Distributed Component-Centered Design as Agent-Based Distributed Constraint Optimization

Van Parunak†, Al Ward‡, Mitch Fleischer†, John Sauter†, and Tzyy-Chuh Chang‡

†Industrial Technology Institute
PO Box 1485
Ann Arbor, MI 48106
{van, john, mitch}@iti.org

‡Ward Synthesis
3446 Gettysburg Rd.
Ann Arbor, MI 48105
{award@, shuh@alumni.}engin.umich.edu

Abstract

Many manufactured systems (both consumer goods and manufacturing systems) consist of a number of discrete subsystems and components that interact with one another through various interfaces to provide the required functionality. Designing such a system requires finding values for interface variables that are compatible among the various components, and is analogous to the constraint optimization problem, with subsystems and components playing the role of constraints among the variables. Today's business environment requires design to be done by distributed teams of engineers, so the analogy can be extended to distributed constraint optimization (DCOP). This paper develops the parallel between distributed component-centered design (DCCD) and DCOP, discusses the particular flavor that industrial requirements impart to the mapping, and reports how this parallel is being exploited in the RAPPID system for agent-based distributed design.

1. Introduction

Constraint satisfaction has been an extraordinarily fertile paradigm in Artificial Intelligence, providing a common vocabulary and structure for a wide range of problems. The basic problem structure [Mackworth 1992] has three components: a set of variables $X = \{x_1, x_2, ..., x_m\}$, a set of corresponding domains, one per variable, $D = \{d_1, d_2, ..., d_m\}$, and a set of constraints $C = \{c_1, c_2, ..., c_n\}$, each a relation (in some refinements, a function) over a subset of the Cartesian space spanned by D. The problem is to assign values to the variables that satisfy the constraints. Section 2 describes how previous research has extended this basic pattern to distributed domains.

This paper grows out of the RAPPID[1] project, which is developing an agent-based environment to support the collaboration of a distributed team of designers, each responsible for a different component or subsystem of the entire product. RAPPID represents each component and subsystem by an agent, which negotiates with other components in a marketplace over assignments to shared variables. This form of design organization is increasingly common as manufacturers of end products purchase more and more components of their products from suppliers, and expect those suppliers to participate in the design of the subsystems that they supply and thus of the overall product. For example, the Big Three US automotive manufacturers purchase many of their seating systems from suppliers, who in turn acquire the necessary components and subassemblies from supply chains made up of a dozen or more different companies [Hoy 1996, Fleischer & Liker 1997]. Section 3 draws a parallel between the distributed design of a component-centered artifact (Distributed Component-Centered Design, or DCCP) and DCOP. Section 4 highlights some important distinctions between DCCP and current DCOP technology, and discusses how RAPPID is modifying the DCOP paradigm to support these distinctions. Section 5 summarizes current and upcoming activities in RAPPID that exploit and extend the application of DCOP to DCCD.

[1] Responsible Agents for Product-Process Integrated Design, (www.iti.org/cec/rappid).

2. A Brief History of Distributed Constraint Satisfaction and Optimization

The classic constraint satisfaction problem (CSP) seeks assignments to a set of variables $X = \{x_1, x_2, ..., x_m\}$ from a set of corresponding domains, one per variable, $D = \{d_1, d_2, ..., d_m\}$, that satisfy a set $C = \{c_1, c_2, ..., c_n\}$ 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 are not Boolean, but permit varying degrees of satisfaction, leading to the constraint optimization problem (COP), which may be modeled in different ways. Sometimes the constraints are partitioned into hard and soft categories, and soft categories are relaxed until an overall figure of merit is maximized. In another approach, the constraints are non-Boolean functions over the space spanned by D and the problem is to maximize a weighted sum of their values.

For a number of years the distributed artificial intelligence (DAI) community has explored ways to distribute constraint problems. One fundamental question in such an effort is how to partition the domain among the agents. In general, one wants this partitioning to reflect

the natural structure of the domain, so that each agent can perform significant amounts of work autonomously, without the need for excessive coordination with other agents. The natural options for distributing the constraint model are to represent the variables X, the constraints C, or both, as agents. Early work focused on distributed constraint satisfaction, and modeled subsets of X as agents [Sycara et al. 1991], [Yokoo et al. 1992], each of which must monitor all the constraints in which it is involved. In many domains it makes more sense to assign computational ability to constraints rather than to variables, an approach taken in [Liu & Sycara 1995a], where subsets of related constraints are assigned agenthood.

This early work focused on distributed constraint satisfaction (DCSP). [Liu & Sycara 1995b] extend the application of constraint agents from constraint satisfaction to constraint optimization (DCOP). 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 greatly in the degree to which they impact the value of that function. Such domains are characterized by high disparity metrics, which can be used to identify anchor constraints whose local costs are a reasonable estimator of the global costs. A showcase example of such a domain 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).

3. How is DCCD Like DCOP?

Component-centered products depend for their effective functioning on the interactions of their components and subsystems. These interactions are typically dominated by different design variables.[2] For example, Figure 1 shows some components in a simple power transmission system and some of the variables through which they interact. If design for such a system were being done across a supply chain, the end-product manufacturer would be represented by an agent at the "System" node, while the other three nodes would be occupied by the suppliers responsible for boxes, motors, and transmissions, respectively. RAPPID provides mechanisms to enable these agents to communicate effectively about the constraints they impose on shared variables.

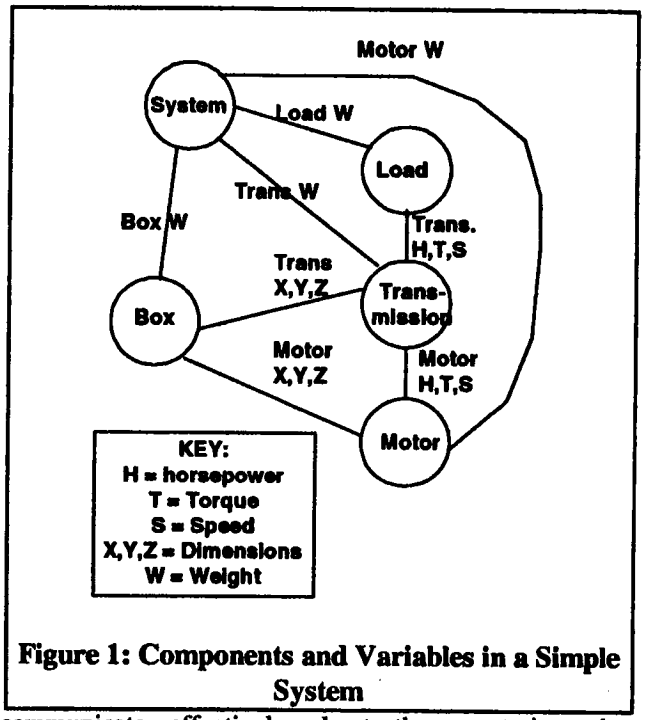

Figure 1: Components and Variables in a Simple System

The fundamental insight behind RAPPID's application of DCOP to design is that the agent responsible for a component or subsystem of a complex mechanism may be viewed *per se* as a constraint (or set of related constraints) among the variables associated with that component. For example, the Motor designer in our simple system constrains the dimension variables of the space that it needs from the Box, the horsepower, torque, and rotational speed it provides to the Transmission, and the weight it contributes to the complete system, and it is the locus that determines how changes in any of these variables affect the others. Sometimes the constraints embodied in an agent are explicit (as when the various permitted sets of alternatives for size, weight, and power are catalogued). In other cases they are implicit in the designer's expertise and professional experience. As in the classical DCOP formulation, these constraints are linked into a network by shared variables. In our case, for instance, the Box designer must provide enough space for the motor and the transmission, and so is just as much concerned with the dimensions of the Motor and the Transmission as those designers are.

The parallel between DCCD and DCOP is not only structural, but also dynamic. Two broad categories of solving constraint problems are backtracking (in which candidate assignments to variables are tried one at a time, with various recovery strategies) and consistency algorithms (in which the elements of D are concurrently shrunk by ruling out impossible assignments to the variables). These two approaches correspond to two general approaches to design, which we call "point-based design" and "set-based design," respectively.

[2] In the jargon of the RAPPID project, the variables that describe components and subsystems are termed "characteristics," to distinguish them from other variables in the RAPPID system. In this paper, we use the term "variable" to emphasize the parallel between DCCD and DCOP.

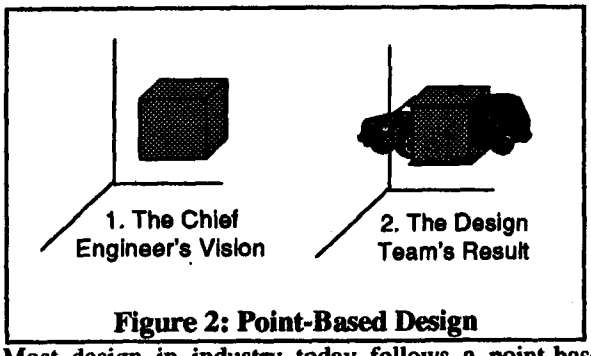

Figure 2: Point-Based Design

Most design in industry today follows a point-based approach, in which the participating designers repeatedly propose specific solutions to their component or subsystem. This approach is typically associated with a chief engineer who is expected to envision the final product at the outset, specifying to the designers what volume in design space it should occupy and challenging them to fit something into that space. Inevitably, as illustrated in Figure 2, some of the chief engineer's assumptions turn out to be wrong, requiring designers to reconsider previous decisions and compromise the original vision. When viewed in the context of a network of components and subsystems mediated by variables, this approach is analogous to constraint optimization by backtracking. Because mechanisms for disciplined backtracking are not well developed in design methodology, this approach usually terminates through fatigue or the arrival of a critical market deadline, rather than through convergence to an optimal solution.

Toyota has pioneered another approach, set-based design [Ward et al. 1995]. In this approach, illustrated in Figure 3, the task of the chief engineer is not to guess the product's location in design space in advance, but to guide the design team in a process of progressively shrinking the design space until it collapses around the product. Each designer shrinks the space of options for one component in concert with the other members of the team, all the while communicating about their interfaces with one another. This approach directly reflects consistency algorithms for solving constraint problems. If the communications among team members are managed appropriately, the monotonicity implicit in shrinking the design space drives the team to convergence.

The use of constraints in design is not new. More than a decade ago, the PRIDE system [Mittal et al. 1986] associated design constraints with design goals in a planning-based approach to design, and the design community has devoted considerable effort to understanding the nature and valid manipulation of design constraints [Ward 1989, Finch & Ward 1996]. However, these constraints have usually been manipulated within a monolithic program, or internally to agents whose organization does not reflect the structure of the constraint network.

[Darr & Birmingham 1996] share several features of the RAPPID vision, including agent communities whose connectivity reflects the constraint structure of a problem and a recognition of the value of set-based design, in the restricted domain of catalog-based design. However, they do not exploit the part-as-constraint insight of RAPPID, instead viewing catalogs of parts as the variables to be instantiated (by selecting a single part from the catalog), and manipulating a separate, abstract set of constraints. RAPPID's approach of recognizing the part or subsystem as the constraint has the advantage of decentralizing the task of defining constraints. Each part's designer is responsible for defining and managing the constraints on variables associated with that part, and there is no need for a centralized knowledge engineering task to define a separate set of constraints.

4. How is DCCD Distinctive?

On the basis of these parallels, the RAPPID team is drawing on DCOP for techniques that are useful in DCCD. At the same time, distinctive features of the DCCD domain require us to develop special techniques for DCOP in design, techniques that may be useful in other domains as well. This section highlights four requirements that we are addressing: different kinds of variables, dynamic implicit constraints, managing global utility in a distributed system, and the high cost of measuring utility. A forthcoming paper will detail the techniques we have developed to address them.

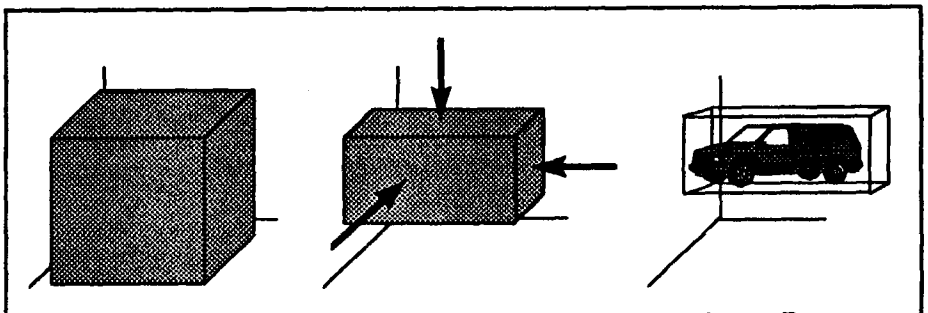

Figure 3: Set-Based Design

4.1. Kinds of Variables

Not all design variables are created equal. A given variable may be meaningful only *internally* to a single component, as an *interface* between selected components, or over the entire *system*. Table 1 summarizes some of the distinctions between interface and system variables.

An *internal* variable is meaningful only within a component, and it is defined entirely by the designer or

Table 1: Classes of Design Variables

	System (Additive)	System (Non-Additive)	Interface (Non-Additive)
Examples	Weight, Power Consumption	Volume, Resonance	Torque, RPM
Main info mvmt	Vertical	←	Horizontal
Major constraints	Among components (e.g., weight)	←→	Among variables (e.g., torque and rpm)
Problem to be solved	Allocation of scarce variable among competing components	←	Compatibility of variables between cooperating components
Quantitative constraints (min, max, range)	Total amount of variable across the system (sum of values for components)	← (not accessible as a sum)	The amount of variable between mating components
Interested components	All components in a system or subsystem	←	Only components that interface directly with one another.

design team responsible for that component. If the component is atomic (that is, with no further decomposition and assignment to lower-level design teams), RAPPID plays no role in the management of its variables. However, if the component is at some higher level of the product decomposition tree, variables that are internal to it may be either interface or system variables among its sub-components.

Interface variables enable the functional interaction of the components within a system. Only components that directly interface with one another manage a given set of interface variables. Little or no vertical information movement (that is, between components and their system) is needed to determine interface variables. Mating components need only to agree on the required interface variables and to instantiate those variables to mutually satisfactory assignments. Not all interface variables need to match exactly on both sides of an interface, since sometimes one component in the interface simply requires that a variable be within a certain range.

A *system* variable must be shared among all sub-components of some component. Thus all components in a system are potentially interested in the system variables for that system. The overall budget of a system variable for a system component is set by the parent component (the "system"), which may be represented either directly by the customer, or by the chief engineer acting as surrogate for the customer. In a complex product with several layers to the product tree, transactions concerning system variables have a strongly vertical flavor. Although peer components do reason about system variables, the constraints on these variables are imposed from above, and components pass them on to lower-level sub-components. Some system variables (like weight or power consumption) are additive. For instance, overall system weight is equal to the sum of the weights of the components. Others, like volume or vibration modes, are not.

These differences in the scope and behavior of different variables mean that RAPPID must attach some reasoning capability to variables as well as to constraints. In RAPPID's market model, interface variables are best supported by extended transactions that resemble the haggling in an oriental *souq*, while system variables offer the potential for markets that close repeatedly at frequent intervals, comparable to a stock market or commodity exchange. The main function of agents that represent variables in RAPPID is to maintain the market protocols appropriate to the given variable.

4.2. Dynamic Implicit Constraints and Carbon Agents

A major difference between design and some other domains that have been modeled as DCOP is that the constraints represented by the designer of a component are often neither explicit nor static.

Traditional approaches to DCOP rely on explicit representation of constraints in forms such as tables, analytic functions, algorithms, or rule sets, but one of the reasons that human designers have not been replaced completely by automated tools is that many important design decisions are still a matter of art and experience. One might argue whether or not all design constraints can in principle be represented explicitly. As a matter of fact, they have not been, and in building a system that will support real design activities in today's industry, we must support implicit as well as explicit constraints. The implication of this observation for RAPPID is that our approach to DCOP must include ways to coordinate the inferences of carbon agents (human designers), rather than simply to replace them with silicon agents. Our experiments suggest that a market-based interface can provide most of the bandwidth needed to coordinate design decisions among agents. It is simple enough to be implemented on automated agents that handle explicit constraints, and intuitive enough that human designers can quickly learn to express their needs through it.

Traditional DCOP also assumes that constraints are static, defined at the outset of the problem and invariant from that point on. Practical design experiences often result in the discovery of new constraints or the relaxation of earlier ones in the course of the process. Again, theoretically one could argue that with sufficient foresight one should be able to define the appropriate set of

constraints at the outset, but foresight is rarely sufficient. A real tool must permit designers to vary the importance they assign to a variable and to communicate this shifting importance to their peers as the process moves forward. RAPPID's design marketplace can reflect these changing priorities through price signals.

The dynamic implicit nature of constraints in design is symptomatic of a more fundamental issue in AI research. Many of AI's most powerful techniques require a model of the environment that will remain accurate long enough for a reasoning engine to conclude something about it. Constructing such a model requires that the environment not change unexpectedly, and that changes initiated by the computer lead to a well-defined steady state. Some common objectives of computerized systems, such as prediction, optimization, and automation, make sense only in such a context.

- Prediction is clearly impossible if changes in the environment invalidate the assumptions on which the prediction was based. It may also be impossible for a stable but nonlinear environment, for then changes made by the computer may move the system into an unstable regime.
- Optimization seeks to identify the assignments to a set of accessible control variables that will yield the best assignment to some dependent variable, given specific assignments to inaccessible state variables (the "environment"). It is meaningful only with respect to that state of the environment, and further assumes that any transients in the system have died out. If the environment changes unexpectedly, or if the dynamics of the system do not have a fixed point, many optimization objectives are not well defined.
- Automation seeks to replace detailed human supervision of a process over time with a predefined computer program. It is often preceded by an optimization activity that identifies desirable assignments to control variables, and the validity of the adjustments that it is programmed to make depend on the predictability of the system. When environmental changes or system nonlinearities invalidate prediction and optimization, automation is also jeopardized.

The real world is overwhelmingly nonlinear, and it holds still only over very short periods of time. Things continue to work because people have the ability to detect unexpected variations and the creativity to take unanticipated steps to deal with them. Computerized systems work best when they are closely integrated with these human capabilities. Algorithms for prediction, optimization, and automation can greatly increase productivity as long as the environment behaves itself, but a human is the best mechanism currently available to keep watch over the environment, and effective computer systems provide for close coupling between the human and the machine. An important contribution of DCCP to the broader constraint optimization community is an example of how humans can participate in the dynamics of an optimization problem, rather than turning it loose on its own and coming back later to see what it has produced.

4.3. Managing Global Utility in a Distributed System

A fundamental tension in any distributed optimization system is balancing the need for a global figure of merit against the desire for local autonomy in decision-making. In some domains, a few constraints may dominate the overall system behavior, and techniques exist to identify them so that they can anchor the decision-making process by using local utility measures to estimate system performance [Liu & Sycara 1995b]. In design, even if one component turns out to dominate system performance, it may be impossible to identify it during the design process.

It is a misnomer to speak of "optimization" in the current practice of distributed industrial design. The coordination mechanisms in common use do not even provably converge, let alone yield a guaranteed optimum. Designer fatigue, budget depletion, or the arrival of a deadline determines the completion of a design project more often than does convergence on a true optimum. In this context, RAPPID brings value to its users by providing a disciplined coordination mechanism, even though we do not yet have an explicit proof of general optimality. However, its underlying mechanisms draw on market dynamics that are known to be optimal in specific settings, and that have been found empirically to perform better than other robust distributed coordination schemes.

Following its supply chain orientation, RAPPID views design as a process of mediating between a set of suppliers (either distributors of catalog parts or custom fabricators) and a set of customers (whose needs the product under design must satisfy). Customers determine the utility of the overall design in the context of their needs, while suppliers determine the cost of its components, and a good design is one that maximizes the difference between customer utility and supplier cost.

A large body of economic theory and practical experience suggests that market mechanisms are an effective way to balance the relative valuations of customers and suppliers through a network of intermediaries. RAPPID's market mechanisms perform this same mediation in distributed design. Designers in RAPPID do not simply compare potential assignments to design variables to find jointly acceptable assignments. They bid for different assignments in a neutral currency. The existence of customers (who buy specific assignments to design variables but do not sell them) and suppliers (who sell specific assignments of variables but do not buy them) imposes a directionality to the constraint network that current DCOP mechanisms do not exploit. The market transactions of a designer with suppliers provide cost information about specific design options, while transactions with customers provide utility

information. The individual designer seeks to maximize the difference between utility and cost.

A major configuration decision in setting up the network is identifying which variables a given constraint buys and which it sells. The general rule is that a constraint sells any variables in which its part plays a causal role, while a constraint buys any variables in which its part has an interest but whose value it does not cause. For example, a motor sells its output torque, and a transmission buys its input torque. We have found that designers are more comfortable with the system if we reverse the roles of buying and selling for variables whose price would be negative under the causal rule (for example, weight in an airframe), and RAPPID can handle this reversal without any problem.

The current RAPPID protocols 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 variables rather than over point assignments, and the bids they offer to one another change as the various assignment ranges shrink. The design is complete when all assignments have converged to point values. 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 have value in organizing the behavior of a design team across multiple design projects, but we have not exploited it in our current scenarios.

At any moment in time, a constraint (representing the designer of a part) sees a range of costs over the range of potential assignments that it is considering for variables that it buys, and a range of utilities over the range of possible assignments that it is considering for variables that it sells. It computes the prices it is willing to pay (to buy a variable assignment) or accept (to sell one) based on the relative costs and utilities that it sees. The function that generates a constraint's bids is altruistic, in the sense that it passes on as much utility as possible to sellers and as little cost as possible to buyers. We can show that when a system using this function converges, the added utility resulting from local decisions by individual designers can be rolled up in a straightforward way to represent the overall utility of the design. This formal result is the basis of ongoing work to explore the degree to which our mechanisms can support true optimization.

RAPPID draws from the growing body of work in nontraditional applications of market mechanisms to multi-agent coordination [Clearwater 1996], and extends the point-based catalog design marketplace of [Wellman 1995] to a set-based methodology that supports implicit as well as explicit constraints. By keeping cost and utility distinct rather than merging them into a single figure of merit, RAPPID addresses the growing demand for explicit cost management in design highlighted in such strategies as Target Costing and Cost as an Independent Variable (CAIV).

4.4. High Cost of Measuring Utility

The utilities of individual parts, not to mention the entire product, are often very expensive to evaluate at a single point. A finite-element model may consume hours of supercomputer time; some design parameters can be evaluated only by building a prototype and testing it destructively. Thus optimization schemes that depend on frequent repetitive computation of point utility as search proceeds will incur excessive cost. Some systems to support designers avoid this problem by selecting components from pre-compiled catalogs [Darr & Birmingham 1996; Wellman 1995]. RAPPID handles this problem by using carbon-based agents (human designers) and a market economy. Designers can often estimate the marginal gain in utility available over a range of possible assignments to a variable, even though locating the exact optimum may be expensive. RAPPID's set-based market mechanisms permit designers to narrow the intervals of interest considerably before making expensive evaluations. Then the estimates of marginal utility available within the interval can guide a decision of whether the additional information from a point evaluation is worth the cost.

5. Summary and Prognosis

Modern design of component-centered products is often distributed across many different companies, reflecting the increased tendency of manufacturers to purchase, rather than make, much of the content of their products. Agents representing the different participants in such a design system correspond to the components being designed, and thus to implicit and explicit constraints imposed by those components. Perception of the strong parallel between such a network of designers and a traditional constraint network permits us to leverage insights from DCOP into distributed design, and also to identify distinctive features of the DCCP domain that suggest mechanisms (such as market dynamics) that may be useful in other DCOP applications.

The RAPPID environment has applied these techniques to support successful experimental design sessions with human designers. In one experiment on a catalog-based power transmission system, a distributed team of designers without central coordination achieved a design that corresponded to the optimal as identified by exhaustive search over the space of alternatives. In a future experiment RAPPID will be used with a major subsystem for a military vehicle using actual subsystem designers and engineers. Another experiment in the works will involve design of a subsystem for a military missile at the missile contractor's site. Both of these experiments will involve many more design variables and designers than the power transmission experiment, and will provide critical insight into the ability of RAPPID (and by extension, DCOP in general) to be applied to highly complex situations.

In addition to these design-specific experiments with real designers, we are currently designing a set of experiments in a simulated environment to quantify the benefits that the enhancements inspired by the design problem can bring to DCOP in general, and studying analytically the relation between the local decisions made by constraint agents and the global character of the resulting solution.

Acknowledgments

RAPPID is sponsored by the Rapid Design Exploration and Optimization (RaDEO) program (formerly MADE) of DARPA, directed by Kevin Lyons, and is administered through the AF ManTech program at Wright Laboratories under the direction of James Poindexter. In addition to the authors, the project team includes Steve Clark, Mike Davis, Bob Matthews (all ITI) and Mike Wellman (University of Michigan)

References

Clearwater, S. H. ed. 1996. *Market-Based Control: A Paradigm for Distributed Resource Management.* Singapore: World Scientific.

Darr, T. P., and Birmingham, W. P. 1996. "An Attribute-Space Representation and Algorithm for Concurrent Engineering." *AI EDAM* 10:1, 21-35.

Finch, W. W., and Ward, A. C. 1996. "Quantified Relations: A Class of Predicate Logic Design Constraints Among Sets of manufacturing, Operating, and Other Variations." *Proceedings of the 8th International Conference on Design Theory and Methodology.*

Fleischer, M., and Liker, J. K. 1997. *Concurrent Engineering Effectiveness.* Cincinnati, OH: Hanser-Gardner.

Hoy, T. 1996. "The Manufacturing Assembly Pilot (MAP): A Breakthrough in Information System Design." *EDI Forum* 10:1, 26-28.

Liu, J., and Sycara, K. 1995a. "Emergent Constraint Satisfaction Through Multi-Agent Coordinated Interaction." C.Castelfranchi and J.P.Mueller, eds., *From Reaction to Cognition: 5th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'93, Neuchatel, Switzerland.* Lecture Notes in Artificial Intelligence 957. Berlin: Springer, 107-121.

Liu, J., and Sycara, K. 1995b. "Exploiting Problem Structure for Distributed Constraint Optimization." V.Lesser, ed., *ICMAS-95: Proceedings, First International Conference on Multi-Agent Systems.* Menlo Park: AAAI, 246-253.

Mackworth, A. K. 1992. "Constraint Satisfaction." S.C.Shapiro, ed., *Encyclopedia of Artificial Intelligence.* New York: Wiley, 285-293.

Mittal, S., Dym, C. L., and Morjaria, M. 1986. "PRIDE: An Expert System for the Design of Paper Handling Systems." *IEEE Computer* 19:7 (July), 102-114.

Parunak, H.V.D. 1997. "'Go to the Ant': Engineering Principles from Natural Agent Systems." *Annals of Operations Research* (forthcoming); http://www.iti.org/~van/gotoant.ps.

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

Ward, A. C. 1989. "A Theory of Quantitative Inference for Artifact Sets, Applied to a Mechanical Design Compiler." D.Sc. Dissertation, Dept. of Mechanical Engineering, Massachusetts Institute of Technology.

Ward, A. C., Liker, J. K., Cristiano, J. J., and Sobek II, D. K. 1995. "The Second Toyota Paradox: How Delaying Decisions Can Make Better Cars Faster." *Sloan Management Review* (Spring), 43-61.

Wellman, M. P. 1995. "A Computational Market Model for Distributed Configuration Design." *AI-EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 9:2 (April), 125-34.

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