in Design: Towards a Computational Model.* New York, NY, Springer, 1991.

[17] H. V. D. Parunak. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997.

[18] H. V. D. Parunak. RAPPID Project Index Page. http://www.iti.org/cec/rappid/, 1997.

[19] H. V. D. Parunak, A. Ward, M. Fleischer, J. Sauter, and T.-C. Chang. Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization. In *Proceedings of AAAI Workshop on Constraints and Agents*, pages 93-99, American Association for Artificial Intelligence, 1997.

[20] H. V. D. Parunak, A. C. Ward, M. Fleischer, and J. A. Sauter. The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research. *Autonomous Agents and Multi-Agent Systems*, Forthcoming, 1998.

[21] H. V. D. Parunak, A. C. Ward, and J. A. Sauter. A Systematic Market Approach to Distributed Constraint Problems. In *Proceedings of International Conference on Multi-Agent Systems (ICMAS'98)*, American Association for Artificial Intelligence, 1998.

[22] M. Sobolewski. Multiagent Knowledge-Based Environment for Concurrent Engineering Applications. *Concurrent Engineering: Research and Applications*, 4(1):89-97, 1996.

[23] K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed Constrained Heuristic Search. *IEEE Trans. Systems, Man, and Cybernetics*, 21(6 (Nov/Dev)):1446-1461, 1991.

[24] S. N. Talukdar and P. S. de Souza. Scale Efficient Organizations. In *Proceedings of IEEE International Conference on Systems Science and Cybernetics*, pages 1458-1463, IEEE, 1992.

[25] M. P. Wellman. A Computational Market Model for Distributed Configuration Design. *AI-EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 9(2):125-134, 1995. Available at ftp://ftp.eecs.umich.edu/people/wellman/aiedam95.ps.Z.

[26] P. Will. Active Catalogs. 1997.

[27] M. Yokoo. Distributed Constraint Scheduling: Foundation and Applications. In *Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 14-15, IEEE Computer Society, 1998.

[28] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed Constraint Satisfaction for Formalizing Distributed Problem Solving. In *Proceedings of 12th IEEE International Conference on Distributed Computing Systems*, pages 614-621, 1992.

INTERNET DOCUMENT INFORMATION FORM

A . Report Title: The MarCon Algorithm: A Systematic Market Approach to Distributed Constraint Problems

B. DATE Report Downloaded From the Internet 4/05/99

C. Report's Point of Contact: (Name, Organization, Address, Office Symbol, & Ph #): **Industrial Technology Institute**
P.O. Box 1485
Ann Arbor, MI 48106

D. Currently Applicable Classification Level: Unclassified

E. Distribution Statement A: Approved for Public Release

F. The foregoing information was compiled and provided by:
DTIC-OCA, Initials: VM_ **Preparation Date**: 4/05/99__

The foregoing information should exactly correspond to the Title, Report Number, and the Date on the accompanying report document. If there are mismatches, or other questions, contact the above OCA Representative for resolution.