

# Some Issues in the Design of Market-Oriented Agents

Tracy Mullen and Michael P. Wellman

Artificial Intelligence Laboratory, University of Michigan  
1101 Beal Avenue, Ann Arbor, MI 48109-2110 USA

{mullen,wellman}@umich.edu

**Abstract.** In a computational market, distributed market agents interact with other agents primarily through the exchange of goods and services. Thanks to a well-developed underlying economic framework, we can draw on a rich source of analytic tools and theoretical techniques for designing individual agents and predicting aggregate behavior. For many narrowly scoped static problems, design of a computational market is relatively straightforward. We consider some issues that arise in attempting to design computational economies for broadly scoped, dynamic environments. These issues include how to specify the goods and services being exchanged, how these market-oriented agents should set their exchange policies, and how computational market mechanisms appropriate for idealized environments can be adapted to work in a larger class of non-ideal environments.

## 1 Introduction

Approaches to resource allocation in distributed systems can be bounded by two extremes. At one end (the “software engineering” approach), we write a specification of the required behavior of the system in terms of software assets, architectural knowledge, and resource constraints. If our synthesized distributed system is guaranteed to satisfy these constraints, it is because we have built in a set of resource commitments that will lead to that result under the specified assumptions. At the other end of the spectrum (the “distributed agent” approach), rather than specify the required behavior under given resources, we design protocols or mechanisms [20] whereby a set of relatively autonomous software modules can, through their interaction and run-time allocation of resources, achieve some desirable aggregate behavior using available system resources. This second approach has several advantages. It is more flexible, allowing for a dynamically changing configuration of agents. It promotes modularity and simplicity of design since agents must make decisions based only on local, rather than global, information. It is also a natural model when agents are designed, implemented, or owned by different companies and individuals. However, the distributed agent approach does present additional challenges, namely how to determine *at design time* how well the various possible configurations of agents and available resources will achieve our desired results.

One metaphor gaining increasing currency is that of a collection of distributed agents as an economic system [15]. Numerous projects have applied market mechanisms to particular problems in distributed resource allocation [6]. In our work, we have been applying the economic metaphor literally, conceiving of the market-price system as an architecture for multiagent systems. In this approach, agents are participants in a *computational economy*, interacting in the market to further their own interests. Behaviors

are described in standard economic terms of production, consumption, bidding, and exchange.

The market-oriented approach offers several potential advantages, which we summarize below. Following the description of our basic approach, we consider the issues of agent design (theories, architectures, languages) peculiar to this approach.

## 2 Market-Oriented Programming

### 2.1 Overview

Market price systems constitute one well-studied class of mechanisms for allocating resources among distributed decision makers. By implementing a virtual market system for computational agents, we can hope to realize some of the desirable properties of markets in the distributed computing context. For example, in some well-defined circumstances, one can demonstrate that markets produce efficient allocations with minimal communication overhead. In that sense, we can sometimes reason about the mechanism in a principled way to decide whether it is appropriate. Indeed, one of the primary motivations of this *market-oriented programming* approach [32, 33] is to exploit the analytical framework of economic theory as a design tool for multiagent systems.<sup>1</sup>

The idea of market-oriented programming is to solve a distributed resource allocation problem by formulating a computational economy and finding its competitive equilibrium. To formulate a problem as a computational economy, we must cast the activities of interest in terms of production and consumption of goods, and define a set of agents that choose strategies for production and consumption based on their own capabilities and preferences and the going market prices.

To act in accord with the theory of competitive behavior, the agents must adhere to certain rationality conditions. *Consumer agents* are endowed with an initial quantity of goods and engage in trades so as to maximize their utility. *Producer agents* are associated with a *technology*, which specifies an ability to transform some goods into other goods. The sole objective of producers is to choose an activity within their technology so as to maximize profits. From the agents' perspective, the state of the world is completely described by the going prices; that is, the prices determine the maximizing behaviors. This arrangement is extremely modular, as agents need not expressly consider the preferences or capabilities of others, and communication consists exclusively of offers to exchange goods at various prices.

Since these computational economies are instances of general-equilibrium systems, the analytical tools and results of general equilibrium theory are directly applicable [25, 28]. In particular, under certain classical conditions, a simultaneous equilibrium of supply and demand across all of the goods is guaranteed to exist, be reachable via a distributed bidding process, and be Pareto optimal (that is, there is no solution that makes some agent better off without making some other one worse off). Other theoretical properties of equilibrium can be used by designers to configure the system so that it achieves

---

<sup>1</sup> For more complete and current pointers to work in market-oriented programming, see: <http://ai.eecs.umich.edu/people/wellman/MOP.html>.

**Fig. 1.** UMDL environment

are modeled as consumers, the collections and mediators as producers. A variety of specialized mediator agents help users find and acquire various goods and services, such as retrieving documents or answering queries. For example, there is a mediator agent who can alert users when a new paper, in their field of interest, is added to a collection.

One major consideration for an information network is that many of the resources are limited, including computational resources and network capabilities, as well as user funds and time. Ideally one would like to allocate these limited resources so that a globally desirable combination of services and service levels will be provided. However, this is difficult to ensure in a distributed environment, where relevant information is dispersed and decision making is decentralized. By setting up markets in the basic resources, we have some hope of allocating these resources to the most valuable activities.

The quality of market solutions, however, depends on many factors, including the behavior of the agents. In particular, some of the desirable properties of market coordination mechanisms depend on the assumption of competitive behavior. But since the

various agents may have been built by different individuals or companies, it is generally unrealistic to assume that all behave according to such a strategy, or even to require any standard design process across agents. Here, as in most agent-based architectures, the specific construction of agents is typically below the abstraction level at which we are designing the distributed system. Therefore, the best we can do is design market structures that *promote* competitive behavior, by providing *incentives* to agents to act competitively.

### 2.3 How Valid are Competitive Assumptions?

Competitive market approaches are based on the assumption that agents act as though their behavior can have no influence on the price. That is, they take prices as given and assume that their own impact on the market is negligible. Generally speaking, this assumption is realistic in markets with many agents, with no single agent unusually large compared to the others.

In any market with a finite set of agents, there is some nonzero payoff for an agent to behave strategically, that is, consider the effect of its own behavior on prices. As the number of agents increases, the benefit decreases, and there exists some point where the cost of considering such strategic interactions exceeds the gain of doing so. Therefore, one approach to promoting competitive behavior is to increase the homogeneity of the markets, thus tending to increase the number of participants in each. This can be done by establishing an abstract fundamental unit through which a wide variety of related goods can be traded.

Where information goods are cheap, and the market interactions are dynamic, it may not require very many agents to make the cost of considering strategic interactions too expensive to be worthwhile. Thus, increasing dynamism or other sources of uncertainty will tend to promote competitiveness.

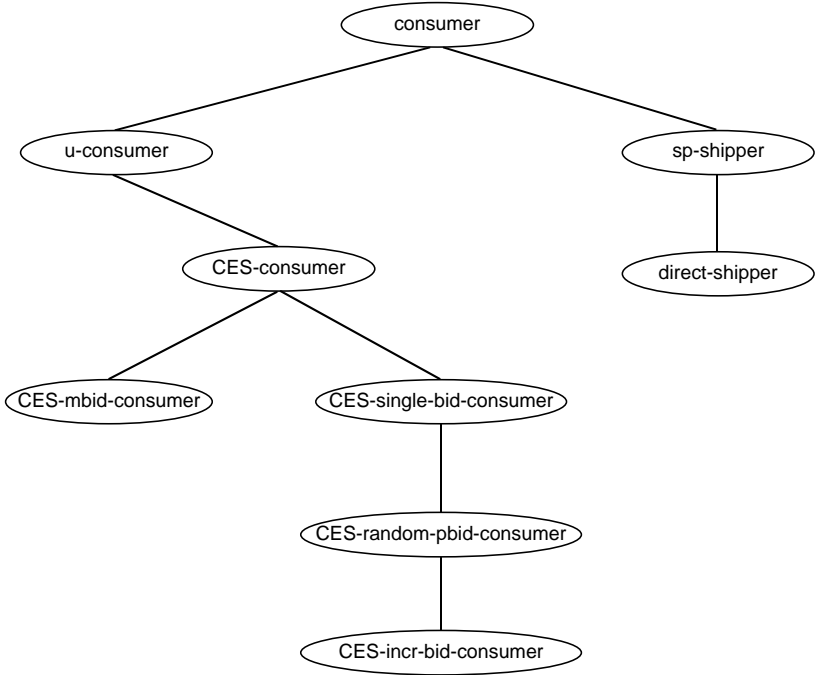
Finally, we may sometimes have sufficient control to directly enforce competitive behavior among participating agents. In the extreme case, the agents are synthesized to our standards, and hence we are assured that the competitive assumptions will be followed.

Of course, there will inevitably be cases where the competitive mechanism simply does not apply. Where other mechanisms are more appropriate they should be used. A more precise characterization of when the competitive mechanism is and is not effective is a goal of our research, as is the development of mechanisms admitting imperfect competition.

### 2.4 The WALRAS Environment

We have developed a testbed environment, called WALRAS, for designing, constructing, and executing computational economies. The main version of WALRAS runs in a single process, simulating the distributivity of a multiagent system through internal protocols. The system is written in Common Lisp and CLOS, making substantial use of class hierarchies to represent the various types of agents (consumers and producers) and bid structures (continuous, piecewise, discrete) from which a market specification can be created. Consumers and producers are further specialized according to agent-specific

properties. For example, consumer agents can be classified according to the form of their utility functions, as in the CES-consumer class (constant elasticity of substitution). Part of the current consumer taxonomy is presented in Figure 2. Note that the class hierarchy includes both generic agent types (those on the left branch), and types specific to a particular problem domain (those on the right), in this case transportation planning.



**Fig. 2.** Taxonomy of consumer types.

To set up an economy, one determines what the goods (including basic resources, service performance attributes, or whatever) of the problem are, the agents who will be doing the trading, and their bidding characteristics. For each good, WALRAS sets up an *auction* object to handle bids submitted by agents for that good. Bids in WALRAS are actually supply and demand functions which, for a given price, specify how much of that good an agent would be interested in either buying or selling. Thus each auction can find its current clearing price (where the latest aggregate supply and demand bid schedules cross). Once the auction computes its latest clearing price, it posts the price onto a “tote-board” of current prices that any of the agents can access. Consumers and producers keep a bid agenda of auctions they need to send updated bids. When they calculate a new bid

for a given auction, the demand for the associated good is based on the current prices of the other goods. Since the prices for those other goods are simultaneously undergoing price changes themselves, it is clear that the bidding process must be iterative.<sup>2</sup>

Given a few technical assumptions, the computational economy will indeed eventually converge to an equilibrium [5]. Discussion of these assumptions and computational properties can be found elsewhere. In this paper we focus on one of these assumptions, namely that the agents must be *rational*.

### 3 Rational Agents

It is inherent in the economic perspective that the participating entities be conceived as rational agents. Indeed, the rationality abstraction is one of the main features distinguishing economics from other social science disciplines. In that sense economics and artificial intelligence are natural allies, as AI is the branch of computer science that takes the rationality abstraction (viewing computational systems at the *knowledge level* [19]) seriously. Indeed, the quest to build a rational agency can be seen as a common thread throughout the field of AI [21]. In fact, several papers in this volume are based on the notion of rational agency, including [12, 29, 27].

Why should we be concerned with rational computational agents? In addressing this question, we start from the most basic view of what computation is for. Our fundamental premise is that much of what computers do and will be called on to do in the future is best viewed as decision making, and that to effectively control the behavior of these *decision machines* [34], we require a body of engineering principles that expressly relates the structure and content of a computational system to the nature and value of the decisions it produces. Bayesian decision theory can provide us with a suitable general framework (i.e., accounting for uncertainty and graded preferences) to serve as a foundation for these principles, and thus our challenge is the development of a computational version of this theory, leading to representations and reasoning strategies combining computational plausibility with decision-theoretic accountability. The name for this enterprise is *decision-theoretic planning* [13, 31].

Just as decision theory is an appropriate abstraction for single-agent behavior, economics can provide a framework for decentralization of decision making *across* rational agents. The key idea is that agent rationality makes it more convenient to design systems built of agents. Why should this be true? For the same reason that the rational-agent perspective (or as Dennett calls it, the *intentional stance* [9]) is often the simplest and most accurate way to characterize a complex computational system. Furthermore, getting the agents to believe or want particular things may be the most direct way to influence their behavior. This is a primary motivation for Shoham's *agent-oriented programming* approach [24], where the agents interact through speech-act communication. In the economic context, it corresponds to the methodology of *mechanism design*, or achieving overall objectives by *incentive engineering*.

---

<sup>2</sup> Unless, of course, there is only one good, in which case no iteration is necessary. This simpler *partial-equilibrium* version of the problem is actually the one tackled by most computational market systems described in the literature.

For example, what kind of incentives are necessary for publishers and service providers in a digital library to supply truthful information about their products? If we reward publishers according to the number of accesses (or hits) to their collections, then it is to their advantage to claim that their products cover more topics than they really do. Incentive mechanisms to prevent this might include using user feedback to rate the value of the information, and employing fee structures that impose penalties on publishers for accesses that are not followed up by the users.

Despite the centrality of decision-theoretic rationality in our view of computational economies, at present we have little to say (beyond ideas in the current literature [13, 17]) about how to make economic agents rational in the decision-theoretic sense. The reason is that there is no difference in the problem of achieving computational rationality in this context compared to any other context. That is, designing rational agents is the general AI problem, and we are working on pieces of it just like every other research group.

Rather, the contribution on which we wish to focus in the present paper is to identify some special issues that arise in the context of computational markets, and how this may affect the design of rational *market agents*. Because the internal decision problem facing such agents is completely general, we further focus on the interface to the economic system, that is, dealing with concepts such as goods, prices, and bids.

## 4 Tasks of a Market Agent

Fundamentally, a market agent is no different than any other agent. It has mental state (beliefs, preferences, intentions), and capabilities (technology, resources), and can communicate with other agents. The key distinguishing feature of the market agent is its interface with the rest of the world. Specifically, its interactions with other agents are primarily through the exchange of goods and services, and its communications are primarily devoted to arranging such exchanges.

There are three major tasks specific to the economic environment, which must be addressed in the design of market agents. Following brief descriptions here, we elaborate each in turn in subsections below. We devote special attention to the first of these, as it is the major enabling problem for deploying large-scale, open, distributed, computational markets.

The first task of the market agent is to identify the goods and services, so that it knows in what terms to interact with other agents. That is, there must be a commonly understood vocabulary for describing the resources to be exchanged or activities that agents can perform for each other. Of course, this is necessary for any type of agent interaction, but in the market context it has a special structure. Resources, activities, and the like are cast as goods for which markets exist. Once an agent knows the goods, it knows the range of possibilities for its interactions with other agents.

Second, the agent must solve its own optimization problem. That is, once it knows the goods and the possible terms by which they may be exchanged, it needs to determine its exchange policy. In doing so it optimizes some criterion—utility or profits—subject to feasibility constraints of available resources and capabilities for generating them.

Finally, the market agent must aim to implement its optimal policy through negotiating terms. In a competitive market, this entails managing its bidding activity, determining how best to arrange its requests and commitments to other agents in order to best further its interests.

#### 4.1 Identify Goods and Services

For any given computational economy, the first step is to identify what the goods and services are. Selecting the array of goods and services available strongly constrains the design space. The more standardized the goods, the simpler are the choices for each agent, but the less diverse the marketplace is. Thus, depending on design requirements, goods may either be decomposed according to various properties such as location, quality, and timeliness, or else they can be combined to hide relatively unimportant distinctions. For example, consider the goods in a simple network information services economy [18], which modeled provision of a single service (a weather service, called *Blue-Skies* [22]), with a single service level, to several sites. In this model, we chose to represent an amalgamation of network characteristics such as throughput and reliability, which were not important distinctions for this economy, as a single *Network\_Resource* good. On the other hand, location was an important characteristic here: users wanted to purchase *Blue-Skies* at their local site only. We represented this by distinguishing between the service *Blue-Skies* and *Blue-Skies@local-site*. The units for this service combined both quantity and quality elements, which could be measured, for example, in terms of kbytes of service delivered within a certain time period.

When designing small and static computational markets, it is possible to know all the goods needed ahead of time. However, in the case of large-scale dynamic markets, the set of goods as well as their important distinctions may be expected to change over time. Also, we may want to bundle two separate goods together, at any given time, to make a new good. In this environment, specification of all possible goods will require the creation of a description language. See Section 5 for more about the requirements of such a description language.

#### 4.2 Solve Optimization Problem

Given an economic environment defining the available goods and the pricing terms available, the market agent faces an optimization problem. For consumers, the problem is to maximize utility (or expected utility), subject to the *budget constraint* that it be able to afford its consumption bundle. Specifically, the budget constraint dictates that the value of its consumption bundle at the going prices not exceed the value of its *endowment*, or initial allocation of resources, at these prices.

The optimization problem facing producers is to maximize profits, subject to technological feasibility. Profits are simply revenues (value of outputs at going prices) minus costs (value of inputs at these prices). Technological feasibility simply dictates that the production of the outputs from the inputs is feasible, according to the producer's technology, or specification of its capabilities.

We make several observations about these optimization problems. First, they are both constrained problems, where the constraints are absolute restrictions, given by the

agent's capabilities or the solvency rule of the market. Second, the type of problem is definitional for the type of agent. Consumers have preferences and endowment but no technology; producers have a technology but no preferences. This purity enables us to specialize the agents and allow them to focus on a single criterion (utility or profits). In a very general setting (i.e., in the absence of production externalities), we can without loss divide any combined entity into pure consumer and producer agents.

Third, the agent is *competitive* exactly when the prices are taken as given in the optimization problem. This is simply the definition of competition in economic theory. Competitive behavior simplifies the agent's problem by allowing it to neglect the effect of its own actions on prices, and frees the agent from having to reason about the capabilities and preferences of other agents.

Fourth, with or without perfect competition, the focus on the agent's own optimization criterion (consumption or profits) is a strong form of modularity that greatly facilitates scalability and distributed design of multiagent systems.

Finally, exactly how the agent goes about solving this optimization problem is below the agent abstraction level. That is, the agent can solve algebraic equations, prove theorems, follow hand-coded rules, or whatever in service of its objective. The important thing for understanding this module as its agent is that the solution to the optimization problem, abstractly specified, be a good approximation to the agent's actual behavior. This is simply the rationality property discussed above.

### 4.3 Manage Bidding Strategy

Given the solution to its optimization problem, the competitive agents simply transmits this solution to the entities managing the markets for all of its goods of interest. However, even this simplest case is complicated by the fact that multiple agents are submitting bids for multiple goods simultaneously and asynchronously. Because the goods are highly interconnected through joint preferences and technology relations, the price of one good affects demand for another. Thus, the bidding process must be iterative, with price changes necessitating recomputing of bids, causing further price changes, and so on.

Given the continual demand for computation of optimal policies and bids, the agent faces the problem of which goods to compute bids for, and when. The choices might depend on how obsolete the agent's outstanding bids are, how relatively important are the various goods, and other factors. Ultimately, this is a kind of deliberation scheduling problem, the solution of which may depend on the characteristics of particular market environments.

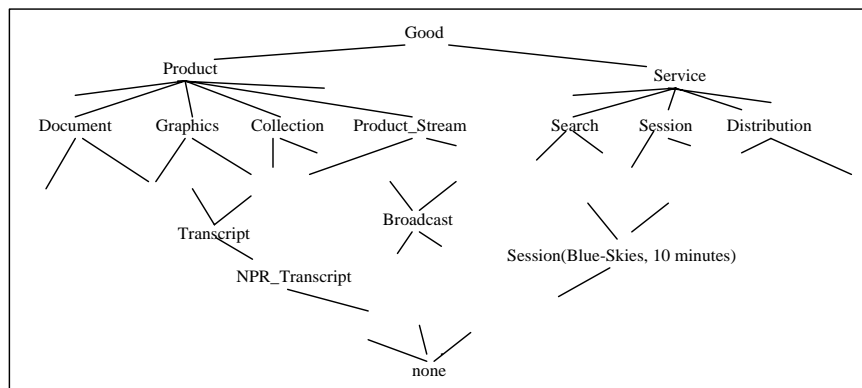
Special problems arise in highly dynamic markets, where goods may be exchanged before a global equilibrium is reached across all goods. In such cases, the agent may need to consider how close a market is to clearing in weighing the priority of alternate bid tasks.

A related problem is that of multiple outstanding commitments. Bids actually represent provisional commitments, which may be withdrawn or modified by subsequent bids. However, given uncertainty about the resolution of some of an agent's outstanding commitments, it may be unsure about the value or even the feasibility of others.

## 5 Description Language for Goods and Services

Clearly the key component of any marketplace, computational or real, is how easy it is for buyers and sellers to transact. Producers need to be able to describe their goods well enough to interest prospective buyers. Consumers need to be able to describe their wants well enough to find the appropriate goods. Thus a central representation issue is how to describe the goods and services being exchanged.

A description language requires that a reasonable taxonomy be defined for primitive good types. Language operators, such as abstraction and composition, determine the set of possible goods that can be described in terms of these primitives. This set forms a lattice, exemplified by Figure 3. There is a strong correspondence between the description language and its market implementation, an issue we examine in more detail in Section 5.1.



**Fig. 3.** Example good lattice.

To show the usefulness of a description language more concretely, consider a particular good, *NPR\_Transcript*, a National Public Radio (NPR) transcript. We can tell from the taxonomy generated from the descriptions that the *NPR\_Transcript* good inherits characteristics from the more abstract *Transcript* good. Therefore, assuming polymorphism, any code or interfaces that work on *Transcripts* will also work on *NPR\_Transcripts*, although it may have additional properties.

Abstraction can also serve as a guide for consumers. Consumers cannot be expected to know the precise name of every good they want, but they can often arrive at the good by using various kinds of abstraction as well as known relationships between types of goods. For example, they may know about NPR and *Broadcast*, from which they can get to *NPR\_Broadcast*. Since one of the possible operations which can be performed on *Broadcasts* is to make *Transcripts* from them, it should be possible to construct a description for *NPR\_Transcript* even if this particular good had not been expressly specified in advance.

Parameterization provides a particularly useful kind of abstraction over goods. A

parameterized good description does not have to be fully specified in order to describe the desired good. For example, *NPR\_Transcripts* may have fields relating to date and topic. Consumers interested only in acquiring the latest transcript can ignore any fields, such as topic, which they have no preferences over.

## 5.1 Technology Specifications

There is a natural correspondence between a service description and a technology which creates it. In fact, a service description can be considered a partial description of a producer's technology. For example, the service

*subscription of NPR\_Transcript for 6 months*

can also be thought of as a specification for a producer:

**Input:** *NPR\_Transcript@location1 for 6 months, Network\_Resources*

**Output:** *NPR\_Transcript@location2 for 6 months.*

Of course, this description only specifies the input/output types, not the quantities of input required to produce a given amount of output, i.e., not the production sets. By adding an algebraic specification capability, a full technology specification could be made.

A technology specification can serve as a template for creating producer agents. For example, a designer could use a *subscription* template, fill in some parameters, and end up with a custom agent, without having to program it. Even if the designer chooses not to use the entire default agent, by adhering to the formal interface representation, agents can be prevented from certain classes of errors such as trying to link up with inappropriate markets.

Extending this approach to allow hierarchical specifications, i.e., specifications over groups of producers, would allow the creation of sub-economies just by filling in some input/output parameters. For example, a network-transportation sub-economy was inserted into the Blue-Skies economy just by properly initializing the sub-economy parameters.

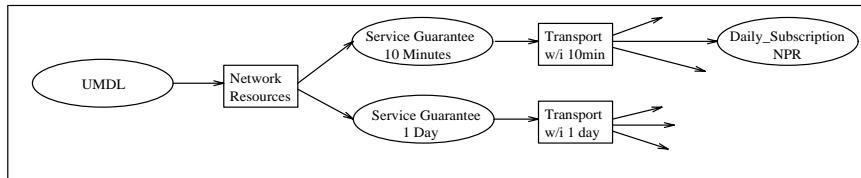
In certain restricted circumstances, producer agents could be synthesized automatically to meet dynamically changing technology needs in the environment. In the next section, we discuss how one might dynamically exploit this isomorphism between services and producers.

## 5.2 Examples

Dynamic creation of new goods can be done using description language operators, such as abstraction, composition, or coercion. The use of these operators may have a direct reflection in terms of a market implementation. Below we examine the potential usefulness of this via three simple examples. Note, these examples are meant to be illustrative in nature only. Diagrammatic conventions are that producers are represented as ovals, consumers as circles and markets as squares.

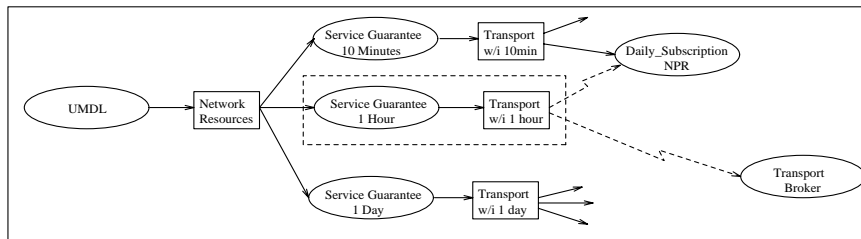
**Abstraction** One central requirement is to be able to hide unimportant distinctions between goods, while keeping the important ones. This can be achieved by use of abstraction. More specialized good descriptions are used when distinctions are important, more general ones when they are not.

Consider a NPR daily subscription service which can either buy its transportation from a market with a ten minute delivery guarantee or a one day guarantee. The one day guarantee is too long, so it is forced to buy transportation with a ten minute guarantee, even though it only needs a guarantee of one hour.



**Fig. 4.** Transport services

The two service guarantee producers in Figure 4 are both instances of a Transport Service Guarantee template. They were created by filling in a service guarantee and a broker-to-notify parameter. From that a producer and its associated auction were automatically synthesized. The *Transport Broker* agent specified by the broker-to-notify parameter notifies agents which might be interested in this new service.

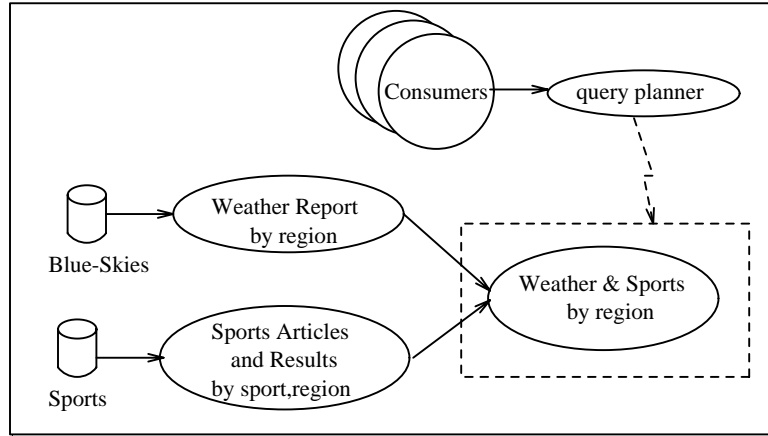


**Fig. 5.** Automatic synthesis of new transport service

By monitoring the requests of its users, the ten minute service producer can see that there would be a market for a one-hour service. It could spawn this service (and its related auction), by specifying a one hour service guarantee and the proper *Transport Broker* agent to contact in the Transport Service Guarantee template. The ten minute service producer might also directly notify some of its customers, such as the NPR daily subscription, about this new service.

**Composition** Composition operators allow two or more goods to be combined, or bundled, into one good. Suppose a query planner agent notices that it is often handling re-

quests for local weather and major sports news. It might spawn a producer which per-



**Fig. 6.** Automatic Synthesis of Bundling Producer

forms this composition operation by instantiating a bundling template with the input and output goods:

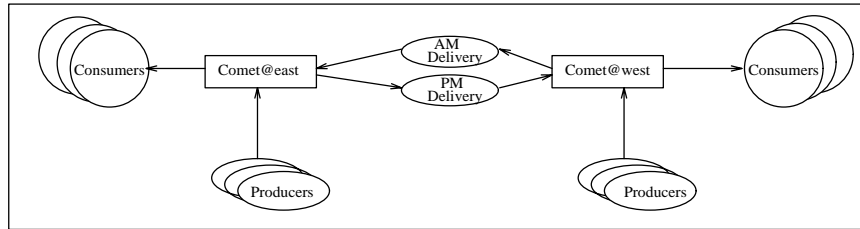
**Input:**  $Weather(\$region), Sports(\$region, \$sport)$

**Output:**  $bundle(Weather(\$region), Sports(\$region, \{basketball, football, baseball\}))$

This description allows the region variable,  $\$region$ , to remain the same across weather and sports, while at the same time setting the  $\$sport$  variable to the major sports of basketball, football and baseball.

**Coercion** In most programming languages coercion of one type to another would play a minor role. In market-oriented programming, we expect it to play a much more prominent one. Goods which are expressed in different terms, either more abstractly or composed in different ways, can be made equivalent by using a coercion operator to link the two goods. The description language might express this by  $identity(good1, good2)$ .

The coercion operator has a direct implementation as an arbitrageur agent which takes as input  $good1$ , and outputs  $good2$ . One potentially useful application of this operator will be coercion between goods that differ only in their location. Let's say there are two markets for comet pictures – one on the east coast, one on the west coast, see Figure 7. These goods are indistinguishable except for location. Assuming, simplistically, that the east coast consumers are 'closer' to the east coast market, (i.e. network transport costs are lower), one might expect that in the morning the east coast market would get swamped, being 3 hours ahead, while the west coast market was relatively unburdened, (vice versa in the late afternoon). An arbitrageur agent which could take as input the 'west coast' pictures and produce as output the 'east coast' pictures in the morning would, in effect, be distributing the load.



**Fig. 7.** Arbitrageurs as coercion operators

## 6 Conclusions

Computational economies present some common as well as some unique challenges for the agents that operate within them. The special problems of market agents arise on the interface, due to the particular mode of interaction of agents in a market. We expect these sorts of environments to become increasingly common ones for software agents, if only because “the market” is the world-standard default interface for interacting entities. As we introduce autonomous bits of software into such environments, will clearly need new ideas about agent theories, architectures, and languages to make them maximally effective. In this short essay we have identified just a few special issues that need be addressed by such theories.

## References

1. Agorics, Inc. Real-time video delivery with market-based resource allocation. Technical Report ADd004P, Agorics, Inc., 1994.
2. William P. Birmingham, Karen M. Drabenstott, Carolyn O. Frost, Amy J. Warner, and Katherine Willis. The University of Michigan Digital Library: This is not your father’s library. In *Proceedings of Digital Libraries '94*, pages 53–60, June 1994.
3. William P. Birmingham, Edmund H. Durfee, Tracy Mullen, and Michael P. Wellman. The distributed agent architecture of the University of Michigan Digital Library (extended abstract). In *AAAI Spring Symposium on Information Gathering in Heterogeneous, Distributed Environments*, Stanford, CA, March 1995.
4. Nathaniel Rockwood Bogan. Economic allocation of computation time with computation markets. Master’s thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1994.
5. John Cheng and Michael P. Wellman. The WALRAS algorithm: A convergent distributed implementation of general-equilibrium outcomes. In preparation.
6. Scott H. Clearwater, editor. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1995.
7. Scott H. Clearwater, Rick Costanza, Mike Dixon, and Brian Schroeder. Saving energy using market-based control. In Clearwater [6].
8. Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.
9. Daniel C. Dennett. Intentional systems. In John Haugeland, editor, *Mind Design*. MIT Press, 1981.

10. Jon Doyle. A reasoning economy for planning and replanning. In *Technical Papers of the ARPA Planning Initiative Workshop*, February 1994.
11. Donald F. Ferguson, Christos Nikolaou, Jakka Sairamesh, and Yechiam Yemini. Economic models for allocating resources in computer systems. In Clearwater [6].
12. P. J. Gmytrasiewicz. On reasoning about other agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).
13. Steve Hanks, Stuart Russell, and Michael P. Wellman, editors. *AAAI Spring Symposium on Decision-Theoretic Planning*. AAAI Press, March 1994.
14. Kieran Harty and David Cheriton. A market approach to operating system memory allocation. In Clearwater [6].
15. Bernardo A. Huberman and Tad Hogg. Distributed computation as an economic system. *Journal of Economic Perspectives*, 9(1):141–152, 1995.
16. James F. Kurose and Rahul Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705–717, May 1989.
17. Joerg Mueller et al., editors. *ECAI Workshop on Decision Theory for DAI Applications*, 1994. Unpublished working notes.
18. Tracy Mullen and Michael P. Wellman. A simple computational economy for network information services. In *First International Conference on Multi-Agent Systems*, pages 283–289, San Francisco, June 1995.
19. Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.
20. Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. MIT Press, 1994.
21. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.
22. Perry J. Samson, Ken Hay, and Jeffrey Ferguson. Blue-skies: Curriculum development for k-12 education. In *American Meteorological Society Conference on Interactive Information and Processing Systems*, Nashville, TN, January 1994.
23. Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of the National Conference on Artificial Intelligence*, pages 256–262, Washington, DC, 1993. AAAI.
24. Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.
25. John B. Shoven and John Whalley. *Applying General Equilibrium*. Cambridge University Press, 1992.
26. Michael Stonebraker, Robert Devine, Marcel Kornacker, Witold Litwin, Avi Pfeffer, Adam Sah, and Carl Staelin. An economic paradigm for query processing and data migration in Mariposa. In *Third International Conf. on Parallel and Distributed Information Systems*, Las Vegas, NV, 1994.
27. B. van Linder, W. van der Hoek, and J. J. Ch. Meyer. How to motivate your agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).
28. Hal R. Varian. *Microeconomic Analysis*. W. W. Norton & Company, New York, second edition, 1984.
29. J. M. Vidal and E. H. Durfee. Recursive agent modeling using limited rationality. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents Volume II — Proceedings of the 1995 Workshop on Agent Theories, Architectures, and Languages (ATAL-95)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996. (In this volume).

30. Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and Scott Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18:103–117, 1992.
31. Michael P. Wellman. Challenges of decision-theoretic planning. In *AAAI Spring Symposium on Foundations of Automatic Planning*, pages 156–160, 1993.
32. Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–22, 1993.
33. Michael P. Wellman. Market-oriented programming: Some early lessons. In Clearwater [6].
34. Michael P. Wellman. Rationality in decision machines. In *AAAI Fall Symposium on Rational Agency*, November 1995.
35. Michael P. Wellman. A computational market model for distributed configuration design. *AI EDAM*, to appear.