Autonomous Agents and Multi-Agent Systems, 2, 111–140 (1999)

The RAPPID Project: Symbiosis between Industrial Requirements and MAS Research

H. V. D. PARUNAK, J. SAUTER AND M. FLEISCHER {v parunak,j sauter,m fleischer}@erim.org
Erim, PO Box 134001, Ann Arbor, MI 48113-4001, USA

A. WARD award@dundee.net
Ward Synthesis, 3446 Gettysburg Rd, Ann Arbor, MI 48105 USA

Abstract. The RAPPID project (Responsible Agents for Product-Process Integrated Design) is developing agent-based software tools and methods that use market place dynamics among members of a distributed design team to coordinate set-based design of discrete manufactured products. This report describes the interplay of industrial requirements and Multi-Agent System (MAS) research in the design, implementation, and testing of RAPPID. Like any industrial project, RAPPID begins with the requirements of the problem domain, and draws selectively from research results to meet those requirements. However, the flow of information is not unidirectional. In the process of addressing its requirements, RAPPID developed some new concepts that hold promise for broader application to distributed constraint optimization. RAPPID is work in process, not a completed product, and this report includes an assessment of what needs to be done in addition to what has been accomplished.

Keywords: manufacturing design, set-based design, emergent behavior, market-based coordination, human-computer integration, least commitment negotiation

1. Introduction

There is a fundamental tension between academic research in multi-agent systems (or any other technology) and industrial application. The researcher focuses on a particular technology (e.g., communication, planning, learning), and seeks practical problems to demonstrate the usefulness of this technology (and justify further funding). The industrial practitioner has a practical problem to solve, and cares much more about the speed and cost-effectiveness of the solution than about its elegance or sophistication. Whatever problem the researcher may address, one can be sure that the solution will include the technology in which the researcher specializes, and the problem will be cast in a way that guarantees the relevance of that technology. The practitioner has no *a priori* commitment to a particular technology, but stubbornly insists on a problem statement that often inconveniences the researchers.

One consequence of this difference in perspective is that industrial applications of MAS place more emphasis on the problem requirements than on the methods used to address them. Industrial people view what they do in terms of a life cycle that begins with requirements analysis (including requirements, positioning, and specification). The results of requirements analysis guide all subsequent stages in

the life cycle: design, implementation and deployment, operation, logistics and maintenance, and decommissioning [16]. The main contribution of this paper is to illustrate the interplay between industrial requirements and agent technology in a specific context, the RAPPID project (Responsible Agents for Product-Process Integrated Design), and to provide a concrete example of the development process outlined in [17]. Section 2 develops requirements by considering the task of industrial design, which RAPPID is intended to facilitate. Section 3 outlines the design of the RAPPID system, while Section 4 surveys the operation of the system and its performance in experimental tests. Section 5 reviews other work in agent-based design systems to highlight the special contributions of RAPPID.

Another consequence of the difference in perspective between academic research and industrial application is that the industrialist must retain many details of the problem that the basic researcher is free to set aside. The academic process of abstracting from the real world is a valuable tool for uncovering fundamental uniformities across problems that at first appear to be quite distinct from one another. However, the loss of detail can sometimes be damaging, in two rather different ways. The lost details may include the fundamental problem to be solved, so that the solution to the abstraction is not useful in real world application. Alternatively, they may hold the key to solving the problem, so that the academic solution to the abstraction is more complex than the solution to the more specific problem. RAPPID exemplifies the second situation. In general, design is a multivariate constraint optimization problem, an abstraction for which many techniques exist that do not consider the specific domain of design or the causal details it brings to the problem. Our experience with RAPPID suggests that these details can lend important direction to the underlying constraint management problem, and we are exploring generalizations of our mechanisms to other constraint problems [18].

Both of these consequences reflect the interplay of domain details with MAS techniques. The first reminds us that ultimately people buy MAS systems because such systems solve real problems more effectively and more efficiently than other techniques, so we should pay careful attention to the details of their problems and what they think constitutes a solution. The second reminds us that the process of engineering applications is not void of research creativity, but can lead to new ways of solving research problems that would not be naturally apparent in a less situated environment.

2. Requirements for RAPPID

The first step in the industrial life cycle of any system is determining the requirements that the system must satisfy. These requirements grow out of careful study of the work processes that the system is to support. Industrial designers are responsible for describing the function and structure of new products and the facilities that manufacture them. The nature of this process suggests several requirements that must be satisfied by any MAS intended to support it.

2.1. The real-world context of design

A designer seeks to embed a set of *functions* (e.g., optical, electromechanical, control) in an artifact with specified *characteristics* (e.g., weight, color, complexity, materials, power consumption, physical size). Conflicts arise when different teams disagree on the relation between the characteristics of their components and those of the entire product. Some conflicts are within the design team: How much of a mechanism's total power budget should be available to the sensor circuitry, and how much to the actuator? Others pit design against other manufacturing functions: How should we balance the functional desirability of an unusual machined shape against the increased manufacturing expense of creating that shape?

It is easy to represent how much a mechanism weighs or how much power it consumes, but there is seldom a disciplined way to trade off weight and power consumption against one another. The more characteristics are involved in a design compromise, the more difficult the trade-off becomes. The problem is the classic dilemma of multivariate optimization. Analytical solutions are available only in specialized and limited niches. Even when explicit methods of evaluating constraints are available, they may be so time-consuming or costly that their use must be limited. Thus many decisions rely only on the expert intuitions of experienced designers, and trade-offs are usually supported by processes such as QFD (Quality Functional Deployment) [7] or resolved politically, rather than in a way that optimizes the overall design and its manufacturability.

Because of the complexity of balancing characteristics against one another, there is strong pressure to select characteristics sequentially, fixing their values one at a time so as to narrow the search space for subsequent characteristics. This approach unfortunately excludes some portions of the search space from consideration, and can actually slow down the design process by forcing decisions to be made one at a time rather than permitting parallel exploration.

These problems are compounded when design teams are distributed across different companies. Modern industrial strategists seek to eliminate excessive layers of management and push decision-making down to the lowest level. They envision the "virtual enterprise," formed for a particular market opportunity from a collection of independent firms with well-defined core competencies [12]. It is increasingly common for the original equipment manufacturer (OEM) of a complex product to purchase half or even more of the content in the product from other companies. For example, an automotive manufacturer might buy seats from one company, brake systems from another, air conditioning from a third, and electrical systems from a fourth, and manufacture only the chassis, body, and powertrain in its own facilities. The suppliers of major subsystems (such as seats) in turn purchase much of their content from still other companies. As a result, the "production line" that turns raw materials into a vehicle is a network, or "supply chain," of many different firms. As the responsibility for manufacturing components moves from the OEM to suppliers, the design capability naturally follows, leading to a situation in which no single organization houses the entire design team for the final product.

Collaboration among designers who are distributed across several firms is difficult enough because of geographical separation. In addition, proprietary concerns limit the exchange of information among designers beyond what might take place within a single company. For example, a designer of a composite part may develop specific design details that relate to a proprietary process used by the manufacturer of that part, but these details cannot be disclosed to designers in other firms without compromising the manufacturer's trade secrets.

2.2. A set of requirements

Five requirements emerge from this review of modern industrial design.

Multidimensional.—No single characteristic of a design is an unambiguous metric of its goodness. Every designer is responsible for some constellation of features or characteristics that contribute to the goodness of the overall product. A successful system to support design will provide ways to rationalize the trade-offs among these varied characteristics.

Distributed.—As firms draw their inputs through extended supplier networks, the responsibility for design tends to spread as well. As a result, the designers who must work together on a single product are usually located in different places, owe their allegiance to different companies, and may be involved in several different projects involving different team members. A successful system will accommodate this distribution.

Limited Information Interchange.—Because designers are distributed organizationally as well as geographically, proprietary concerns often limit the amount and kinds of information that they can share with one another. While some design details must be shared among participants to ensure the coherence of the entire product, others may concern trade secrets of a single firm's processes, and cannot be shared openly without jeopardizing those secrets. A successful system will be selective in the information that must move among designers.

Parallel Exploration of Design Space.—The sequential fixing of design decisions that is common in practice is widely recognized to be sub-optimal. It excludes consideration of design options whose combinations may be superior even though by themselves they are not attractive, and it forces design decisions to be made serially, thus slowing down the process. A successful system will permit parallel exploration of the design space.

Integration of Humans and Computer Tools.—Computerized tools can help with many details of design, such as circuit emulation in electronics or finite element analysis of physical structures, but much of the design process will rely on human creativity for the foreseeable future. In particular, many of the constraints that must be balanced in formulating a design cannot be represented explicitly or analytically, or explicit evaluation may be too time-consuming or costly for indiscriminate use. Thus most decisions depend on the intuition and experience of a human designer. A successful system will support seamless integration of both human and computer-based agents.

3. Design of RAPPID

With requirements in hand, the MAS engineer develops a design. In RAPPID, this process begins with identifying some strategic mechanisms that seem most likely to satisfy the requirements. These mechanisms serve as the basis for sketching out an overall concept of operation that permits us to identify the various species of agents and their behaviors.

3.1. Strategic mechanisms

RAPPID rests on three strategic mechanisms: autonomous agents as a way to distribute decision-making among a community of humans and computers, markets as a mechanism for coordinating distributed decision-making, and set-based design as a means of making decisions in parallel.

3.1.1. Autonomous agents. Different practitioners use the term "agents" in many different ways. We have found it most helpful in communicating with industrial users to view agents as an incremental extension of previous software technologies, as suggested in Table 1. In the beginning was the Program, a monolithic deck of machine instructions and data tied together with tangled "goto" statements that took over the complete resources of the computer when the user fed it into the card reader. The structured programming movement modularized program code through constructs such as subroutines and structured loops. These constructs localized the definition of *how* the program would function, but relied on external definition of the data on which the code would operate (defining *what* the program would do) and external invocation (to tell *when* the program would do it). The next major development, objects, gave software modules local state as well as local code, but invocation was still determined externally, by sending a message. Agents add two things to (passive) objects: a local thread of control, and local initiative (usually expressed as local goals). Together, these enable the agent to monitor and respond to its environment autonomously (that is, without being externally invoked).

Two of our requirements make agents an attractive technology for a modern design environment. First, agents are intrinsically distributed. While their local

Table 1. Agents in historical perspective

	Monolithic program	Structured programming	Object-oriented programming	Agent-oriented programming
How does a unit behave? (Code)	External	Local	Local	Local
What does a unit do when it runs? (State)	External	External	Local	Local
When does a unit run?	External	External (called)	External (message)	Local (thread; goals)

threads can be supported on a single processor, it is also natural to distribute them across a network, supporting the distribution requirement. Second, the modularity of agents makes it natural to encapsulate humans as peer agents to computer processes, as in [14], using a common language and protocols to integrate people and machines. In the nature of the case, this integration requires people to reduce the bandwidth of their communication to a level that computerized agents can handle. While RAPPID's market mechanisms and partitioning of the problem permit much of the design task to be accomplished with simple protocols, we recognize that other tasks will exceed their capacity, so that RAPPID will not automate the entire design problem. Its automatic mechanisms must function alongside conventional interactions, which we describe as SLOWH (Standard Legacy-Oriented Work Habits). Agents permit us to build a system partly in computer algorithms and partly in human behaviors, a mixture that we describe as "hybrid carbon-silicon systems." Table 2 summarizes these distinctions.

3.1.2. Market-based control. Researchers addressing distributed problems in a wide range of domains have recently begun to turn to market-based mechanisms for coordination. [3] offers a convenient collection of applications to fields as diverse as computer network control, memory allocation, factory scheduling, pollution management, and air-conditioning load balancing. In all of these areas, competitors for scarce resources can efficiently express their needs in terms of a common currency, and the balance between supply and demand can set prices for those resources that rationalize the distribution of resources across competitors.

[22] reports some early experiments in market-based automatic configuration of a manufactured product, in the case where all components and their characteristics are defined in advance. RAPPID extends these methods to deal with systems that include components being designed concurrently with the rest of the system. Designers communicate their relative costs and utilities for various system characteristics in terms of a common currency, and market dynamics rationalize the various trade-offs among these characteristics.

Using market mechanisms to coordinate the activities of designers supports four of the requirements outlined above. Market dynamics can develop valuations of various characteristics in a common currency, thus addressing the Multidimensional requirement. Markets are a natural way to coordinate Distributed systems, because of the simplicity of the associated protocols. Communications in terms of the cost and utility of assignments to characteristics runs less risk of disclosing

Table 2. RAPPID/SLOWH and carbon/silicon distinctions

	RAPPID (market mechanisms)	SLOWH mechanisms
Carbon (Human)	RAPPID-trained designers	Traditional human interactions (phone conversations, meetings, back-of-envelope sketches, prototype models, ...)
Silicon (Computer)	RAPPID server and clients	Traditional tools for CAD, FEM, Workflow, data exchange, ...

proprietary information than more detailed exchange of technical information, supporting Limited Information Interchange. Because people are accustomed to market interactions in other areas of life, such interactions form a natural medium for interacting with one another and with computerized tools, fostering Human-Computer Integration.

In addition to addressing these requirements, a market-based approach provides a credible mechanism for relieving second-law pressures to disorder in a multi-agent community [15]. The Second Law of Thermodynamics observes that closed systems progressively become more disordered over time. It is not obvious that a large collection of agents will organize itself to do useful things. The Second Law warns that the result of such an architecture may be disorder.

Natural agent-based systems do organize themselves with striking efficiency. A common explanation is that a system can become more organized if energy is added to it from the outside (for example, by the metabolism of the food gathered by an insect hive). The addition of energy is necessary for self-organization, but hardly sufficient. Gasoline in construction equipment can erect a building, but the same gasoline in a terrorist's bomb can destroy it.

In natural systems, agents can organize themselves at the macro level because their actions are coupled to a dissipative or disorganizing process at a micro level. The system can reduce entropy at the macro level by generating more than enough entropy at the micro level to pay its second-law debt. To adopt another metaphor, it provides an entropy leak to drain disorder away from the macro level (where useful work is done) to the micro level (where it won't interfere with the system's function). Agents that perceive the flow field generated by the dissipative process can use it to coordinate their activities with one another. Markets provide such a field in the distributions of prices and the flows of currency that they support, and are an easily engineered mechanism to avoid disorder in agent communities.

3.1.3. Set-based ideas. A product's characteristics can be thought of as dimensions of a Cartesian space within which the product is defined. Traditionally, designers seek to *build* a design to fit a subspace of characteristics that the chief engineer has specified in advance. In many cases, the chief engineer's vision may in fact be unrealizable. The design team routinely and repeatedly finds that feasible solutions to the problem require violating the boundaries of the initial specification (Figure 1). Since these boundaries were the original basis for coordination among different designers, these violations usually require previous decisions to be withdrawn and reconsidered, delaying the process and increasing its cost.

Toyota has pioneered a more promising vision that begins with a design space much larger than necessary, and then shrinks it incrementally to *discover* where the desired design is (Figure 2). Now the chief engineer's task is not to divine in advance where the solution is, but to manage a process that guides the team in discovering the solution. A study of Toyota's design methods [21] concludes that this set-based approach is responsible for allowing Toyota to bring a vehicle to market in 27 months from concept approval, compared with more than 30 months in the best US competitors, and with a team only two-thirds the size. As a result, Toyota enjoys not only lower product costs (because it consumes 50% fewer

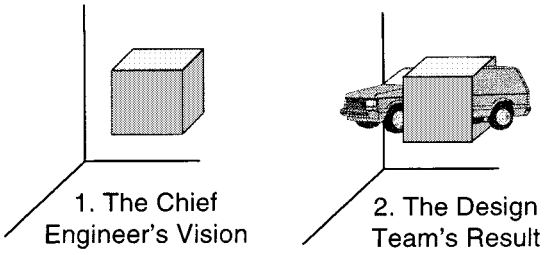

Figure 1. Building a design to fit a design subspace

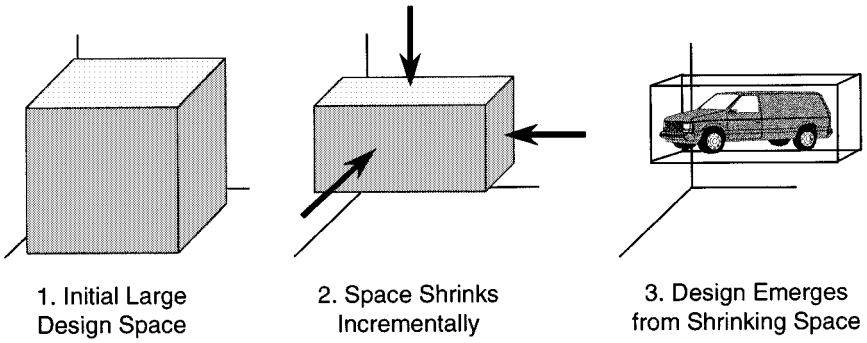

Figure 2. Shrinking design space to discover a design

person-years for product development than do its competitors) but also more frequent product introductions that can track customer desires more closely and thus increase market share.

Current design tools (including those used by Toyota) offer no quantitative support for this vision. With a market in design characteristics, low prices identify slack characteristics (dimensions of the space) that the chief engineer can collapse by buying up allocations of that characteristic. This action simultaneously reduces the amount of that characteristic available for use by designers and increases the funds in the system available to purchase other characteristics to compensate for the decrease in the given characteristic. As designers buy and sell, the relative prices of the various characteristics change, identifying a new slack dimension that can further shrink the design space.

Set-based design offers several advantages over conventional approaches.

- Traditional design evaluates one point solution after another in a hill-climbing search. For many design domains, the evaluation of a point can be expensive, time-consuming, or both: a detailed finite-element analysis might require a day of Cray computer time, and some tests require construction of a vehicle mock-up for crash testing, at a cost on the order of $1M per copy. Furthermore, point-based design serializes design decisions, causing further delays.

- A shrinking design space can guarantee convergence properties that would be difficult to ensure otherwise among a distributed team. If everyone is shrinking his own subspace, the system as a whole must be converging.
- Flow fields are an important mechanism for emergent organization among agents. The flow of currency in a marketplace is one important flow field; the shrinking space approach provides another. Instead of imagining that boundaries contract in a space of uniform density, hold the boundaries fixed and permit the density of the space to vary. A set of design options is then like a Dutch polder from which more and more water is pumped out. In distributed set-based design, the options discarded by one designer guide other designers. That is, the flow of design options across the boundaries of design sets helps organize the community.

RAPPID's support for set-based design addresses the requirement for Parallel Exploration of Design Space.

3.2. Concept of operation

RAPPID uses a marketplace to set prices on each characteristic of a design. Agents representing each component buy and sell units of these characteristics. A component that needs more latitude in a given characteristic (say, more weight) can purchase increments of that characteristic from another component, but may need to sell another characteristic to raise resources for this purchase. In some cases, analytical models of the dependencies between characteristics may help designers estimate their relative costs, but even where such models are clumsy or nonexistent, prices set in the marketplace define the coupling among characteristics.

Figure 3 shows a design decomposed into Component Agents (rounded rectangles), each with one Characteristic Agent (ovals) for each dimension in the design space. For example, the "SS.Weight" Characteristic might represent the constraint that the entire product weigh between 5 and 10 kg. The topmost Component represents the complete product (in this case, a new suspension for a tank), and is the concern of the Chief Engineer, who reflects the Customer's requirements in the initial allocation of design space. The bottom-most Components are either custom-manufactured or (in the Figure) selected from an on-line Parts Catalog. Designers, who typically have responsibility for intermediate levels of the product tree, propagate the constraints from the top and bottom of the tree toward each other. Each Component buys and sells allocations on its Characteristics to and from other Components. In the current version of RAPPID, these transactions are guided by the designer responsible for the Component, through the interface described in Section 4 below.

A product exists not only in Design Space (characterized by features such as weight, power, and shape), but also in Manufacturing Space (characterized by such features as Process, Raw Material, and Operator Skill) and Requirements Space (characterized by behavioral features such as Speed, Range, and Survivability). As illustrated by the curved arrows in Figure 4, the problem of Integrated Product-

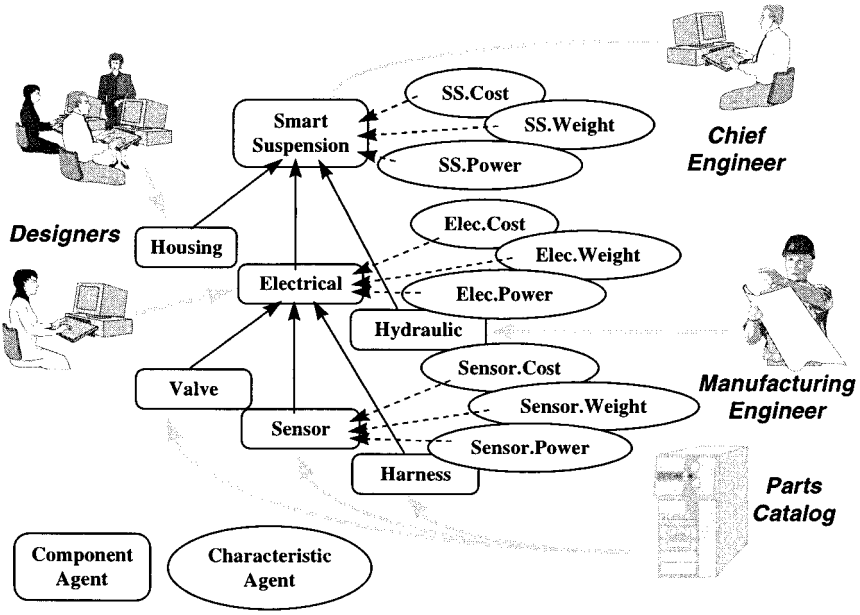

Figure 3. RAPPID design ecosystem for a new tank suspension

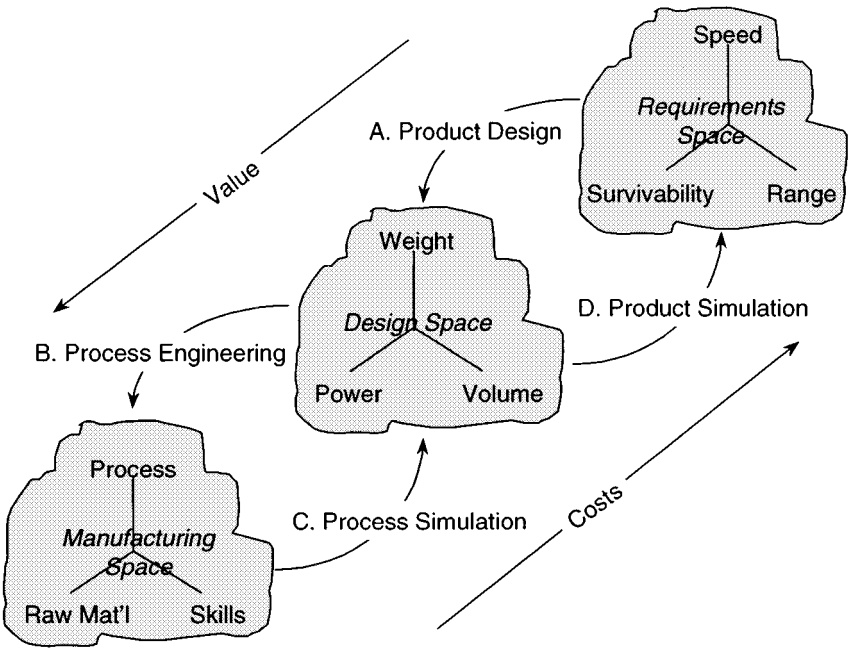

Figure 4. IPPD as a mapping among spaces

Process Design (IPPD) can be viewed as one of mapping among these spaces. Value is defined by the customer, starting in Requirements Space, and flows down to design and then to manufacturing. For example, through battlefield simulations, the customer can determine the incremental value of another 5 kph of vehicle velocity, or another 20 km of range. Cost is defined by manufacturing flows up through design to the customer. For example, the designer (with input from manufacturing) determines the cost of providing an increment of velocity or range. A marketplace in these three spaces supports mapping in both directions among them. The current version of RAPPID focuses on a marketplace in the design space. Later research will extend the market approach to the neighboring spaces.

3.3. Agent species

With system requirements and a high-level behavioral specification in hand, the next step in designing an industrial agent-based system is defining a preliminary set of agent classes. We view multi-agent systems as "synthetic ecosystems," because we draw heavily on analogs from naturally occurring agent-based systems [15]. In this context, it is natural to view the different classes of agents as distinct species interacting in a shared environment. In this section, we discuss the different species we have identified, and refine one particular species.

3.3.1. Four agent species. A fundamental characteristic of agents in both natural and synthetic ecosystems is that they correspond wherever possible to things with well-defined boundaries rather than to functions. This characteristic is in tension with the functional decomposition that is traditional in software engineering, but in our experience yields agent communities that are more robust and redeployable than the traditional approach. Functions have no *a priori* ontological status, but are defined by the analyst, so their boundaries can easily shift from one phase of a system's life to another, and are liable to misunderstanding by successive generations of humans responsible for maintaining the system. In contrast, the boundaries of natural entities in the problem domain are relatively independent of the analyst, and so tend to persist through successive changes in system function and programming staff.

The overall architecture outlined in the previous section suggests four agent species: Components of the product, their Characteristics, the Markets in which external Characteristics are traded, and the Constraints imposed by a Component on the Characteristics in which it has an interest.

A *Component* is a node in the tree that represents the structure of the product. In general, a designer or design team is assigned to each component. Depending on the design approach that is used, the tree may or may not be defined in advance, but it is still important to distinguish between the component agent, which represents a functional slot in the design, and a specific candidate to fill that slot. For example, the transmission agent in the design of a power system might consider several different physical transmissions. An alternative approach is for each physical component to function as an agent, competing for a role in the

design. While this approach may be useful for catalog-based design, the more fundamental view of the product node as the component agent supports a broader set of problems, including those in which the specific physical attributes have not yet been defined. In the current version of RAPPID, the component agents consist of the human designers responsible for the components and a user interface to the rest of the system.

A *Characteristic* is a definable attribute or parameter of a component, such as its weight, power consumption, RPM, torque, or size. Characteristics are defined per component. The weight of (say) the motor is a different characteristic than the weight of the transmission, though both are of the same characteristic type.

A *Constraint* is a relation between two or more characteristics. Constraints typically arise either from laws of nature (e.g., power consumption equals voltage times current flow) or design decisions (e.g., the output RPM from the motor equals the input RPM to the transmission; a given RPM and torque characterize the same shaft). Many constraints are implicit, that is, not known in analytical or tabulated form, but only accessible through the designers' experience. In the current version of RAPPID, all constraints exist in the minds of human designers, not as separate software agents. We define a separate species for them to anticipate extensions of the system in which some constraints can be captured explicitly and manipulated automatically.

A *Market* is a process that maps potential buyers and potential sellers of a good to one another and optionally to a price at which a sale can take place. The goods traded in such a market are characteristics or options for characteristics. Each distinct good requires a separate market, and markets for different goods may have different protocols. We visualize each market as existing in the form of a server on a network. In practice, many markets might exist on a single server, but this implementation detail makes no difference to the actual operation of the system. In the current implementation of RAPPID, there is a one-to-one correspondence between markets and characteristics, and a single agent represents a given characteristic and its market.

3.3.2. A closer look at characteristics. All characteristics are not created equal. A given characteristic may be classified on the basis of the number of components to which it applies, whether its aggregation across components is additive or not, and whether it is coupled to other characteristics.

Scope of Characteristics.—A given characteristic may be meaningful only *internally* to a single component, as an *interface* between selected components, or over the entire *system*.

An *internal* characteristic is meaningful only within a component, and it is defined entirely by the designer or 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 mechanisms play no role in the management of its characteristics. However, if the component is at some higher level of the product decomposition tree, characteristics that are internal to it may be either interface or system characteristics among its sub-components.

Interface characteristics enable the interaction of components within a system. A classic example is the torque and RPM between a motor and a transmission. Only components that directly interface with one another trade in interface characteristics, and the issue they seek to resolve is compatibility between cooperating components rather than distribution across competing components. Little or no vertical information movement (that is, between components and their system) is needed to determine interface characteristics. Mating components need only to agree on the required interface characteristics and to achieve those characteristics within the scope of their allocated system characteristics. Not all interface characteristics need to match exactly on both sides of an interface, since sometimes one component in the interface simply requires that it receive at least (or at most) some amount of a characteristic from its partner.

A *system* characteristic must be shared among all sub-components of some component. Thus all components in a system are potentially interested in the system characteristics for that system. The overall budget 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 characteristics have a strongly vertical flavor. Although peer components do trade back and forth in system characteristics, the constraints on these characteristics are imposed from above, and components pass them on to lower-level sub-components.

Aggregation of Characteristics.—Characteristics that apply to more than one component may be either additive (like weight or power) or non-additive (like resonant frequency or volume). The total system value for additive characteristics is the sum of the component values. If one component consumes more of an additive system characteristic, other components must make do with less. Non-additive characteristics aggregate in more complicated ways. Interface characteristics are intrinsically non-additive, and many system characteristics are additive, but there are non-additive system characteristics that in some ways resemble interface characteristics.

Table 3 summarizes the three kinds of non-internal characteristics. The arrows in the "System (Non-Additive)" column indicate whether the particular feature for these characteristics is more like additive system characteristics or interface characteristics.

Coupled Characteristics.—Some sets of characteristics are tightly coupled, and must be varied together in exploring the space of possible designs. Such coupled characteristics frequently arise in dealing with non-additive characteristics (both system and interface), which often come in coupled sets. One cannot get torque on a shaft without also getting RPM, so markets to deal in these coupled characteristics must provide ways of ensuring that they trade together. A buyer who advertises for torque and RPM will not be happy receiving one bid for the appropriate torque but a useless RPM from one vendor, and another bid for the right RPM but a useless torque from a second vendor. Because these characteristics are coupled non-additively, they need to be bought and sold together.

Table 3. Classes of design characteristics

	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 characteristics (e.g., torque and rpm)
Problem to be solved	Allocation of scarce characteristic among competing components	←	Compatibility of characteristics between cooperating components
Quantitative constraints (min, max, range)	Total amount of characteristic across the system (sum of values for components)	← (not accessible as a sum)	The amount of characteristic between mating components
Interested components	All components in a system or subsystem (e.g., "power" is relevant only to electrical subsystems)	←	Only components that interface directly with one another

3.4. Agent behaviors

Once agent species have been tentatively identified, the next step is to define their behaviors. Components offer to buy or sell ranges of assignments to characteristics of interest to them, by posting bids with the markets for the characteristics. Each characteristic compares the bids it receives from its components and recommends how those components should shrink their ranges in converging toward a solution.

3.4.1. What do components do? A component expresses its interest in a characteristic by telling the market for that characteristic:

1. Whether it wants to buy or sell the characteristic;
2. A range of assignments to the characteristic 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 characteristic.

Items 2 and 3 define a region of price/assignment space. The component that is bidding indicates how its price varies with assignment (item 4). In catalog-based design, this price function is known in great detail, but many constraints are not known in advance, so RAPPID supports qualitative indications of price shapes, as in Table 4.

Ordinal characteristics such as weight or torque, whose potential assignments can be ordered, support price/assignment ranges and qualitative price shapes. Another class of characteristics ("nominal characteristics") does not support a

Table 4. Price/assignment shape indicators

/	Price increases with assignment
\	Price decreases as assignment increases
∧	Price is maximum between extreme assignments
∨	Price is minimum between extreme assignments
–	Price is relatively flat
~	Price/assignment relation is unspecified

natural ordering of potential assignments (e.g., the material out of which a product should be made). Shape indicators are not meaningful for such characteristics. RAPPID uses directed acyclic graphs (DAG's) to record a component's preferences for nominal characteristics. Section 3.5 discusses the definition and use of both price shapes and preference DAG's.

It can be shown that if components compute their prices according to a particular algorithm, their local behavior yields a global maximum of utility over cost. We define:

- $B_{i,o}$ is the sum of the prices of offers to buy made by the ith component (the o standing for "offer").
- $B_{i,r}$ is the sum of the prices of the "received" bids to sell to component i, that is, the total amount that component i would pay if it bought at the prices offered to it.
- $S_{i,o}$ is the sum of the prices of component i's offers to sell, and $S_{i,r}$ is the sum of the prices offered to component i by its buyers.
- C_i is the internal costs experienced by component i (that is, costs experienced by the agent but not visible as purchases from suppliers 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. In addition, other constraints may have internal costs or utilities in addition to the external ones that they receive from their trading partners.

All of these variables are functions over the set of characteristics in which the component has an interest, evaluated at the point when the components' ranges on characteristics have collapsed. The rules that yield globally desirable behavior are then:

Rule 1: $B_{i,o} \leftarrow S_{i,r} + U_i - C_i$.—The total amount that component i offers to spend with its suppliers ($B_{i,o}$) includes all of the revenue that it receives from its customers ($S_{i,r}$) and all of the additional utility that it realizes (U_i), reserving only the internal costs that it incurs (C_i). That is, a component does not retain any local profit, but passes it all on to its suppliers.

Rule 2: $S_{i,o} \leftarrow B_{i,r} + C_i - U_i$.—The total of the prices that component i charges to its customers ($S_{i,o}$) includes all of the costs imposed on it by its suppliers ($B_{i,r}$) and all of its internal costs (C_i), less the internal utility that it realizes (U_i). That is, a component does not absorb any costs locally.

Informally, these rules express a cooperative economy, one in which the "self-interest" of the individual participants is carefully bounded by a global "ethic" to which they all subscribe. RAPPID would operate even if these rules were not enforced, but the convergence of local decisions to a global optimum could no longer be assured.

In practice, components do not know in advance the point to which their variable ranges will collapse, and so cannot apply these rules exactly. The essential point of the rules is that two underlying heuristics yield global coherence:

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

Empirically, these approximations to the formal rules yield reasonable results in the problems to which we have applied them.

In some cases a component 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 component'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 components 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 component 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.

3.4.2. What do markets for characteristics do? A market agent maintains a market in the assignments to its associated characteristic. Its basic functions guide the components interested in that characteristic in narrowing the ranges of their bids.

Recommend Trimming.—Because the market knows the assignment ranges, price ranges, and shapes of the bids submitted by interested components, it can recommend how those components should trim their assignment ranges, thus narrowing in on an assignment that is maximally beneficial to all of them. RAPPID defines two trimming heuristics, one for ordinal variables and another for variables with no natural ordering, as discussed in Section 3.5 below.

Enforce Protocol.—The market knows the "one-of/any-of" status of all interested components. It enforces the rule that a component 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 market uses its knowledge of each bidder's status to tune bidding recommendations.

Recommend Direct Negotiation.—Sometimes the market'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 permit strong recommendations by the market. In these cases, the market recommends that the components 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 SLOWH communication between human designers.

Other Services.—In addition to its basic function of guiding components in narrowing their bid ranges, a market provides a number of other services that cannot be discussed in detail for lack of space.

- At any moment in the process, a given component is considering a number of different variables and their impact on the satisfaction of the constraint. Each market can compute various "figures of merit" that its components can use to compare the state of problem solving with respect to the different characteristics and select the most important one to consider next. For example, one characteristic may offer a much wider price range than another.
- Sometimes a group of characteristics needs to be traded together. For example, the torque and RPM of a shaft must both be purchased from the same source, the supplier of the (single) shaft. The associated markets can be grouped together to help the components deal with them as a unit.
- Some characteristics are of interest to multiple buyers, multiple sellers, or both. RAPPID defines methods to integrate the bids from these multiple participants.

3.4.3. 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 RAPPID 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. Components 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.

3.5. Qualitative representations

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

3.5.1. Qualitative trading of ordinal characteristics. Where quantitative details are not available, a component communicates its utility (as a buyer) or its cost (as a seller) for an ordinal characteristic in terms of a range of possible assignments to the characteristic, a range of prices over that range of assignments, and a curve shape indicator (Table 4) that estimates the qualitative relation between the two. The market agent for the characteristic computes the intersection of the assignment ranges of interest to the components. 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 components trim their assignment ranges to maximize that difference.

For example, one component bids to buy a given characteristic with assignment range 2–7, price range 5–20, and curve shape '/', and another component offers to sell the same variable with assignment range 5–8, price range 5–25, and curve shape '∧'. Figure 5 illustrates graphically the range of assignments of common interest to the components (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 components 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 6) no single trim recommendation can be made. In these cases, the market recommends direct negotiation between the components. 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 components to narrow in on the most promising regions of the search space.

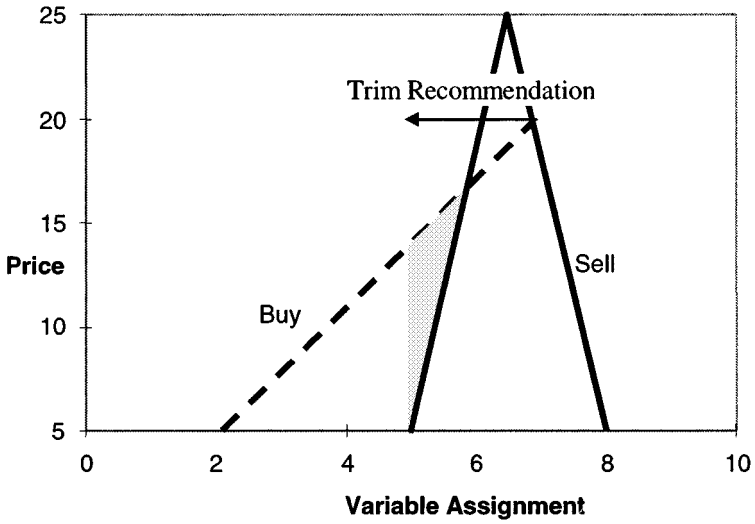

Figure 5. Resolving qualitative bids

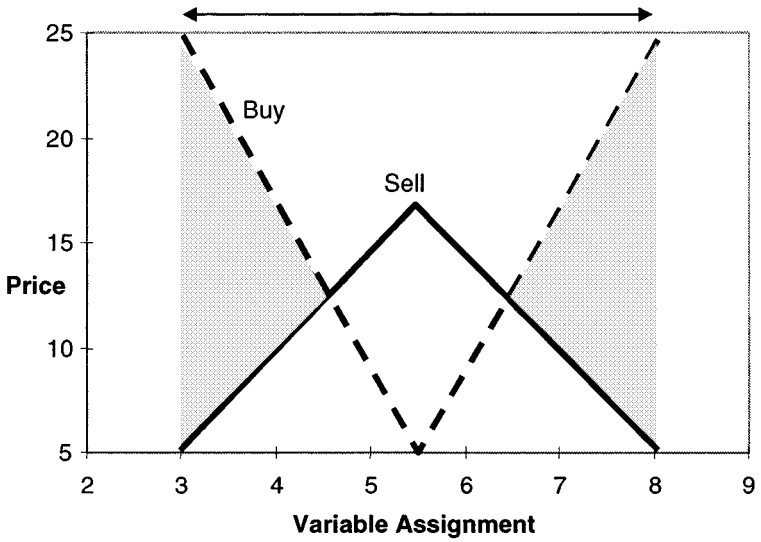

Figure 6. A configuration that does not support automatic trimming

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

Price DAG's.—Each component interested in a nominal characteristic 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 component 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 7, the component (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.

A Constraint Agent offering Material for sale constructs a similar DAG (Figure 8).

Computationally, price DAG's are represented as adjacency matrices, with an entry in each cell in which the column directly dominates the row. Table 5 shows the adjacency matrix $\boldsymbol{B}$ for the buy DAG in the example. The sell adjacency matrix is $\boldsymbol{S}$.

Support for Trimming.—A market for a nominal characteristic uses the price DAG's to recommend which possible assignments may be excluded from further consideration, by executing the following steps.

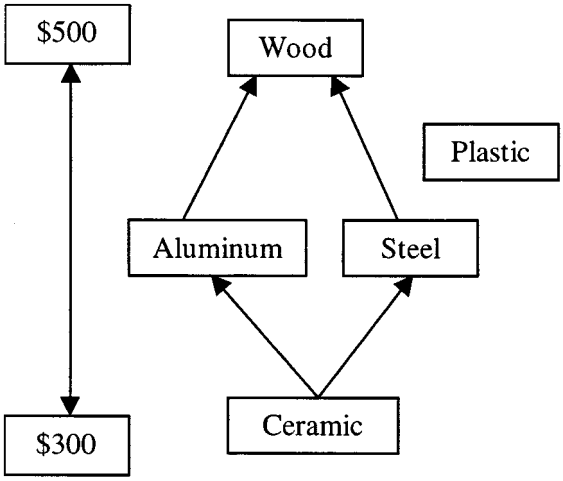

Figure 7. A buy DAG

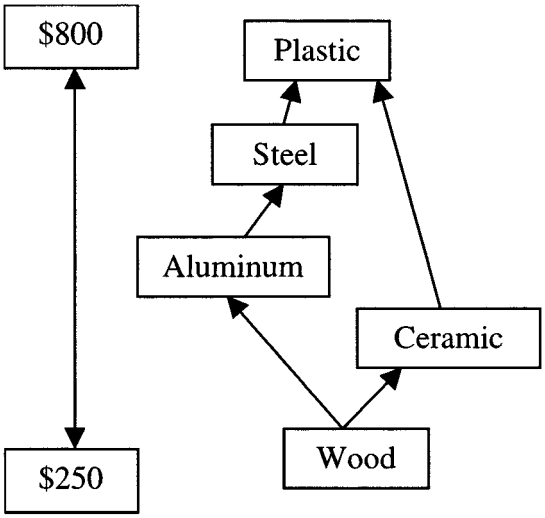

Figure 8. A sell DAG

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. $\boldsymbol{B}^*$ and $\boldsymbol{S}^*$ are the buy and sell dominance matrices, respectively.
2. Transpose $\boldsymbol{S}^*$ 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 $\boldsymbol{B}^*$ and $(\boldsymbol{S}^*)^T$, the one with higher utility also has lower cost, and is thus clearly preferred to the alternative.

Table 5. 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

3. The non-zero cells in $\boldsymbol{J} = \boldsymbol{AND}(\boldsymbol{B}^*, (\boldsymbol{S}^*)^T)$, where the logical ***AND*** is cell-by-cell, identify orderings that are true of both buyer and seller, and define a joint DAG (Figure 9) that reflects the excess of utility over cost. On the basis of this particular DAG, the market recommends that its participating components determine their relative preferences for Plastic and Wood, and trim from the bottom of the joint DAG.

Support for Direct Interaction.—Price DAG's also enable markets for nominal characteristics to identify a non-overlap condition and provide information to the components that may help them close the gap. A non-overlap can be detected either because the price ranges specified by the components 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 components. These cases are the non-zero cells in $\boldsymbol{NOT}(\ \boldsymbol{AND}(\ \boldsymbol{OR}(\boldsymbol{S}^*, (\boldsymbol{S}^*)^T), \boldsymbol{B}^*))$. When one component provides a preference and another does not, this mismatch indicates to the second component which preferences it should explore further to advance the overall relaxation.

The components 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. $\boldsymbol{AND}(\boldsymbol{B}^*, \boldsymbol{S}^*)$ identifies these cases.

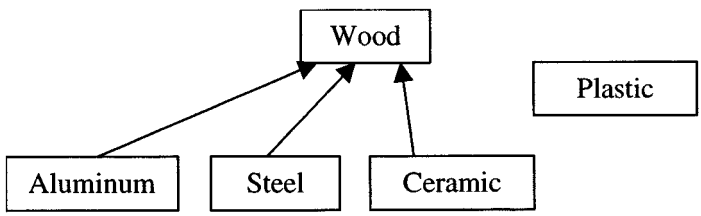

Figure 9. Joint price DAG

4. RAPPID in operation

Currently, RAPPID implements market agents and their associated characteristics as a Web-based market server. Component agents are human designers, supported by a Java-based user interface and spreadsheets that partially automate the computation of constraints, particularly for components whose alternatives can be cataloged in advance. Experiments with the system show that it is effective in permitting a distributed group of designers to search a complex design space, but also highlight the need for additional tools to make it fully deployable in an industrial environment.

4.1. User interface

RAPPID provides two tools to a designer (the heart of the component agent). One tool, a spreadsheet, helps the designer keep track of the utilities and costs of various design alternatives. The other tool, a Java-based Web interface, permits the designer to make bids in the markets for the various characteristics. We illustrate the application of these tools to the design of 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 10 shows the overall ecosystem for this problem.

Table 6 shows a fragment of the catalog of mechanical transmissions available to the transmission designer. The lower portion of the catalog (with the large number "22" in the upper left-hand corner) contains a row for each transmission and a column for each characteristic constrained by the transmission. Only four of the twenty-two transmissions in the catalog and seven of the fifteen characteristics are reproduced in the figure. Strictly speaking, three of the columns are not "characteristics" in the sense of our agent species: the stock number, the price (off the figure), and the "select in or out" column (a computed column that indicates whether each entry satisfies the current ranges of the characteristics). The two rows above the catalog labeled "Min Value" and "Max Value" provide the range of values of each characteristic in the entire catalog, while the top two rows, labeled "Lower Bound" and "Upper Bound," can be set by the designer to filter the catalog on specific ranges of individual characteristics. Another region of the spreadsheet (not shown here) records the current ranges of prices in other markets in which the transmission agent must deal, enabling it to compute its overall costs and utilities as a function of the range of transmissions it is currently considering.

Figure 11 shows the interface through which the transmission enters its bids in one of the market clusters in which it has an interest, that for the shaft that connects it to the motor. This cluster includes markets for several characteristics that must be traded together, since they are physically linked together: the torque,

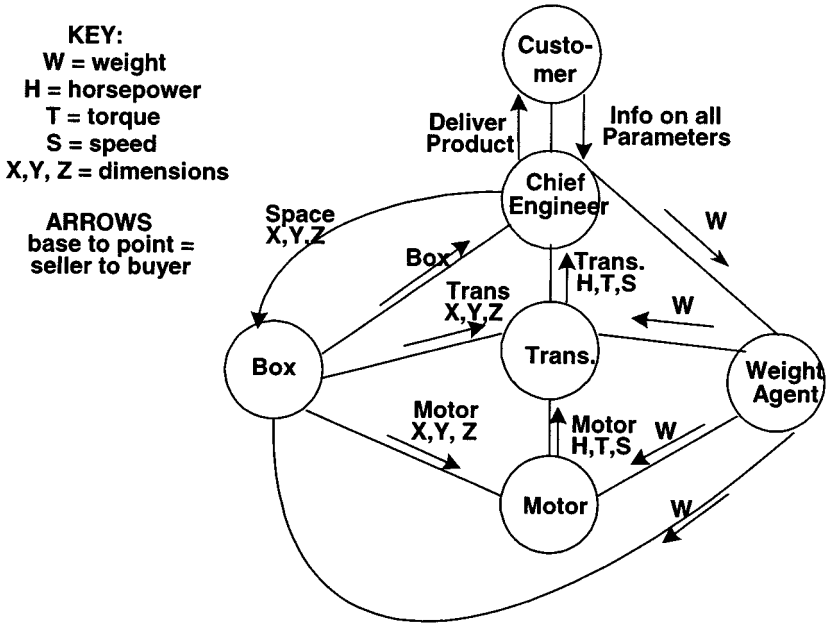

Figure 10. RAPPID ecosystem for the power transmission problem

Table 6. A fragment of a transmission catalog

Lower Bound:		1			800		30	0.125	600
Upper Bound:		1		2000		65		5	1600
Min Value:	6Z439	1	29	601.74	1303.78	13.45	33.62	1/8	107
Max Value:	6Z460	1	86	1784.48	3866.38	39.88	99.71	3	1836
Template:									
		1	29	1784	3866	13	34	1	
1	1	2	3	4	5	6	7	8	9
22	Stock #	Select in or out	RPM @ 1725 input	Min input RPM req'd	Max input RPM req'd	Min output RPM supplied	Max output RPM supplied	Max input HP	Max output torque (in-LB)
1	6Z439	1	29	1784	3866	13	34	1/4	192
1	6Z440	1	29	1784	3866	13	34	1/3	401
1	6Z441	1	29	1784	3866	13	34	1/2	641
1	6Z442	1	29	1784	3866	13	34	3/4	1043

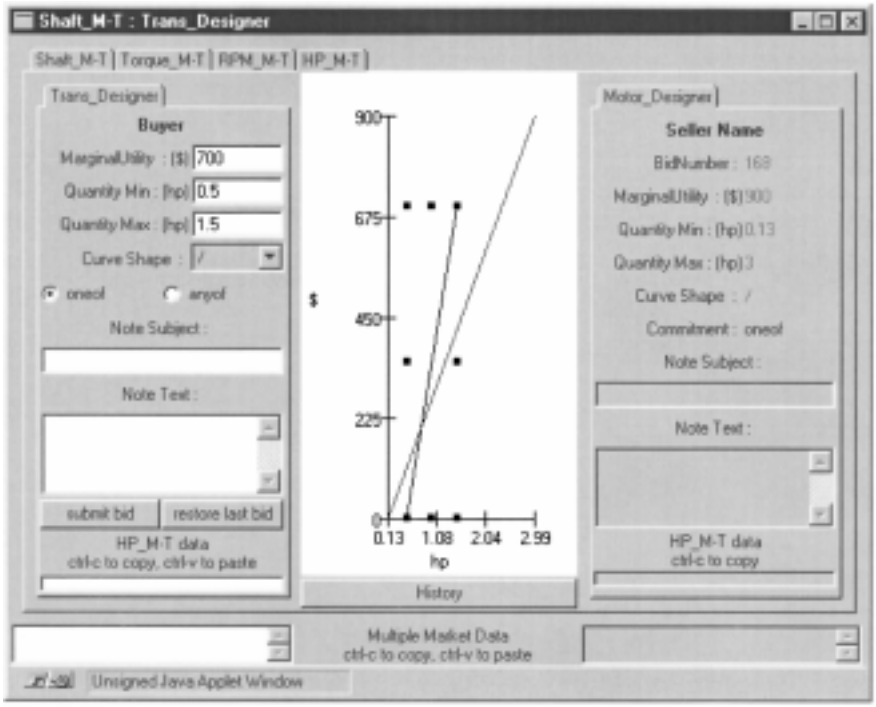

Figure 11. Market interface for the motor-transmission shaft

RPM, and horsepower on the shaft. All three of these markets are accessible as tabs in the interface. Each tab shows the current bid from the buyer (in this case, the transmission) and the seller (the motor), as well as a graphical depiction of the qualitative curve shapes quoted by each component. The suffix "M-T" on the various characteristics indicates that they apply to the Motor-Transmission interface.

In this example, the highest power ranges offered by the motor are beyond the capacity of the transmission, and so can be eliminated. In the range where the components' ranges overlap, the excess of buyer's utility over seller's cost is greatest toward higher power levels. Thus the appropriate action for both participants is to trim their horsepower ranges from the bottom to explore the higher end of the range more thoroughly.

Figure 12 shows the sequence of bids in the horsepower market between motor and transmission in an actual experiment. The motor makes an initial sell bid (A), and is able to narrow this bid (B) without an offer to buy from the transmission, on the basis of bids in other markets. The transmission's offer to buy (C) cuts the range to about a third of its original size. The subsequent sell bids (D, E) by the motor and the final collapse to a single assignment (F) appear unilateral, but in fact are in response to buy bids from the transmission in other related markets (RPM and torque). The shrinkage in the range of assignments to the horsepower

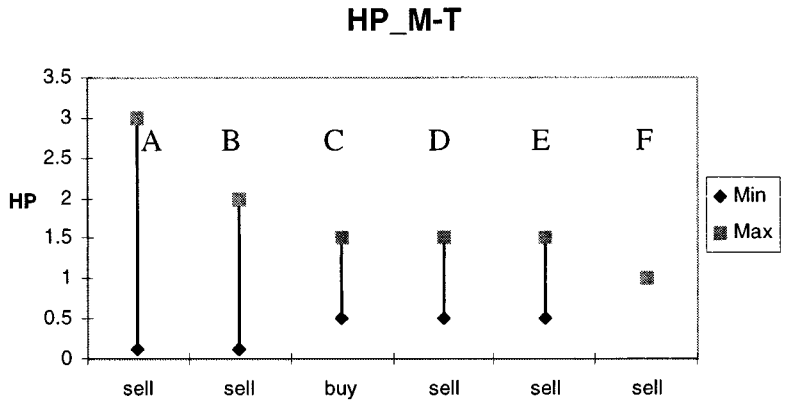

Figure 12. A sample history of RAPPID bids

characteristic illustrates RAPPID's set-based dynamics. The lack of alternation between buying and selling in this single market emphasizes the interaction of different characteristics coupled through the components.

4.2. Experimental results

One requirement for RAPPID 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 component agents and computerized market agents for each characteristic. 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, a team of five humans designing the power transmission mechanism described in the last section, illustrates the potential of the approach.

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 and component weights, 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 $\approx$ 9000) messages and a need for each agent to process $5 * 1518 \approx 7500$ messages (the original configuration and the responses of the other four agents). If we assume something more intelligent 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.

The team converged on a solution after posting only 145 separate bids. In this example, all bids were between pairs of agents. Thus RAPPID's communication

requirements are on the order of 2% (145/9000) of the benchmark, and each agent has to process only an average of $2*145/5 = 58$ messages, or less than 1% (58/7500) of the benchmark. Although we do not claim that RAPPID 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 RAPPID realizes these efficiencies while supporting implicit constraints and set-based reasoning in a hybrid carbon-silicon community.

In another experiment, professional industrial designers at the U.S. Army's Tank and Automotive Command (TACOM) used RAPPID in the conceptual design of a future infantry vehicle. This exercise yielded mixed results.

On the one hand, the team did not converge on a design. Debriefing shows that a major factor was the lack of a clear customer voice to evaluate the marginal utilities of different design options. Designers had difficulty determining how much to offer to pay for different assignments to the design variables of interest to them, and sometimes could not define the cost of providing those assignments. We traced their difficulties to the non-commercial nature of the defense marketplace.

The market dynamics in RAPPID are grounded in an overall utility function owned by the top-level customer for the finished product and the cost functions owned by the bottom-level suppliers, and when either of these is ill defined, the mechanisms break down. In general, the customer for a new military vehicle (the U.S. Department of Defense) is not accustomed to thinking in terms of how much value to attach to an incremental 5 kph in velocity or an incremental 50 km in range. Warfare tends to be an "all-or-nothing" domain that is relatively insensitive to cost arguments. If the technology exists to give a warfighter an extra advantage over an adversary, there is tremendous pressure to provide that advantage, without sacrificing other functions. On the supply side, contracts with government suppliers are often "cost-plus" contracts that fund whatever the supplier spends on permitted expense categories, plus some percentage allowance for profit. Such contracts do not motivate suppliers to reduce their costs. In fact, the more they spend, the more their retained profit. Thus traditional government procurement mechanisms deprived RAPPID of just the market information it needs to guide its designers. There are ways to help even government users develop the necessary utility functions. For example, one can mine the results of previous designs to make the cost/benefit trade-offs explicit. This experiment has caused us to add such tools to our development agenda.

In spite of the technical inconclusiveness of the military exercise, the design team and its management were enthusiastic about RAPPID because its set-based dynamics help them think about the problem in parallel rather than sequentially. Most existing design tools only support point solutions. For example, finite element tools compute stresses on a single physical design, not the range of stresses that will be realized over a range of designs. Professional designers recognize the potential of RAPPID to structure their interactions around the shrinking set paradigm, and with their encouragement we are developing the support mechanisms (such as tools for estimating end-product utilities) that we have found it requires. In addition, reduced defense budgets in the wake of the cold war are raising the importance of cost as a decision variable in defense procurement. Our

design team saw RAPPID's market orientation as a way to help them identify missing cost/benefit information and as a result focus on decisions that could make them more competitive in this new environment.

5. Related work

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., [1, 8]) attempt to apply agent technologies to the long tradition of research into feature-based design (e.g., [13]).

Systems that represent individual *parts* as agents (e.g., [6, 22, 23] 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 [23] focuses on attaching semantic information and simulation models to catalog entries, so that designers can "try before they buy." ACME [6] propagates constraints among individual catalogs (each representing 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 [22], 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. [9, 10] 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 [5], SBD [4], and the blackboard support for RRM [11] 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 [19] recursively divides design tasks into subtasks and assigns them to general problem-solving agents with access to task-specific knowledge bases. SiFA [2] is a variety of the A-Team architecture [20], 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 [9, 10]. 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 [23] 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 (Table 6). In turn, RAPPID's market mechanisms have no counterpart in ActiveCat. The preference functions of ACME [6] and the market formalization of [22] 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.

6. Conclusions and prognosis

RAPPID illustrates how the industrial notion of a life cycle can guide the definition and development of an agent-based application. It supports the widely-held belief that results at the frontiers of research (for example, autonomous agents and market-based programming) hold considerable promise for solving real problems. At the same time, it illustrates the pragmatic need to consider real-world requirements in the industrial domain, notably the need to build systems that interface cleanly with their human users.

RAPPID also illustrates how attention to specific domain requirements can stimulate new research directions. Most research in distributed constraint optimization abstracts away domain features such as causality. Because causality is so intimately linked to design thought, RAPPID developed ways to deal with it, and in the process defined mechanisms that may well be useful in optimizing distributed constraints in other causal domains [18].

RAPPID is a work in process. The next steps center around constructing ancillary tools that we have found necessary to help human users reason about the marginal utilities associated with the end product, and thus sustain the overall economic field that coordinates agents in RAPPID.

Acknowledgments

Portions of this research were sponsored by the Rapid Design Exploration and Optimization (RaDEO) program (formerly MADE) of DARPA (program manager Kevin Lyons), and 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, and Mike Wellman (University of Michigan). Testing of the RAPPID prototype at the U.S. Army's Tank and Automotive Command (TACOM) in Warren, MI was made possible by the enthusiastic support of Gene Baker and Paul Cag of the Technology Integration Division.

References

1. S. Balasubramanian and D. H. Norrie. "A multiagent architecture for concurrent design, process planning, routing, and scheduling." *Concurrent Engineering: Research and Applications*, vol. **4**-1 pp. 7–16, 1996.

2. I. Berker and D. C. Brown. "Conflicts and negotiation in single function agent based design systems." *Concurrent Engineering: Research and Applications*, vol. **4**-1 pp. 17–33, 1996.
3. S. H. Clearwater (Ed.). *Market-Based Control: A Paradigm for Distributed Resource Allocation*, World Scientific: Singapore, 1996.
4. A. Cox, E. Bouchard, and D. Drew. "SBD system design." *Concurrent Engineering: Research and Applications*, vol. **4**-1 pp. 35–46, 1996.
5. 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*, vol. **26**(1) (January) pp. 28–37, 1993.
6. J. D'Ambrosio, T. Darr, and W. Birmingham. "Hierarchical concurrent engineering in a multiagent framework." *Concurrent Engineering: Research and Applications*, vol. **4**-1 pp. 47–57, 1996.
7. R. Hauser and D. Clausing. "The house of quality." *Harvard Business Review*, vol. **66** (May–June) pp. 63–73, 1988.
8. 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.
9. M. Klein. "Supporting conflict management in cooperative design teams." in *Proceedings of 11th International Workshop on DAI*, pp. 155–181, 1992.
10. M. Klein. "Supporting conflict management in cooperative design teams." *Journal on Group Decision and Negotiation*, vol. **2** pp. 259–278, 1993.
11. 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*, vol. **4**-1 pp. 59–71, 1996.
12. R. N. Nagel and R. Dove, *21st Century Manufacturing Enterprise Strategy*, Agility Forum: Bethlehem, PA, 1991.
13. D. Navinchandra, *Exploration and Innovation in Design: Towards a Computational Model*, Springer: New York, NY, 1991.
14. J. Y. C. Pan and J. M. Tenenbaum. "An intelligent agent framework for enterprise integration," in *Artificial Intelligence Applications in Manufacturing*, Famili, Nau, and Kim (Eds.), MIT Press, pp. 349–383, 1992.
15. H. V. D. Parunak. "'Go to the ant': Engineering principles from natural agent systems." *Annals of Operations Research*, vol. **75** pp. 69–101, 1997.
16. H. V. D. Parunak. "Industrial and practical applications of DAI," in *Introduction to Distributed Artificial Intelligence*, G. Weiss (Ed.), MIT Press (forthcoming), 1998.
17. H. V. D. Parunak, J. Sauter, and S. J. Clark. "Toward the specification and design of industrial synthetic ecosystems," in *Intelligent Agents IV: Agent Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence*, Springer: Berlin, pp. 45–59, 1998.
18. 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*), 1998.
19. M. Sobolewski. "Multiagent knowledge-based environment for concurrent engineering applications." *Concurrent Engineering: Research and Applications*, vol. **4**-1 pp. 89–97, 1996.
20. S. N. Talukdar and P. S. de Souza. "Scale efficient organizations," in *Proceedings of IEEE International Conference on Systems Science and Cybernetics*, pp. 1458–1463, 1992.
21. A. Ward, J. K. Liker, J. J. Cristiano, and D. K. S. II. "The second Toyota paradox: How delaying decisions can make better cars faster." *Sloan Management Review*, (Spring) pp. 43–61, 1995.
22. M. P. Wellman. "A computational market model for distributed configuration design." *AI-EDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. **9**-2 pp. 125–134, 1995.
23. P. Will. "Active Catalogs." http://www.isi.edu/active-catalog/, 1997.