To appear in *AI EDAM*, 1995.

A Computational Market Model for Distributed Configuration Design

Michael P. Wellman

University of Michigan, AI Laboratory
1101 Beal Avenue
Ann Arbor, MI 48109-2110
wellman@engin.umich.edu

Abstract

This paper presents a precise market model for a well-defined class of distributed configuration design problems. Given a design problem, the model defines a computational economy to allocate basic resources to agents participating in the design. The result of running these "design economies" constitutes the market solution to the original problem. After defining the configuration design framework, I describe the mapping to computational economies and our results to date. For some simple examples, the system can produce good designs relatively quickly. However, analysis shows that the design economies are not guaranteed to find optimal designs, and we identify and discuss some of the major pitfalls. Despite known shortcomings and limited explorations thus far, the market model offers a useful conceptual viewpoint for analyzing distributed design problems.

This page intentionally blank

1. Introduction

The current dramatic expansion in network technology and infrastructure, including the explosive growth of the internet, will soon see an inevitable proliferation of automated, distributed, information agents (for example, of the sort described by (Etzioni and Weld 1994)). With these advances, opportunities and incentives for the decentralization of design activities are rapidly emerging. Computational support for distributed design collaboration presents a variety of new challenges (Cutkosky et al. 1993; Edmonds et al. 1994; Reddy et al. 1993). In particular, the specialization of design expertise suggests a future where teams form *ad hoc* collaborations dynamically and flexibly, according to the most opportunistic connections. Centralized coordination or control is anathema in this environment. Instead we seek general decentralized mechanisms which respect the local autonomy of the agents involved, yet at the same time facilitate results (in our case, designs) with globally desirable qualities.

Design of complex artifacts can be viewed as fundamentally a problem of *resource allocation*. We generally have numerous performance and functional objectives to address, with many options for trading among these objectives and for furthering these objectives in exchange for increased cost. When the design problem is distributed, then these tradeoffs are not only across objectives, but also across agents (human or computational) participating in the design. Typically each agent is concerned with a subset of the components or functions of the artifact being designed, and may individually reflect a complex combination of the fundamental objectives.

Consider a hyper-simplified scenario in aircraft design. (We choose this not as a serious exemplar of the methodology, but merely as an evocative illustration of the concept of decentralized resource allocation.) Suppose we have separate agents responsible for the airfoil, engines, navigation equipment, etc. Suppose also that we have a target for the aircraft's total weight. Since total weight is the sum of the weights of its parts, we might consider allocating *a priori* each agent a slice of the "weight budget". This approach has several serious problems. First, if we do not choose the slices correctly, then it could be that one of the agents makes extreme compromises (e.g., an underpowered engine or expensive exotic metals for the fuselage) while the others could reduce weight relatively easily (e.g., replace the lead in the wings). Second, it will typically be impossible to determine good slices in advance, because the appropriate allocation will depend on other design choices. For example, depending on the position of the wings, reducing body weight while maintaining structural soundness may be more or less expensive. Or if we have a more powerful engine, then it may be that extra total weight can be accommodated, and the fixed budget itself was not realistic.

Rather than allocate fixed proportions of resources, we desire an approach that dynamically adjusts the allocation as the design progresses. One way to achieve this sort of behavior is via a negotiation and trading process. For example, if the engine agent would benefit substantially from a slight increment in weight, it might offer to trade some of its drag allowance to the airfoil agent for a share of the latter's weight allocation. If there is enough of this kind of trading going on, then it makes sense to establish *markets* in the basic resources—weight, drag, etc. Then we can view the entire system as a sort of economy devoted to allocating resources for the design. We call this system the *design economy*.

In this paper, we present a precise market model for a well-defined class of distributed configuration design problems. Given a design problem, our model defines a computational economy to solve the design problem, expressed in terms of concepts from microeconomic theory. We can then implement these economies in our WALRAS system for market-oriented programming (Wellman 1993), thus producing a runnable design economy.

The following sections describe the configuration design framework, the mapping to computational economies, and preliminary results. For some simple examples, the system can produce good designs relatively quickly. However, analysis shows that the design economies are *not* guaranteed to find optimal designs, and we identify and discuss some of the major pitfalls. We argue that most of these problems are inherent in decentralization generally, not in the design economy *per se*, and moreover that the market model offers useful concepts for engineering the configuration of a variety of multiagent tasks.

2. Distributed Configuration Design

We adopt a standard framework for configuration design (Mittal and Frayman 1989). Our specific formulation follows (Darr and Birmingham 1994), and covers many representative systems (Balkany et al. 1993). Design in this framework corresponds to selection of *parts* to perform *functions*. More precisely, a design problem consists of:

- A set of attributes, $A_1, \ldots, A_n$, and their associated domains, $X_1, \ldots, X_n$.
- A set of available parts, P, where each part $p \in P$ is defined by a tuple of attribute-value pairs.
- A distinguished subset of attributes, F, the *functions*. The domain of a function attribute is the set of parts that can perform that function.
- A set of constraints, R, dictating the allowable part combinations (either directly or indirectly via constraints on the attributes).
- A utility function, $u: X_1 \times \cdots \times X_n \to \Re$, ranking the possible combinations of attribute values by preference.

A *design* D is an assignment of parts to functions. Thus, the role of functions in this framework is to define when a design is complete. D is *feasible* if it satisfies R. The *valuation* of a design, $val(D)$, is the assignment to attribute values induced by the assignment of parts to functions. Typically, this is just the sum over the respective attribute values of a design's component parts. Finally, a feasible design D is *optimal* if for all feasible designs D', $u(val(D)) \geq u(val(D'))$.

In addition, it is often useful to organize the parts into *catalogs*. Let catalog i, $1 \leq i \leq m$, consist of parts C_i, where $C_1, \ldots, C_m$ is a partition of P. A catalog design includes at most one part per catalog.[1] In distributed design, catalogs are the units of distribution: each catalog agent selects a part to contribute or chooses not to participate.

In this general form, the design task is clearly intractable (it subsumes general constraint satisfaction and optimization). Tractability can be obtained only by imposing restrictions on R and u. Instantiations of this framework adopt specialized constraint languages, and often relax the requirement of optimality. We shall accept comparable compromises in our mapping to the market model.

3. Computational Markets

The market model we adopt is a computational realization of the most generic, well-studied theoretical framework, that of general equilibrium theory (Hildenbrand and Kirman 1976; Shoven and Whalley 1992). General equilibrium is concerned with the behavior of a collection of interconnected markets, one market for each good. A general-equilibrium system consists of:

---

[1]In the degenerate case where each catalog lists only one part, catalog design reverts to the situation without catalogs. Typically catalogs will correspond to functions, although this is not a requirement. Indeed, this definition places no restriction on the number of functions any part may implement.

- a collection of goods, $g_1, \ldots, g_n$, and
- a collection of agents, divided into two types:
  - *consumers*, who simply exchange goods, and
  - *producers*, who can transform some goods into others.

A consumer is defined by (1) a *utility function*, which specifies its relative preference for consuming a bundle of goods $\langle x_1, \ldots, x_n \rangle$, and (2) an *endowment* $\langle e_1, \ldots, e_n \rangle$ of initial quantities of the goods. Consumers may trade all or part of their initial endowments in exchange for quantities of the other goods. All trades must be executed at the established market prices, $\langle p_1, \ldots, p_n \rangle$. A consumption bundle is feasible for a consumer if and only if it satisfies the *budget constraint*,

$$\sum_{i=1}^{n} x_i p_i \leq \sum_{i=1}^{n} e_i p_i ,$$

which says that a consumer may spend only up to the value of its endowment. The decision problem faced by a consumer is to maximize its utility subject to the budget constraint.

The other class of agents, producers, do not consume goods, but rather transform various combinations of some goods (the inputs) into others (outputs). The feasible combinations of inputs and outputs are defined by the producer's *technology*. For example, if the producer can transform one unit of g_1 into two units of g_2, with constant returns to scale, then the technology would include the tuples $\{\langle -x, 2x, 0, \ldots, 0 \rangle \mid x \geq 0\}$. This production would be profitable as long as $p_1 < 2p_2$. The producer's decision problem is to choose a production activity so as to maximize *profits*, the difference between revenues (total price of output) and costs (total price of input). Note that a producer does not have a utility function.

In the computational market price system we use, WALRAS, we can implement consumer and producer agents and direct them to bid so as to maximize utility or profits, subject to their own feasibility constraints. The system derives a set of market prices that balance the supply and demand for each good. Since the markets for the goods are interconnected (due to interactions in both production and consumption), the price for one good will generally affect the demand and supply for others. W ALRAS adjusts the prices via an iterative bidding protocol, until the system reaches a *competitive equilibrium* (see (Wellman 1993) for details), i.e., an allocation and set of prices such that (1) consumers bid so to maximize utility, subject to their budget constraints, (2) producers bid so to maximize profits, subject to their technological possibilities, and (3) net demand is zero for all goods.

An economy in competitive equilibrium has the desirable property of Pareto Optimality: it is not possible to increase the utility of one agent without decreasing that of other(s). Moreover, for any Pareto optimum (i.e., any admissible allocation), there is some corresponding set of initial endowments that would lead to this result. Given certain technical restrictions on the producer technologies and consumer preferences, equilibrium is guaranteed to exist and the system converges to it. Specifically, if technologies and preferences are smooth and strictly convex, then a competitive equilibrium exists (see (Varian 1984) for formal treatment). If in addition, the derived demands are such that increasing the price for one good does not decrease the demand for others, then the equilibrium is unique and iterative bidding process is guaranteed to converge to it (Arrow and Hurwicz 1977; Milgrom and Roberts 1991).

3.1. The Design Economy

We next proceed to instantiate the general-equilibrium framework for our distributed configuration design problem. As noted above, our purpose in applying the economic mechanism is to provide a principled basis for resolving tradeoffs across distributed design agents. By grouping the agents together in a single economy, we establish a *common currency* linking the agents' local demands for available resources. Using this common currency, the market can (at least under the circumstances sketched above) efficiently allocate resources toward their most productive use with minimal communication or coordination overhead. The fact that all interaction among agents occurs via exchange of goods at standard prices greatly simplifies the design of individual agents, as they can focus on their own attributes and the parameters of their local economic problems. In addition, the price interface provides a straightforward way to influence the system's behavior from the outside—by setting the relative prices of exogenously derived resources and performance attributes. And similarly, the prices within the system can be interpreted from the outside in terms of economic variables that are meaningful to those participating in the design process.

3.2. Market Configuration

In general, to cast a distributed resource-allocation problem in terms of a computational market, one needs to specify

- the goods (commodities) traded,
- the consumer agents trading and ultimately deriving value from the goods,
- the producer agents, with their associated technologies for transforming some goods into other goods, and

- the agents' bidding behavior.

There are two classes of goods we consider in the problem of distributed design. First, we have basic resource attributes, such as weight and drag in the fanciful aircraft example above. These are resources required by the components in order to realize the desired performance, but are limited or costly or both. Generally, we desire to minimize our overall use of resources. The second class of goods are performance attributes, such as engine thrust or leg room in a passenger aircraft. We also include the function attributes in this category. Performance attributes measure the capabilities of the designed artifact, and we typically desire to maximize these.

In terms of our framework for distributed configuration design, both resource and performance attributes are kinds of attributes. Functions can be viewed as a subclass of performance attributes. So, the first step in mapping a design problem to our market model is to identify the goods with the attributes. Although it is not strictly necessary to distinguish the resource from performance attributes, we do so for expository reasons, as it facilitates intuitive understanding of the flow of goods in the design economy.

The remaining steps are to identify the agents and define their behavior. As mentioned above, it is typical in multiagent design to allocate individual agents responsibility for distinct components or functions. Within our distributed design scheme, these agents correspond to catalogs. In a distributed constraint-satisfaction formulation of the problem (Darr and Birmingham 1994), catalog agents select a part to contribute to the overall design. The chosen part entails a particular pattern of resource usage and performance, as specified in its associated attribute vector.

Correspondingly, in the design economy, each part in the catalog is associated with a vector of resource and performance *goods*. The resources can be interpreted as input goods, and the performance goods as output. In this view, the *catalog producer* is an agent that transforms resources to performance. For example, the engine agent in our aircraft design transforms *weight* and *drag* (and *noise*, etc.) into *thrust*. The particular combinations of weight/drag/thrust/… represented by the various engine models constitute this producer's *technology*. Thus, to specify a catalog producer's technology (and that's all there is to specify for a producer), we simply form a set of the attribute-value tuples characterizing each part. We then go through and negate the values for the resource goods, leaving the values for performance goods intact.[2]

[2] This assumes that all attributes are measured in increasing resource usage or increasing performance. If not, we would simply rescale the values in advance.

An economy with only producers would have no purpose. To ground the system, we define a consumer agent, conceptually the end user or customer for the overall design.[3] The consumer is endowed with the basic resource goods. So for example, we will initialize the system by endowing the consumer with a total weight, typically an amount greater than the heaviest conceivable aircraft defined by combination of the heaviest available parts. The idea is that the consumer then (effectively) sells this weight to the various catalog agents, in exchange for performance goods like thrust.

The consumer has preferences over the overall design attributes, as specified by its utility function. This function is essentially equivalent to the function u defined in the general framework for configuration design. There is one syntactic modification, however. In the original specification, u was increasing in the performance attributes and decreasing in the resource attributes. The consumer's utility function, in contrast, must be increasing in all attributes. To ensure this, we define the "consumption" of a resource good as the amount left over after the consumer sells part of its endowment. Thus, if the consumer's endowment of weight is w and the total weight of the aircraft (sum of the weight of the parts) is w', then the effective weight consumed is $w - w'$. If the consumer prefers lighter airplanes, then it prefers to "consume" as much weight as possible.

Thus far, we have accounted for all elements of the configuration design framework except for the constraints (recall that the functions are a subset of performance attributes). We are able to map some kinds of constraints into this framework, but not others. The simplest type of constraint is an upper bound on the total usage of a particular resource (e.g., the total noise cannot exceed FAA regulations). To capture this kind of constraint, we simply endow the consumer with exactly the upper bound. Since the consumer is the only source of resource goods in the economy, this effectively restricts the combined choices of the producers. Some other kinds of constraints can also be handled by defining new intermediate goods. This is best illustrated by example in the next section. (Unfortunately, we do not have a compact characterization of the constraints that can be easily mapped in this way.)

Finally, we must note an implicit assumption underlying the mapping described above. In order for the resource and performance goods to be reasonably traded among the consumer and producers, it must be the case that the total resource usage (resp. performance achieved) does

---

[3] If there are more than one class of users or customers with distinct preferences, then we could introduce several consumer agents. The underlying computational market accommodates any number of agents, but our design economies thus far have employed only one.

indeed correspond to the sum of the parts. In other words, the *val* function described above must be additive (but note that the utility function *u* need not be). There may be some encoding tricks to get around this restriction in some cases, but the basic design economy assumes this property.

3.3. An Example

To illustrate the mapping from configuration design problems to design economies, we carry out the exercise for an example presented by (Darr and Birmingham 1994). The problem is a very simplified computer configuration design. In the example, we have two functions to satisfy: *processing*, and serial *port*. Our design criteria are *dollar* cost and reliability, as measured in failures-per-million-hours (*fpmh*). These criteria are represented in the design economy as resource goods. There are no performance goods, except for the two functions mentioned. There is also another good, *memory* access, which is conceived here not as an overall performance attribute but as an intermediate good enabling the *processing* function.[4]

We also have three constraints. The total dollar cost must not exceed 11, the total failure rate must not exceed 10 fpmh, and the RAM must be at least as fast as the CPU's memory access. The first two of these constraints are captured simply by endowing the consumer with 11 dollars and 10 fpmh. The consumer's utility function specifies preferences over these two goods, as well as the functions *processing* and *port*. Representing the functions as utility attributes rather than constraints deviates from the original framework somewhat by allowing the user the flexibility to trade off functionality for performance or resource savings.

There are 14 available parts, organized into three catalogs. The CPU catalog contains four CPU models, each of which supplies the *processing* function. Five serial ports are available to support the *port* function, and five RAM models can supply *memory* access. The complete CPU catalog is presented in Table 1.

CPU	process	dollars	fpmh	memory
CPU1	1	6	1	40
CPU2	1	4	7	140
CPU3	1	2	2	40
CPU4	1	1	5	30

Table 1: CPU catalog.

[4]We chose this interpretation specifically to illustrate the notion of intermediate goods; in the next section we simplify the model to treat *memory* as a function.

The catalogs are converted into technologies simply by listing the attribute tuples as good tuples, possibly after some rescaling. For example, the CPU agent's technology is:

$$\{\langle 1,-6,-1,-1/40\rangle,\langle 1,-4,-7,-1/140\rangle,\langle 1,-2,-2,-1/40\rangle,\langle 1,-1,-5,-1/30\rangle,\langle 0,0,0,0\rangle\}.$$

This means that the agent has five options, with the net good productions listed. The first option is to produce an output of 1 unit of processing (by default, functions take values in $\{0,1\}$) from an input of 6 *dollars*, 1 *fpmh*, and 1/40 *memory* access units. This last input requires some explanation. In order to represent the constraint that CPU be at least as fast as RAM, we define the intermediate good *memory* access, which is an output of RAM and an input of CPU. The constraint is then enforced by requiring that CPU buy enough speed from RAM. Since the values listed in the catalog above are memory access times, we invert them to convert to speed units.

Finally, the tuple of zeros is an element of every technology. An agent always has the option to produce nothing, in which case its resource usage is zero (as are its profits).

Similar technologies are generated for the RAM catalog (input *dollar* and *fpmh*, output *memory*) and serial port (input *dollar* and *fpmh*, output *port*), as shown in Table 2.

RAM	$	fpmh	memory	**Port**	$	fpmh	port
null	0	0	0	null	0	0	0
RAM 1	–6	–2	1/40	SP1	–6	–1	1
RAM 2	–3	–3	1/10	SP2	–3	–3	1
RAM 3	–2	–6	1/35	SP3	–2	–4	1
RAM 4	–2	–4	1/30	SP4	–1	–8	1
RAM 5	–1	–7	1/35	SP5	–1	–6	1

Table 2: Technologies for RAM and Port catalog producers.

The configuration of the design economy is best described in terms of flow of goods, depicted in Figure 1. The consumers supply the basic resources, which are transformed by the producers to performance goods (as well as intermediate resources needed by other producers), which are ultimately valued by the consumer.

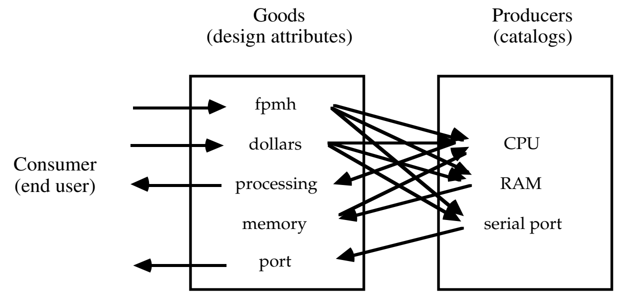

Figure 1: Flow of goods in the example design economy.

3.4. Agent Behavior

All that remains to specify our mapping is to define the behavior of the various agents. The consumer's problem is to set demands maximizing utility, subject to the budget constraint. This defines a constrained optimization problem, parametrized by current going prices.

In our example, we adopt a standard special functional form for utility, one exhibiting constant elasticity of substitution (CES) (Varian 1984). The CES utility function is

$$u(x_1,\ldots,x_n)=\left(\sum_{i=1}^{n}\alpha_i^{1-\rho}x_i^{\rho}\right)^{1/\rho},$$

where the α_i are good coefficients and ρ is a generic substitution parameter. One virtue of the CES utility function is that there is a closed-form solution to the optimal demand function,

$$x_i(p_1,\ldots,p_n)=\frac{\alpha_i\sum_{j=1}^{n}p_je_j}{p_i^{1/(1-\rho)}\sum_{j=1}^{n}\alpha_jp_j^{\rho/(\rho-1)}}.$$

The consumer can simply evaluate this function at the going prices whenever prompted by the bidding protocol. In WALRAS, bids are actually demand *curves*, where the price p_i of the good i being bid varies while the other prices are held fixed. CES consumers transmit a closed-form description of this curve, with p_i the dependent variable.

The producers' bidding task is also quite simple. Catalog producers face a discrete choice among the possible component instances, each providing a series of values for goods corresponding to resource and performance attributes. Let $\left(x_1^j,\ldots,x_n^j\right)$ be the vector of quantities representing part j in the producer's catalog, and π^j denote the profitability of part j, as a function of prices:

$$\pi^j(p_1,\ldots,p_n) = \sum_{i=1}^{n} p_i x_i^j .$$

All catalogs include the null part 0, where $x_i^0 = 0$ for all i. Given a set of prices, the producer selects the most profitable part:

$$j^*(p_1,\ldots,p_n) = \arg\max_j \pi^j(p_1,\ldots,p_n).$$

Just as for consumers, when a producer bids on good i, it specifies a range of demands for all values of p_i (the price of good i), holding all other prices fixed, which is determined by the optimal part for that price,

$$x_i(p_i) = x_i^{j^*(\bar{p}_1,\ldots,p_i,\ldots\bar{p}_n)}.$$

When all prices but one are fixed, we can simplify the profit function as follows:

$$\pi^j(\bar{p}_1,\ldots,p_i,\ldots,\bar{p}_n) = p_i x_i^j + \sum_{k\neq i} \bar{p}_k x_k^j = p_i x_i^j + \bar{\beta}^j,$$

where the parameter $\bar{\beta}^j$ depends on part j and the fixed prices $\bar{p}_k,\ k \neq i$. For example, part j is more profitable than j' if and only if

$$p_i x_i^j + \bar{\beta}^j > p_i x_i^{j'} + \bar{\beta}^{j'},$$

or equivalently (for $x_i^j > x_i^{j'}$),

$$p_i > \frac{\bar{\beta}^{j'} - \bar{\beta}^j}{x_i^j - x_i^{j'}}.$$

It is clear from the above that parts with higher values of x_i^j become maximally profitable, if ever, at higher prices p_i. Therefore, one way to calculate $j^*(\bar{p}_1,\ldots,p_i,\ldots,\bar{p}_n)$ is to sort the parts in increasing order of x_i^j and then use the inequality above to derive the price threshold between each adjacent pair.[5] These thresholds dictate the optimal part at any p_i (assuming the other prices fixed), and the associated demand $x_i^{j^*}$. Thus, the form of the producer's demand is a step function, defined by a set of threshold prices where different components become optimal.

[5]This includes the null part, with $x_i^0 = \bar{\beta}^0 = 0$. If the threshold decreases, then the intervening part can never be optimal (it is not on the convex hull) for any price of good i, and can be skipped.

4. Discussion

4.1. Results

We have run the design economy on a few simple examples, including the one presented above modified to treat *memory* as a function rather than an intermediate good. On this example, it produces the optimal design in three bidding cycles, although it never (as long as we have run it) reaches a price equilibrium. After the third cycle, the prices continue to fluctuate, although never enough to cause one of the catalog producers to change its optimal part. Unfortunately, we have no way of detecting with certainty that the part choices are stable.

Although we have yet to run systematic experiments, we have observed similar behavior on a variety of small examples. We have also encountered examples where the design economy does not appear to converge on a single design, or it converges on a design far from the global optimum. This is not surprising, as the general class of design economies producible by the mapping specified above does not satisfy the known conditions for existence of competitive equilibrium. Indeed, the conditions are never strictly satisfied by design economies generated as described, because the discreteness of catalogs violates convexity.

4.2. Limitations

The analytical theories underlying the market approach make it easy to characterize the limitations of the approach. To guarantee optimal results, strictly speaking, we require that the catalogs represent convex sets of attribute vectors, a condition that is not typically satisfied in configuration design problems. Although the system sometimes produces good or even optimal designs despite violating the conditions, we currently lack an adequate model to predict the system's performance on novel problems.

In other (non-design) applications of the market-based approach, we have found that the method scales very well with the number of agents. Scaling the number of goods can be a more serious problem, particularly if the goods are highly interconnected.

The approach as presented here is also fundamentally limited in applicability to a subclass of configuration design problems. While configuration design is a standard framework in the design-automation literature, there are clearly many design domains (including the fanciful aircraft example), where the framework does not fit. Moreover, even within configuration design, we require that overall resource usage be decomposed into the sum of resource usage of the parts, and limit the form of constraints that can be accommodated in the market system.

It is important to note, however, that many of these difficulties are inherent in the problem of decentralization, and not particular to the market approach. Indeed, one of the main virtues of this approach is that, compared to others, its limitations are transparent, and amenable to analysis.

4.3. Extensions

As mentioned above, designs produced by the market model are not guaranteed to be optimal or even feasible, due to the discreteness (and other non-convexities) of the problem. They are more likely to be local optima, where no single change of policy by one of the producer agents will constitute an improved design. In current work, we are attempting to characterize the performance of the scheme for special cases. For those cases where perfectly competitive equilibria do not exist, we are investigating the possibility of relaxing the competitiveness assumption. In addition, we are looking into hybrid schemes that use the market to bound the optimal value by computing the global optimum for a smooth and convex relaxation of the original problem. This is analogous to branch-and-bound schemes that make use of regular linear-programming algorithms for integer-programming problems. In this and other approaches, we expect the economic price information exploited by the market-oriented approach to lead to more rational tradeoffs in the distributed design process.

We are also exploring combinations of market-based and constraint-based methods, where we use general constraints (including those inexpressible in the market model) to prune the catalogs before running the design economy. This might be accomplished using an auxiliary constraint network (Bowen and Bahler 1993), or one built on the existing agent structure (Darr and Birmingham 1994). A hybrid approach interleaving market-directed search for good designs with constraint reasoning uses each method to compensate for inadequacies in the other.

Another specific modification to the underlying model we are exploring is to maintain intervals on prices rather than specific points. For any given interval of prices, the producer and consumer agents in the computational economy can derive their admissible design choices. Given a collection of agents bids in the form of sets of admissible choices, the system can compute a new price interval. When the price converges to a single point, the behavior of the individual agents will correspond to a unique overall design. This too, meshes well with general constraint operations, as pruning based on price bounds complements pruning based on physical constraints.

5. Economics and Distributed AI

5.1. Why Economics?

To coordinate a set of largely autonomous agents, we usually seek mechanisms that (A) produce globally desirable results, (B) avoid central coordination, and (C) impose minimal communication requirements. In addition, as engineers of such systems, we also prefer mechanisms that (D) are amenable to theoretical and empirical analysis. In human societies, advocates of market economies argue that B is essential (for various reasons) and that market price systems perform well on criterion A because they provide each agent with the right incentives to further the social good. Because prices are a very compact way to convey this incentive information to each agent, we can argue that price systems also satisfy C (Koopmans 1970). For these reasons, some have found the market economy an inspiring social metaphor for approaches to coordinating distributed computational agents. Thus, we sometimes see mechanisms and protocols appealing to notions of negotiation, bidding, or other economic behavior.

Criterion D is also a compelling motivation for exploring economic coordination mechanisms in a computational setting. The problem of coordinating multiple agents in a societal structure has been deeply investigated by economists, resulting in a large body of concepts and insights, as well as a powerful analytical framework. The phenomena surrounding social decision making have been studied in other social science disciplines as well, but economics is distinguished by its focus on three particular issues:

Resource allocation. The central aspect of the outcome of the agents' behavior is the allocation of resources and distribution of products throughout the economy.

Rationality abstraction. Most of microeconomic theory adopts the assumption that individual agents are *rational* in the sense that they act so as to maximize utility. This approach is highly congruent with much work in Artificial Intelligence, where we attempt to characterize an agent's behavior in terms of its knowledge and goals (or more generally, beliefs, desires, and intentions). Indeed, this *knowledge level* analysis requires some kind of rationality abstraction (Newell 1982), and is perhaps even implicit in our usage of the term *agent*.

Decentralization. The central concern of economics is to relate decentralized, individual decisions to aggregate behavior of the overall society. This is also the concern of Distributed AI as a computational science.

5.2. Related Work

In addition to my own prior work in market-oriented programming (Wellman 1993; Wellman 1995), there have been several other efforts to exploit markets for distributed computation (Clearwater 1995). Most famous in AI is the contract net (Davis and Smith 1983), but it is only recently that true economic mechanisms have been incorporated in that framework (Sandholm 1993). There have been interesting recent proposals for incorporating a range of economic ideas in distributed allocation of computational resources (Drexler and Miller 1988; Miller and Drexler 1988), as well as some actual experiments along these lines (Cheriton and Harty 1993; Kurose and Simha 1989; Waldspurger et al. 1992). However, there are no other fully computational market models for distributed design of which we are aware.

This approach also shares some conceptual features with other agent-based frameworks for distributed computing (Genesereth and Ketchpel 1994; Wiederhold 1992), particularly Shoham's approach to "agent-oriented programming" (Shoham 1993). Where Shoham defines a set of interactions based on speech acts, we focus exclusively on the economic actions such as exchange and production. In both approaches (as well as some others (Huberman 1988)), the underlying idea is to get an improved understanding of a complex computational system via social constructs.

Finally, there is a large literature on decomposition methods for mathematical programming problems, which could perhaps be applied to distributed design. Many of these and other distributed optimization techniques (Bertsekas and Tsitsiklis 1989) can themselves be interpreted in economic terms, using the close relationship between prices and Lagrange multipliers. The main distinction of the approach advocated here is conceptual. Rather than taking a global optimization problem and decentralizing it, our aim is to provide a framework that accommodates an exogenously given distributed structure.

6. Conclusions

The main contribution of this work is a precise market model for distributed design, covering a significant class of configuration design problems. The model has been implemented within a general environment for defining computational market systems. Although we have yet to perform extensive, systematic experiments, we have found that the design economy produces good designs on some simple problems, but breaks seriously on others. Analysis is underway to characterize the cases where it works. It cannot be expected to work universally, as these are known intractable optimization problems, and distributing the problem only makes things worse.

In the long run, by embedding economic concepts within our design algorithms, we facilitate the support of actual economic transactions that we expect will eventually be an integral part of networks for collaborative design and other inter-organizational interactions. Many other technical problems will need to be solved (involving security for proprietary information and bidding protocols, for example) before this is a reality, but integrating economic ideas at the conceptual level is an important first step.

Acknowledgments

Special thanks to Bill Birmingham and Tim Darr for assistance with the distributed design problem and examples. John Cheng, Daphne Koller, and some anonymous referees provided useful comments on the paper and the work.

References

Arrow, K. J., and L. Hurwicz, Ed. (1977). Studies in Resource Allocation Processes. Cambridge, Cambridge University Press.

Balkany, A., W. P. Birmingham, and I. D. Tommelein (1993). An analysis of several configuration design systems. *AI EDAM* **7**: 1-17.

Bowen, J., and D. Bahler (1993). Constraint-based software for concurrent engineering. *Computer* **26**(1): 66-68.

Cheriton, D. R., and K. Harty (1993). A market approach to operating system memory allocation. Stanford University Department of Computer Science.

Clearwater, S, Ed. (1995). Market-Based Control: A Paradigm for Distributed Resource Allocation. World Scientific.

Cutkosky, M. R., R. S. Englemore, R. E. Fikes, et al. (1993). PACT: An experiment in integrating concurrent engineering systems. *Computer* **26**(1): 28-37.

Darr, T. P., and W. P. Birmingham (1994). An Attribute-Space Representation and Algorithm for Concurrent Engineering. University of Michigan. Revised version to appear in *IEEE Expert*.

Davis, R., and R. G. Smith (1983). Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence* **20**: 63-109.

Drexler, K. E., and M. S. Miller (1988). Incentive engineering for computational resource management. *The Ecology of Computation* Ed. B. A. Huberman. North-Holland. 231-266.

Edmonds, E. A., L. Candy, R. Jones, et al. (1994). Support for collaborative design: Agents and emergence. *Communications of the ACM* **37**(7): 41-47.

Etzioni, O., and D. Weld (1994). A softbot-based interface to the internet. *Communications of the ACM* **37**(7): 72-76.

Genesereth, M. R., and S. P. Ketchpel (1994). Software agents. *Communications of the ACM* **37**(7): 48-53.

Hildenbrand, W., and A. P. Kirman (1976). *Introduction to Equilibrium Analysis: Variations on Themes by Edgeworth and Walras*. Amsterdam, North-Holland Publishing Company.

Huberman, B. A., Ed. (1988). The Ecology of Computation. North-Holland.

Koopmans, T. C. (1970). Uses of prices. *Scientific Papers of Tjalling C. Koopmans* Ed. . Springer-Verlag. 243-257.

Kurose, J. F., and R. Simha (1989). A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers* **38**: 705-717.

Milgrom, P., and J. Roberts (1991). Adaptive and sophisticated learning in normal form games. *Games and Economic Behavior* **3**: 82-100.

Miller, M. S., and K. E. Drexler (1988). Markets and Computation: Agoric Open Systems. *The Ecology of Computation* Ed. B. A. Huberman. North-Holland. 133-176.

Mittal, S., and F. Frayman (1989). Towards a generic model of configuration tasks. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, Morgan Kaufmann.

Newell, A. (1982). The Knowledge Level. *Artificial Intelligence* **18**: 87-127.

Reddy, Y. V. R., K. Srinivas, V. Jagannathan, et al. (1993). Computer support for concurrent engineering. *Computer* **26**(1): 12-16.

Sandholm, T. (1993). An implementation of the contract net protocol based on marginal cost calculations. *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC, AAAI Press.

Shoham, Y. (1993). Agent-Oriented Programming. *Artificial Intelligence* **60**: 51-92.

Shoven, J. B., and J. Whalley (1992). *Applying General Equilibrium*. Cambridge University Press.

Varian, H. R. (1984). *Microeconomic Analysis*. New York, W. W. Norton & Company.

Waldspurger, C. A., T. Hogg, B. A. Huberman, et al. (1992). Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering* **18**: 103-117.

Wellman, M. P. (1993). A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* **1**(1): 1-23.

Wellman, M. P. (1995). Market-oriented programming: Some early lessons. *Market-Based Control: A Paradigm for Distributed Resource Allocation*, Ed. S. Clearwater. World Scientific.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *Computer* **26**(3): 38-49.