Designing Conventions for Automated Negotiation

Jeffrey S. Rosenschein and Gilad Zlotkin

■ As distributed systems of computers play an increasingly important role in society, it will be necessary to consider ways in which these machines can be made to interact effectively. We are concerned with heterogeneous, distributed systems made up of machines that have been programmed by different entities to pursue different goals. Adjusting the rules of public behavior (the rules of the game) by which the programs must interact can influence the private strategies that designers set up in their machines. These rules can shape the design choices of the machines' programmers and, thus, the run-time behavior of their creations. Certain kinds of desirable social behavior can thus be caused to emerge through the careful design of interaction rules. Formal tools and analysis can help in the appropriate design of these rules.

We consider how concepts from fields such as decision theory and game theory can provide standards to be used in the design of appropriate negotiation and interaction environments. This design is highly sensitive to the domain in which the interaction is taking place.

This article is adapted from an invited lecture given by Jeffrey Rosenschein at the Thirteenth International Joint Conference on Artificial Intelligence in Chambery, France, on 2 September 1993.

We've all been hearing a lot about convergence between telephone, television, and computer technology. The basic idea is that the networks that constitute our telephone infrastructure, our television (particularly cable) infrastructure, and our computer infrastructure will be coalescing into one harmonious whole. Then the user, sitting in his/her home or office, has some kind of information appliance that can handle the wealth of resources that are available.

Now, AI has a role to play in how the computer works in this brave new world of information consolidation. Several groups of AI researchers are already actively involved in trying to design automated agents that will help the user filter or retrieve information from the network. To mention just one example, Oren Etzioni's group at the University of Washington at Seattle is building what he calls *softbots,* software robots, to handle the interface between humans and network resources. There are also a lot of good, meaty, classic AI problems that need to be solved in this context, such as knowledge representation and planning problems.

What we discuss here has a strong connection to those efforts to build software agents. What we have been interested in over the last few years has been to look at the ways in which these automated software agents will be dealing with one another. In other words, it's all well and good to have agents residing in your home computer that help you deal with all the information, but this agent of yours is going to have to deal with the agents of other people, either private agents if you're trying to do a job such as setting up a meeting with a group of people or corporate agents if you're trying to access information from a company database. These software agents are on their way, and they're going to be getting a lot of things accomplished by interacting with each other. The question is, How will these agents be cooperating with each other, competing with each other, and negotiating with each other?

Machines Controlling and Sharing Resources

There are actually a lot of different environments in which machines are making more

and more decisions that affect our daily lives, and they're making these decisions not in isolation but in concert with other machines. Computers that control electric grid networks and need to balance their power requirements can share electricity with other computers controlling other networks. If there's a drop or rise in power consumption, one computer can sell or buy excess power from other utilities to which it's connected.

Another example is routing among telecommunication networks. Information, or packets, can pass over a network controlled by one company onto another network controlled by another company, or it can pass through one country on through another. Computers that control a telecommunications network might find it beneficial to enter into agreements with other computers that control other networks about routing packets more efficiently from source to destination. The point is that it might make sense for both machines to be able to exploit the resources controlled by the other so that they can both get their jobs done more effectively.

… the agents that we are interested in looking at are heterogeneous, self-motivated agents.

We're also seeing the emergence of tools such as *personal digital assistants,* small, handheld computers (for example, the Newton). These personal assistants are going to assume the roles of a number of machines or tools involved with managing our daily lives, such as notebooks, communicators, fax machines, telephones, and automated schedulers. We're going to have some kind of agent software on the personal assistant, and it's ultimately going to get part of its work done by interacting with other agents on other personal assistants.

Another example is the proliferation of shared databases, where there's information spread all over the world. They have sprung up with a vengeance in the last decade. You've already got agents, such as Etzioni's softbots, that are going out there to gather information—for example, a person's Internet address—from a shared database. Finally, even something such as traffic control, coordination of vehicular traffic, or air traffic control illustrates situations where software agents will be making decisions based on communication and agreements with other agents.

Each of these situations is an instance where computers are controlling some resource and might be able to help themselves by strategically sharing this resource with other computers. With personal digital assistants, the resource might be a person's time, whereas with a telecommunications network, the resource might be communication lines, switching nodes, or short- and long-term storage. In each situation, the computers that control these resources can do their own job better by reaching agreements with other computers.

Heterogeneous, Self-Motivated Agents

Now, the agents that we are interested in looking at are heterogeneous, self-motivated agents. The systems are not assumed to be centrally designed. For example, if you have a personal digital assistant, you might have one that was built by IBM, but the next person over might have one that was built by Apple. They don't necessarily have a notion of global utility. Each personal digital assistant or each agent operating from your machine is interested in what your idea of utility is and in how to further your notion of goodness. They're dynamic; for example, agents might enter and leave the system in an unpredictable way. The system as a whole is flexible; new personal assistants are coming in, even new types of personal assistants are being built and coming in to the system and have to interact with other agents.

In particular, the personal assistants do not act benevolently unless it's in their interest to do so. They do not necessarily share information, they do not necessarily do things that other agents ask them to do unless they have a good reason for doing so. Thus, imagine lots of agents, each one residing in your personal computer or on your personal digital assistant, trying to carry out tasks, and interacting with other agents.

The Aim of the Research

The aim of the research that we've been doing can really be thought of as a kind of social engineering for communities of machines. With communities of people, social engineering means setting up laws or setting up an environment that causes people to act in a certain beneficial way or causes people to act in certain ways that we decided ahead of time are desirable. At the least, it constrains their behavior.

Similarly, we're interested in the creation of interaction environments that foster a certain kind of social behavior among machines. We want to develop conventions, rules of encounter, for these software agents that will cause them to act in certain ways. Another way to look at this research is that we're trying to exploit formal tools, in this case, game the-

ory tools, for high-level protocol design. We're looking at protocols for interactions among agents, and we would like to design protocols that cause, for example, these personal digital assistant agents to act in certain ways.

Broad Working Assumption

These agents are obviously still pursuing their own utility, they're still pursuing their own goals, but they're going to be constrained somehow by the environment that we design. However, who's "we"? "We" means the designers of the network or the designers of the system. Let's say, as a working assumption, that designers from IBM, Apple, Toshiba, and Sony all come together and say, "OK, we've got this domain. The domain is personal digital assistants doing scheduling, time scheduling. Given this domain, we would like to set up the rules for the way in which schedules will be set. We want to set up the rules, a high-level protocol, that determines the kind of deals, let's say, that agents can make among themselves."

These designers then come together and try to agree on standards for how their automated agents interact. An important part of this process is that it takes place in a given domain. They might decide on different protocols for different domains, but given the domain, like scheduling among personal digital assistants, the designers are going to decide on standards for how the agents reach agreements.

Now, what are the designers actually doing in this meeting? It is a standards committee meeting: They sit around a big table and discuss the trade-offs of different decisions. One of them says, "You know, if we have this kind of protocol, the agents will be able to come to agreement quickly. The agreements probably won't be optimal, but we'll be able to get to them fast." Somebody else at the table says, "Yes, but it's important to us at IBM that these protocols not allow agents to manipulate one another. You know, we don't want any manipulative, exploitative agents getting into the system and taking advantage." Yet another designer says, "Well, that's not so important to me. The most important thing is that the average utility among all the agents be as high as possible."

We are not trying to impose a group of decisions that these company representatives might make. Instead, what we're trying to do is come forward, elucidate a variety of protocol decisions, and show how particular protocols have certain desirable attributes. Once we show that protocol A has a certain attribute, the designers of these personal digital assistants might decide to choose it, and they might not; they might choose something else. However, the idea of the research is to come forward and elucidate and elaborate on the possibilities. Protocol A has this set of attributes, and protocol B has this slightly different set of attributes. Which do you want to choose when you design your agents? It's up to you. Part of the research is to make it clear to those designers what the options are.

Attributes of Standards

What are we looking at when we try to design a standard? Well, we might look at the efficiency of the system, such as Pareto optimality; that is, the agreement that is reached by the agents can be made no better for any one of the them without making it worse for one of the other agents. Thus, we have what might be considered a broadly acceptable doctrine of general goodness.

Stability: An agent has no incentive to deviate from a particular strategy. This idea is related to the game theory notion of equilibrium.

Simplicity: This quality is important for computer science but much less important for game theory. We certainly want our protocols to have low computation and communication costs. This attribute is one that we might propose, and those designers might say, "This attribute is absolutely important; it's important that our small processor not have a heavy load in doing negotiation."

We'd like the protocol to be distributed, typically because if we're going to have a lot of distributed agents, it would be nice not to have a central decision maker. It would be nice to have it be symmetric, so that no agent plays a special role. When the agents come together, they don't have to decide who's going to be the master and who's going to be the slave or who's going to do what.

The idea is to design protocols for specific classes of domains that satisfy some or all of these attributes. As we present protocols, or design decisions, it might be the case that it's simple but not distributed or that it's stable but not efficient or that it's efficient but not stable. These are classic trade-offs.

Distributed AI

How does this work relate to other research being done in the field? It fits into the broad

… what we're trying to do is come forward, elucidate a variety of protocol decisions, and show how particular protocols have certain desirable attributes

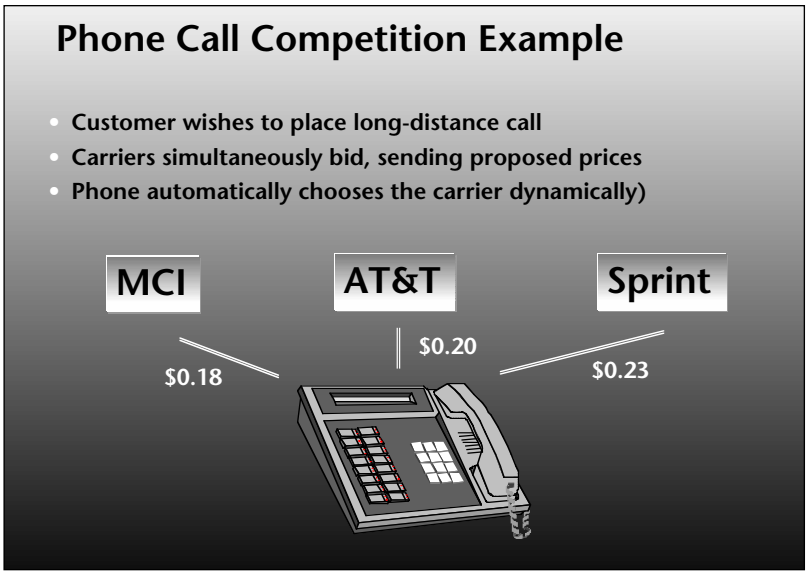

Figure 1. A System for Placing a Call.

area of distributed AI, which can be broken down into two related areas. These two areas constitute distinctions between research agendas; they are not really appropriate as descriptions of running systems.

One stream of research is *distributed problem solving,* which constitutes the original emphasis of distributed AI, namely, the study of distributed but centrally designed AI systems. How do you build a distributed system that is made up of many agents that have some global problem to solve? You're going to design the system so that the agents solve the problem in a good way, in a distributed way, in an efficient way. In distributed problem solving, there is assumed to be a single body that is able, at design time, to directly influence the preferences of all agents in the system.

This area contrasts with another stream of research within distributed AI called *multiagent systems*. In multiagent systems, you again have multiple agents in a distributed system, but you do not assume that there is a single designer who stands behind all of them, or put another way, you do not assume that the individual agents have a group sense of utility. Each of the agents in the system can be working at different goals, even conflicting goals. You have to deal with a system made up of multiple agents where competition or cooperation is possible between the agents. In multiagent systems, no single body is assumed that, at design time, can directly influence the preferences of all agents in the system. The agents' preferences arise from distinct designers.

In particular, if a distributed problem-solving researcher can show that acting in a particular way is good for the system as a whole, he/she can impose this behavior on all the agents in the system at design time. For the multiagent system researcher, such an alternative is unavailable. At best, he/she might be able to design aspects of the environment that motivate all the (selfish) agents to act in a certain way. This need for indirect incentives is one element that distinguishes multiagent system research from distributed problem-solving research.

The work we describe here is multiagent system research.

The Telephone Call Competition Example

We mentioned how we're trying to design protocols for agent interactions. To illustrate this point, we use the example of a hypothetical environment to show how different protocols motivate agents to act in different ways and how these different protocols end up having different global properties.

In the United States, there are several long-distance telephone companies, and each telephone customer sends in a postcard asking to hook up with one or another of them. The selected company becomes the customer's default carrier; you have to dial extra numbers to place a long-distance call with some other company. Imagine another kind of system, one that might operate within the current technology but has certain benefits over the way things are done now. What if a customer lifts a handset and dials a long-distance call, and a microprocessor within the telephone automatically collects bids from the various carriers? Each company's computer automatically and simultaneously declares the price per minute for which it's willing to carry the call. In figure 1, we see the MCI computer relaying 18 cents, but the AT&T computer bids 20 cents, and so on. This process can take place in a split second, without significantly delaying the call.

Best Bid Wins

Our telephone's microprocessor is now collecting the bids. Assume that the protocol involves our telephone choosing the company with the lowest bid, which is completely reasonable, and placing the telephone call with the company. The winning carrier receives a price for each minute equal to the amount that it bid. The prices set by the companies for telephone calls at different times of

the day no longer have to be fixed ahead of time, nor does the pricing system have to be simple enough to be remembered by consumers. It can be completely dynamic and sensitive to the real costs or economies that exist at any given moment. The system appears to be much more open to competition too: A new long-distance carrier doesn't have to win over consumers with a costly advertising campaign; it just needs to set up its computers to enter into the bidding game. In fact, the companies now have greater motivation to sink their budgets into researching ways to lower their costs rather than into advertising to grab consumers. The process sounds good, but this bidding mechanism has a serious flaw (figure 2).

Strategic Behavior Results

We have made the long-distance carrier selection distributed and symmetric, but the protocol, the rules by which the agents play to win the telephone call, does not encourage stable, simple, or efficient behavior on the part of the telephone company computers. Let's pretend that our consumer wants to make a long-distance call and consider the MCI computer's reasoning. Although it could carry the call for 18 cents and get an acceptable profit, it might rationally try to increase the profit by bidding higher. It might think that the next-highest bidder won't go below 22 cents and, consequently, make its own bid 21 cents. Now, this strategy is risky, but the carriers have a great incentive for investing effort in strategic behavior. Instead of putting their money into polished ad campaigns, they'll pay programmers to develop sophisticated models of their opponents' bidding strategies; that is, how much can carrier X really afford to carry a call for right now? They'll put energy into trying to discover relevant information about their opponents, information that might affect the bid the opponent puts in, for example, which switching stations are down and what the other company's profit and loss are for this quarter. Ultimately, this sort of effort drains resources that might be better spent elsewhere. Equally important, the actual bidding procedure can result in an inefficient outcome. In this example, MCI might lose the bid when it could have served as the lowest-cost alternative.

Can we do better by changing the protocol of bidding? The answer is yes.

Best Bid Wins, Gets Second Price

Let's say that in our new protocol, all the company computers put their bids in, and our telephone's computer again automatically chooses the lowest bidder as the winner. However, this time, the carrier that wins gets paid a price for each minute equal to the second lowest bid. This bidding system, called *Vickrey's mechanism*, is attractive because it provides no incentive for a company to underbid or overbid. A company has the incentive only to provide the true minimum acceptable price. A company won't bid lower than its minimum acceptable price because it fears that some other company might bid in the gap that it has opened up (in fact, if no one else bids in the gap, the first company would have won anyway without bidding low). If MCI bid 1 cent and won, somebody

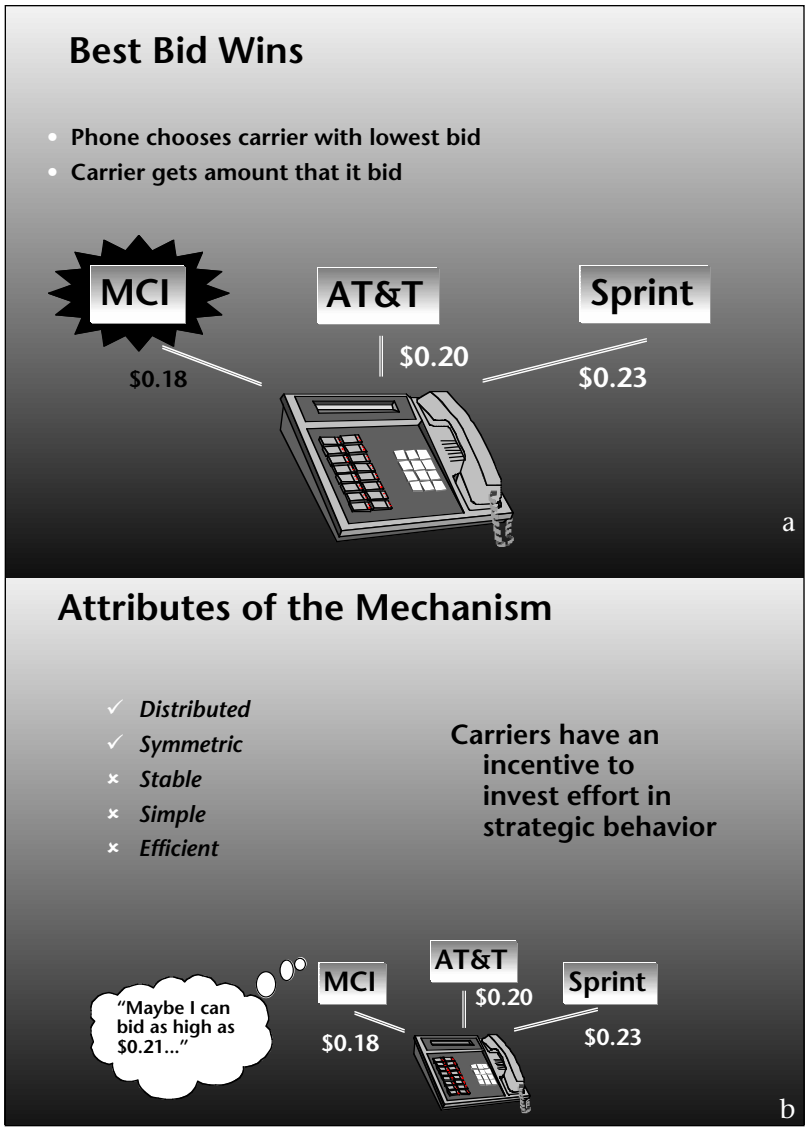

Figure 2. Rules of the Game and Resulting Attributes.

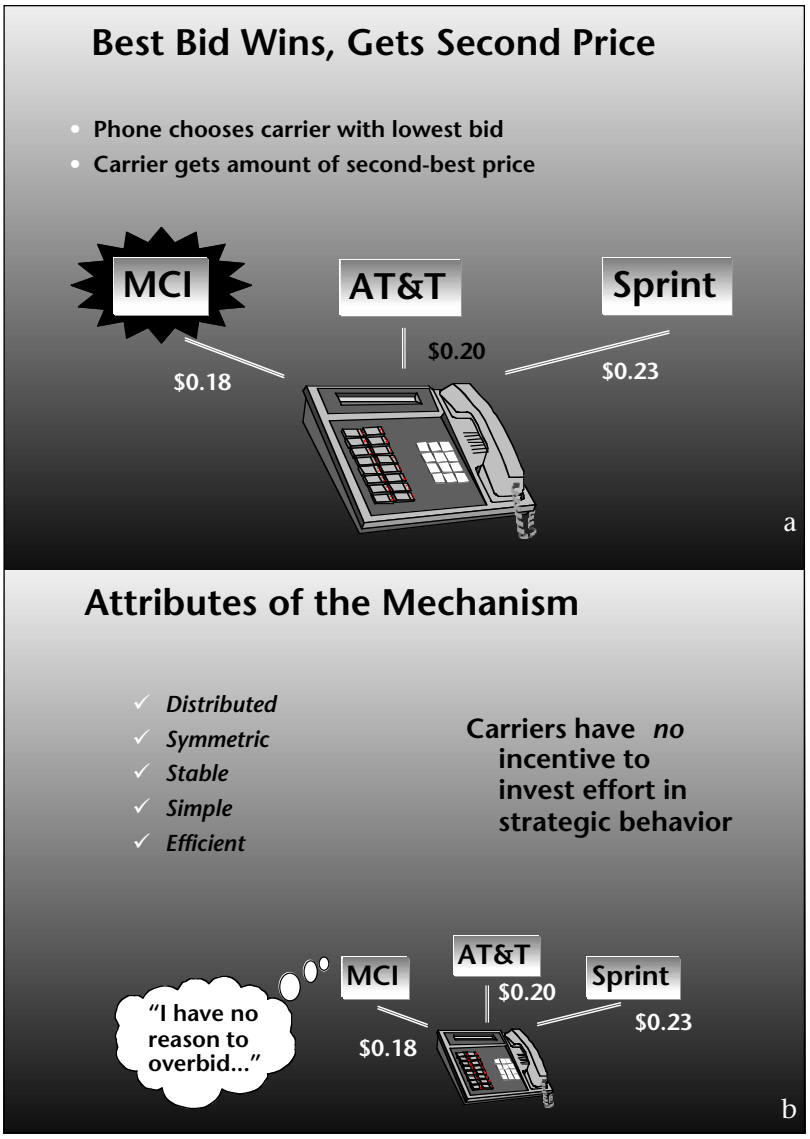

Figure 3. Different Protocol, Different Attributes.

else might bid 10 cents; then MCI would be forced to carry the call for less than its minimal acceptable 18 cents for each minute. However, no company has an incentive to overbid either. What possible benefit could MCI get from declaring a price higher than 18 cents? If it says, for example, 20 cents, it might lose a bid it would otherwise have won, and in any case, its own bid will never affect how much money it gets! By separating the issues of who wins the bid and how much the winner gets, we have fundamentally altered the way in which computers should play the game (figure 3).

Attributes of the Mechanism

Now, the carriers at our long-distance companies have no incentive to invest effort in strategic behavior. They can put all their money into lowering the costs of long-distance calls so that they'll win more bids and get more business. We've got a distributed, symmetric, stable, simple, and efficient mechanism for these self-motivated machines to use. Of course, we've bought these wonderful attributes at a cost; the consumer has to pay a small premium on each call to make things work, in this example, paying 20 cents rather than 18 cents for each minute. With many carriers, this effect will be minimized. In any case, the point of this example is not its details. We illustrated how we can design the rules of the game for multiple-agent interactions and reach a situation where rational agents are motivated to play in certain ways.

The telephone call domain is actually incredibly simple. We are interested in more complicated real-world situations, with computers controlling and sharing resources.

Domain Theory

It is important in which domain our independently motivated agents are working because these domain attributes are going to affect the properties of our protocols. A technique that works in one domain class, motivating agents to act in a certain way, won't necessarily work in another type of domain.

We've found it useful to categorize classes of domains into a three-level hierarchy, where each level is increasingly more general. This categorization is not exhaustive; there are other, more general categorizations of domains. However, the three-level hierarchy covers many of the real-world domains in which we are interested.

The lowest-level, the simplest kind of domain that we've looked at is the *task-oriented domain*. A task-oriented domain exists when agents have nonconflicting jobs to do, and these jobs or tasks can be redistributed among the agents. Thus, the agents receive some list of jobs that they have to accomplish, and the object of negotiation in this kind of environment is to redistribute tasks among the agents to everyone's mutual benefit if possible. Most of the talk will be on this first area.

The next-higher level we call the *state-oriented domain*. State-oriented domains are a superset of task-oriented domains. State-oriented domains have goals that specify acceptable final states in the classic AI way. Impor-

tantly, in contrast to task-oriented domains, actions in state-oriented domains can have side effects, where an agent doing one action might hinder or help another agent. The object of negotiation is to develop joint plans and schedules for the agents. The agents want to figure out when each agent should do each action so that they stay out of each other's way but also help one other if appropriate.

Finally, we come to the *worth-oriented domain*, which is a superset of the state-oriented domain. Worth-oriented domains assume, like state-oriented domains, that goals specify final states, but this fact is encoded in a function that rates the acceptability of states. Every state in the world is better or worse, but it's not the binary notion of goal that we have in state-oriented domains. In a worth-oriented domain, we have a decision-theoretic kind of formulation, with the agent striving for better states. Again, the object of negotiation is a joint plan, schedules, and goal relaxation. In other words, agents might not be able to get to the state that is their ultimate objective, but they might be willing to arrive at a state that is a little bit worse. Because the agents have a function that rates the acceptability of states, they are able to evaluate gradations among goal states, which they can't do in state-oriented domains.

Examples of Task-Oriented Domains

In this subsection, we discuss various examples of task-, state-, and worth-oriented domains. To illustrate task-oriented domains, we present the postmen, database, and fax domains. To illustrate state-oriented domains, we discuss the slotted blocks world. Finally, to illustrate worth-oriented domains, we discuss the multiagent tile world.

Postmen Domain A classic task-oriented domain that we've looked at quite a bit is the *postmen domain* (figure 4). In this case, two agents arrive at the post office early in the morning; they receive sacks of letters that they then have to take and deliver around the city, which is represented by a graph. At each node, there is a little mailbox. Let's say agent 1 has to go to c, f, and e and then return to the post office, but the other agent might have to go to c, b, and d and then return to the post office. This example illustrates a task-oriented domain. The cost of carrying out a delivery is only in the travel distance. There are no side effects to their actions, there is no limit to the number of letters they can carry, and there is no limit to how many letters they can put in a mailbox.

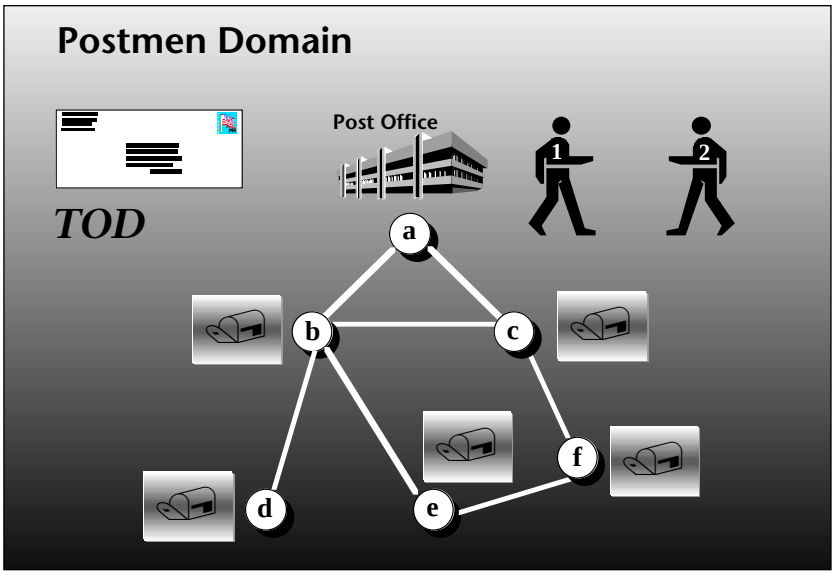

Figure 4. Example of a Task-Oriented Domain.

They have these tasks to do, and there is no possibility of getting in each other's way.

The agents cooperate before they start on their journey. For example, they look at the letters and say, "You know, it doesn't really make sense. You're going past c anyway, why don't you take my letter? It is no extra cost to you; the cost is only in the travel distance, and I'll have a shorter trip." What they try to do is evaluate these deals. There are different possible divisions of the tasks, some are better for one, some are better for the other. We would like the agents to come to some agreement about how they are going to divide the tasks.

Database Domain Let's look at another domain, also a task-oriented domain, called the *database domain*. There's a common database residing on the Internet, let's say. Two agents are sent out to get information. One is supposed to retrieve the names of all the female employees making over $50,000 a year and return, and the other one is supposed to get the names of all the female employees with more than three children and bring them back. In this domain, each subquery to the database costs money. These two agents approach the database and look at each other and say, "You know, one of our subqueries is the same. We could structure our requests for information so that only one of us asked for all female employees, and then subsequently, we would each do another operation on the subset of names. We don't both have to ask for the names of all the

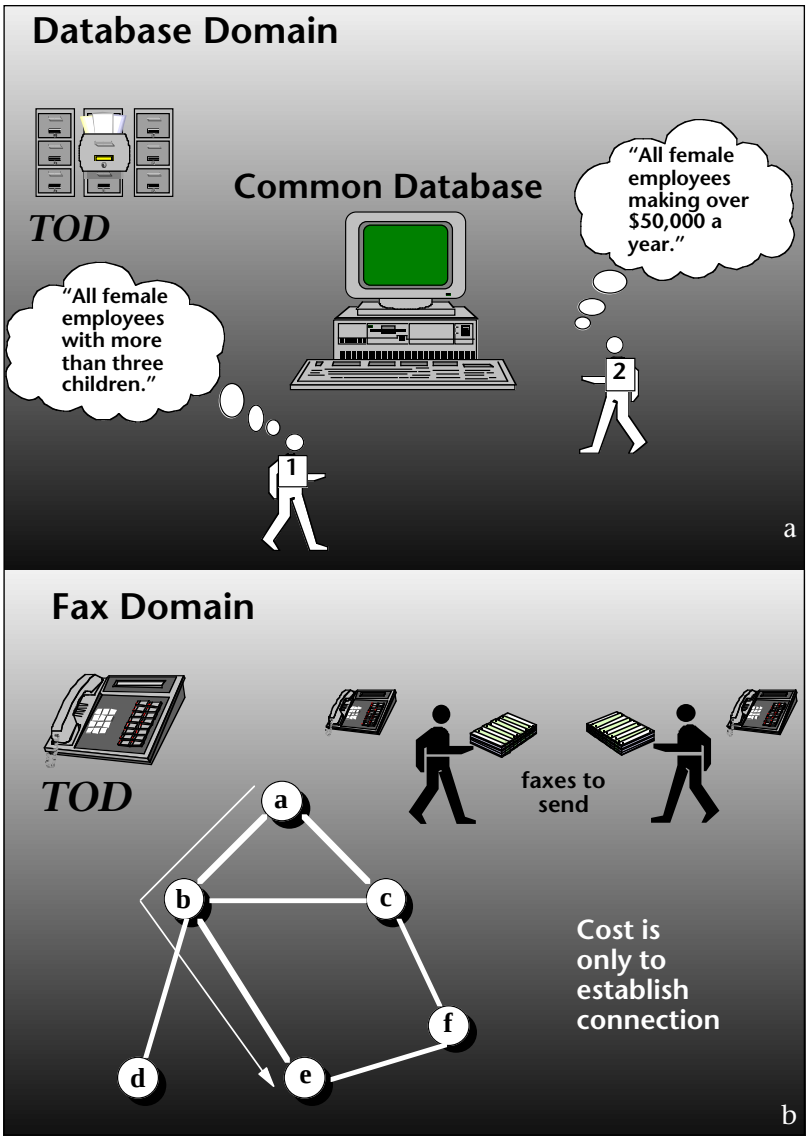

Figure 5. More Examples of Task-Oriented Domains.

female employees." There are no side effects, no possibility for getting in each other's way, just cooperation (figure 5).

Fax Domain One final example of a task-oriented domain, one similar to the postmen domain, is the *fax domain.* In the fax domain, two agents arrive in the morning and are given lists of faxes they have to send all over the world. The agents fortunately only have to pay to connect to the other fax machine; once they connect, they are allowed to down load as many faxes as they want.

The two agents might find that they both have faxes to send to London and say, "It doesn't make sense for both of us to pay the charge of connecting to London. Let's divide the faxes: You take my London faxes, and I'll take your Rome faxes."

Slotted Blocks World—State-Oriented Domain

A *blocks world* is an example of a state-oriented domain. An agent comes and wants the blocks in a certain configuration. The other agent comes and wants the blocks in a certain configuration. These goals can be identical, or they can be in conflict with one another. One agent might want the orange block on top of the blue block, and the other agent might want the blue block on top of the orange block. There's a possibility for real conflict (figure 6).

There are other possibilities, too, such as accidental cooperative action, where one agent inadvertently does something that's good for the other without the other having to ask for it. This type of action can never be true in our task-oriented domains, where there has to be some communication, passing of tasks back and forth, for cooperative action to take place. Here, side effects really affect our analysis of the domains.

The Multiagent Tile World—Worth-Oriented Domain

A *multiagent* version of the *tile world,* originally introduced by Martha Pollack, is an example of a worth-oriented domain. We have agents operating on a grid, and there are tiles that need to be pushed into holes. The holes have value to one or both of the agents, there are obstacles, and agents move around the grid and push tiles into holes. It's true that each agent has a most desirable state, where all the tiles are in its holes, but other states are also good, although less good than the most desirable state. Thus, each agent is able to rank different states, and agreements more easily reflect the possibility of compromise.

Task-Oriented Domain

Let's go back now and look at the task-oriented domains. A task-oriented domain consists of a tuple, $<T, A, c>$, where T is the set of tasks, all the possible actions in the domain; A is the list of agents; and c is some kind of monotonic cost function from any set of tasks to a real number. An *encounter* within a task-oriented domain is a list, $T_1, \ldots T_n$, of finite sets of tasks from the task set T, such that each agent needs to achieve all the tasks in its set. You might as well also call the task set its goal. Thus, we have an encounter, a group of agents coming together, each with a list of tasks.

Remember, we're doing an analysis for the sake of all those designers from IBM, Toshiba, Sony, and so on, that are going to be sitting around the table deciding how to design their personal digital assistants.

Building Blocks

In doing this analysis, we have three things we would like to look at. The first element is a precise specification of the domain, a definition of what a goal is and what agent operations are. We just performed this specification in a broad sense for a task-oriented domain.

The second element is the design of a negotiation protocol for the domain. A negotiation protocol involves a definition of what a deal is among the agents, a definition of what utility is among the agents, and a definition of the so-called *conflict deal*. The conflict deal is the deal, the default deal, that the agents get if they fail to reach an agreement. You can think of the negotiation protocol as something like the rules of the game. In chess, for example, the negotiation protocol would be analogous to a description of what all the possible moves are.

The third element is the negotiation strategy, or how an agent should act given the set of rules. Think again about a chess game. Think about separating the rules that describe the game from the technique that an agent uses in response to the environment, in response to the rules. First, we would like to define the rules of the negotiation, and then, for purposes of illuminating the situation for our designers, we would like to discuss negotiation strategies that they might choose to put into their agents.

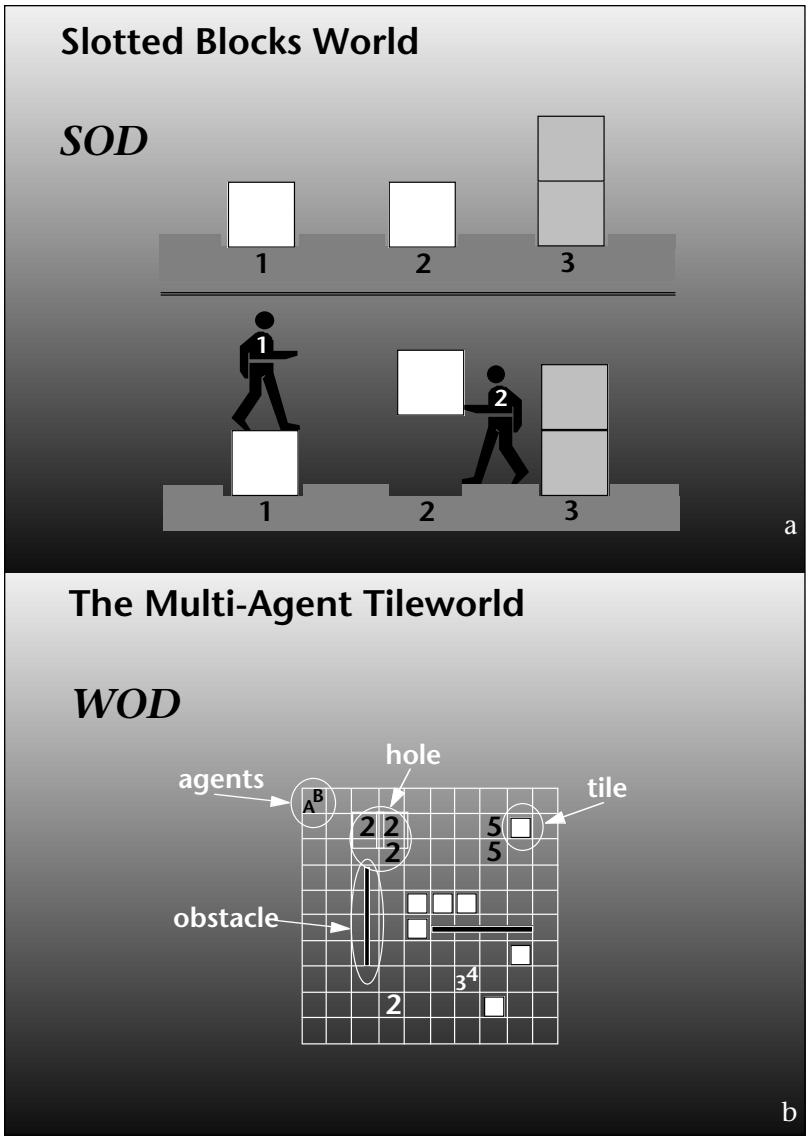

Figure 6. State-Oriented Domain and Worth-Oriented Domain Examples.

Deal and Utility in Two-Agent Task-Oriented Domain

First, we would like to have a definition of the deal, the utility, and the conflict deal. Here we have it for a two-agent task-oriented domain: A deal δ is a pair (D_1, D_2) such that $D_1 \cup D_2 = T_1 \cup T_2$. The conflict deal is defined as $\Theta \equiv (T_1, T_2)$. $\text{Utility}_i(\delta) = \text{Cost}(T_i) - \text{Cost}(D_i)$.

A deal in a two-agent task-oriented domain is a pair, D_1, D_2, such that their union is equal to the union of the original task sets. Think about those postmen with their letters. They come together with T_1 and T_2, their original sacks of letters. A deal is a new distribution, such that all the letters are taken care of.

The conflict deal in this case is simply the original sets of letters. If you don't reach agreement, you deliver your original sack of letters.

The utility of a deal for an agent is defined as the cost of its original work minus the cost of its new work given the deal. The difference is how much it has gained from the deal. It used to have to walk five miles; now it only has to walk three miles. The utility is two.

Negotiation Protocols

As far as the protocol that the agents are going to use, there are lots of good choices. For the purposes of this discussion, we're going to assume that the agents are using some kind of product-maximizing negotiation protocol, such as in Nash bargaining theory. For our purposes, it really doesn't matter which one they use as long as it's symmetric; that is, it maximizes the product of the utili-

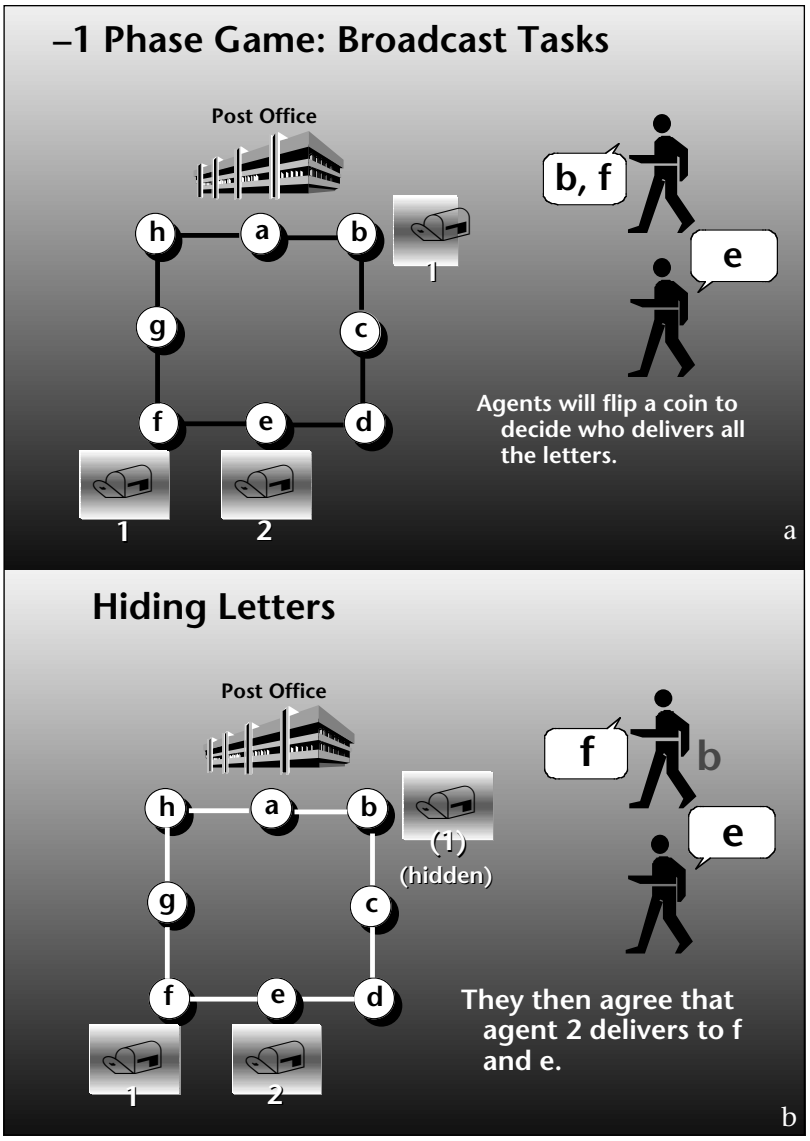

Figure 7. Dealing with (and Exploiting) Incomplete Information.

ties. There are all sorts of examples of different protocols, different rules, that will bring the agents to an agreement that maximizes the product of their utilities. You can even have a one-step protocol; if they know everything, they can compute the agreement point, which will be the point they jump to. Instead, you can have some kind of a *monotonic concession protocol,* where each agent first starts with a deal that's best for it and then iteratively makes compromises to the other agent. There are a lot of different product-maximizing protocols.

Now we return to the last item on our building blocks list. Given that we have a task-oriented–domain specification, a definition of the deal, the utility, the conflict deal, and an overall protocol, what negotiation strategy should the agents use? Given the previous discussion, you could say, "Well, strategy is not really important in this situation because once the designers have decided on the protocol, and they know exactly what the tasks are for the other agent and what their own tasks are, there's a well-defined agreement point. This well-defined point tells them when they have maximized the product of their utilities. They can just move to this point in one way or another. It doesn't really matter how," which is true when the agents have complete information.

Negotiation with Incomplete Information

What about the case where the agents don't have complete information? What about that situation where the postmen show up in the morning, and the sacks of letters are opaque, and they have to decide how to negotiate. Let's assume that the first agent has to carry letters to b and f, and the second agent has to go to e. Each agent knows its own letters but doesn't know the other's letters, so they can't compute the agreement for the two of them. Instead, they need some other kind of mechanism that will allow them to come to an agreement. One simple, straightforward technique is to set up a 21 phase game, a sort of pregame exchange of information (figure 7).

The two agents broadcast their tasks and then continue as before, computing where the agreement is going to be. Agent 2 announces that it has to go to e, agent 1 says "I have to go to b and f," and they decide who's going to do what. Now, just to carry out its own tasks, each one would have to go a distance of eight units. Agent 1 would certainly go all the way around, and agent 2 might go half-way around and back, but it's equivalent to going all the way around. In this particular case, because of the structure of the problem, the agents eventually agree to flip a coin, and one of them travels all the way around while the other one stays at the post office. Assuming it's a fair coin toss, they have divided up the work equally.

Hiding Letters

However, our intrepid agent has been built by a smart group of designers, and it makes the following claim: Agent 2 honestly says, "I have to go to e"; agent 1 says, "I have to go to f," and it hides its letter to b (see figure 7b). Now, the negotiation situation has changed

because agent 1 is purporting to say here that it only has to travel six units, and it should be required to do less of the final work. In fact, in this situation, the only pure deal that the agents can agree to is that agent 2 takes the letters to f and e, and agent 1 supposedly does nothing. This deal is agreed on because it would not be rational for agent 1 to agree to carry letters all the way around the loop. It would then be doing eight units of work, more than the six units of work it would supposedly be doing by itself. It can't be expected to agree to a deal that makes it do extra work. However, agent 2 doesn't benefit from this deal because it still travels eight units, but it isn't harmed either. Thus, the deal where agent 2 does all the work is the only rational, Pareto-optimal deal. In the meantime, agent 1 runs off and delivers its hidden letter to b at a cost of two units. Agent 1 has really made off well with this manipulation, guaranteeing itself two units of work instead of eight if it were alone or four units of expected work if it were honest in its deal making.

Phantom Letters

Let's look at another possibility for deception (figure 8). Let's say our agents both have to deliver letters to nodes b and c. It's an entirely symmetric situation. Obviously, it makes sense for one agent to go to b and one to go to c. However, b is relatively far away, and c is relatively close; each agent would prefer to be the one to go to c. If they tell the truth and declare their true tasks, they'll flip a coin to decide which one goes to which node.

Agent 1 again decides to manipulate the agreement. It announces that it has a letter to deliver to node d, which is a long way off in the direction of node c. It only makes sense for agent 1 to go to c and, presumably, continue to d. If agent 2 were given the right side of this route, it would have to do more work than when it's alone, which wouldn't be acceptable to agent 2. They have to agree that agent 2 goes to the left side and that agent 1 goes to the right side. Of course, the letter to d doesn't exist; agent 1 just goes to c and comes back and benefits from its manipulation.

Part of what's going on here is that the form of a deal that we defined has constrained the kinds of agreement the agents can come to. Remember, a deal is just a division of tasks, and we get certain discontinuities when we define a deal this way. However, it's really that our deal space is discrete that gives rise to some of these possibilities

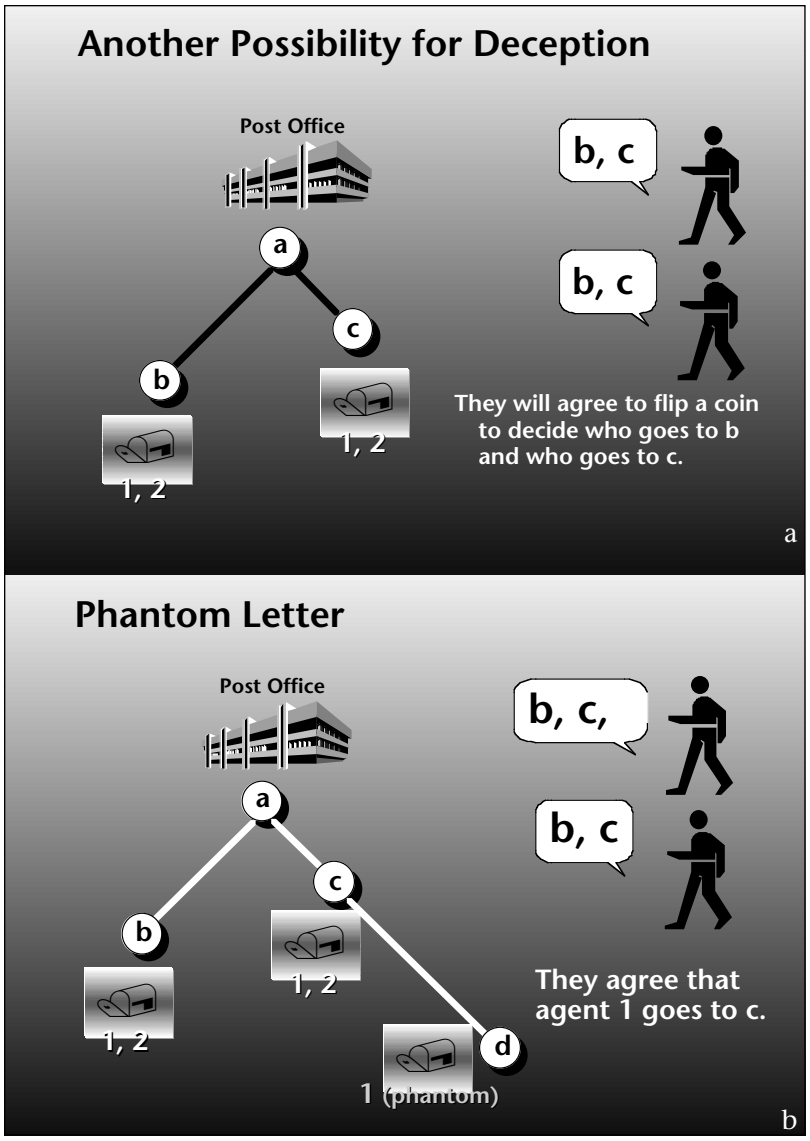

Figure 8. Creating a Phantom Task.

for deception. In other words, you have a certain limited number of ways to divide up tasks between agents. Depending on the particular encounter, an agent might be able to maneuver its way to a certain deal that's better for it, exploiting the fact that there are only certain ways to divide the tasks.

Negotiation over Mixed Deals

One straightforward way of getting rid of some deception is to make the deal space continuous. We can redefine what a deal is, what a division of tasks is, to include probability. A mixed deal is defined to be a division of tasks (D_1, D_2) with an associated probability p, so that with probability p, agent 1 does

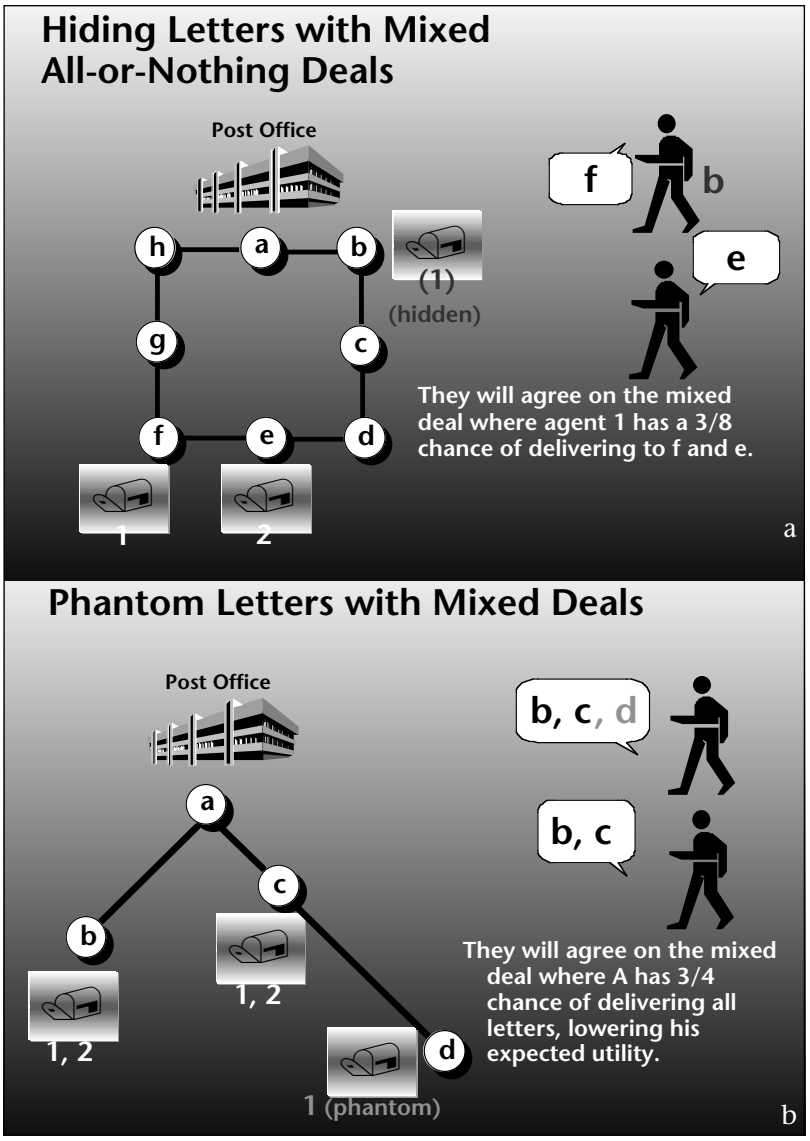

Figure 9. Mixed All-or-Nothing Deals Discouraging Deception.

task set D_1, and with probability 1 2 p, it does task set D_2 (and vice versa for agent 2). This deal definition results in a continuous space of deals. In addition, because of the way that our class of task-oriented domains is defined, if the agents use mixed deals, they can always restrict their agreements to the so-called all-or-nothing deal. In the all-or-nothing deal, all tasks are put into one big set, and a weighted coin is tossed to decide which agent does all the tasks. In our examples, this kind of agreement will always be a potential deal, but there can be others. By adding this probability into the deal definition, we make the space of deals continuous instead of discrete, the way it was originally.

Thus, at least some of the agents' possibilities for deception have vanished. Let's revisit our original postmen domain example. Now the protocol has agents negotiating over mixed all-or-nothing deals (figure 9). If agent 1 hides its letter to node b, it still has a certain probability of going all the way around the loop, which means that it doesn't get off for free, as in the original example. There is some possibility (less than 50 percent, but it's there) that it might deliver the declared letters. In any case, it still has a guaranteed trip of two units to node b, even if he wins the coin toss. If you work out the numbers, you see that agent 1 has not benefited any more from its hiding one of its tasks.

Similarly, in the second encounter, where agent 1 declared an extra, phantom task, the use of probability in the deal ends up worsening agent 1's position. It will end up doing extra work because it had the audacity to claim that it came into the encounter with extra work. The logic of maximizing the product of utilities here means that if it came into the encounter with extra work, it has to bear more of the burden of the final deal. In both these specific cases, we did well by just introducing probability into the deal definition.

Does adding probability really solve all our problems? No, but it removes problems for specific kinds of encounter. We have to understand the kinds of task-oriented domain that exist; not all task-oriented domains are the same.

Subadditive Task-Oriented Domains

All the examples we have given to this point have been illustrations of what are called *subadditive task-oriented domains*. A task-oriented domain is subadditive if for all finite sets of tasks, the cost of the union of tasks is less than or equal to the sum of the costs of the separate sets (for finite X,Y in T, $c(X \cup Y) \leq c(X) + c(Y)$). In other words, if you have two sets of tasks, and you put them together, you'll never increase the cost, and perhaps you'll decrease it. You can see the general idea in figure 10. Putting the sets of tasks together lowers the overall cost of the combined set.

All our examples have been subadditive, but not all task-oriented domains are subadditive. For example, consider a minor variation on the postmen domain; we call it the *delivery domain,* where agents go out and deliver their packages but are not required to return to the post office at the end of the day.

In this case, we don't have a subadditive domain (see figure 10b). If one agent has the task of delivering to the left node, and another agent has the task of delivering to the right node, then each has a task that costs one unit. The combined set of tasks costs three units: down one side, back to the original node, then down the other side; so, the combined tasks cost more than the sum of the individual tasks and, thus, are not subadditive. If this domain were the postmen domain, each separate delivery would cost two units, and the combination would cost four units and, thus, would be subadditive.

Incentive-Compatible Mechanisms

Now we can summarize what we know so far into a table that illustrates when lying is potentially advantageous and when telling the truth is the best policy (figure 11). For subadditive task-oriented domains, we have three protocols: (1) the original deal definition that we call pure deals; (2) the new deal definition that uses probability, which we call mixed deals; and (3) the all-or-nothing protocol that always results in this special mixed deal where one agent does everything with some probability. The original loop example, where a pure deal was being used in the protocol, was an instance where hiding a letter might be beneficial to an agent. The L signifies that encounters exist where lying is beneficial. The second example illustrates where a phantom task was beneficial; it also gets an L. A T in a box means that honesty is the best policy. An agent's best strategy is to always tell the truth. For example, when an all-or-nothing utility product-maximizing mechanism (PMM) protocol is being used in a subadditive task-oriented domain, no agent has an incentive to hide a task. The best strategy is to reveal all the tasks.

The entry T/P means that although creating a phantom letter might sometimes be beneficial, the deception might also be discovered because the nonexistent task might have to be handed over to the other agent using these probabilistic protocols. Thus, with a high enough penalty mechanism, telling the truth becomes the best strategy.

Also notice that there's a relationship between table entries, denoted by the white arrow. The relationships are implications that arise naturally from the definitions of the columns and rows. Here, for example, the fact that all-or-nothing deals are a subset of mixed deals means that if telling the truth is the best strategy in mixed deals, it is certainly also the best strategy in all-or-nothing deals, which are a subset.

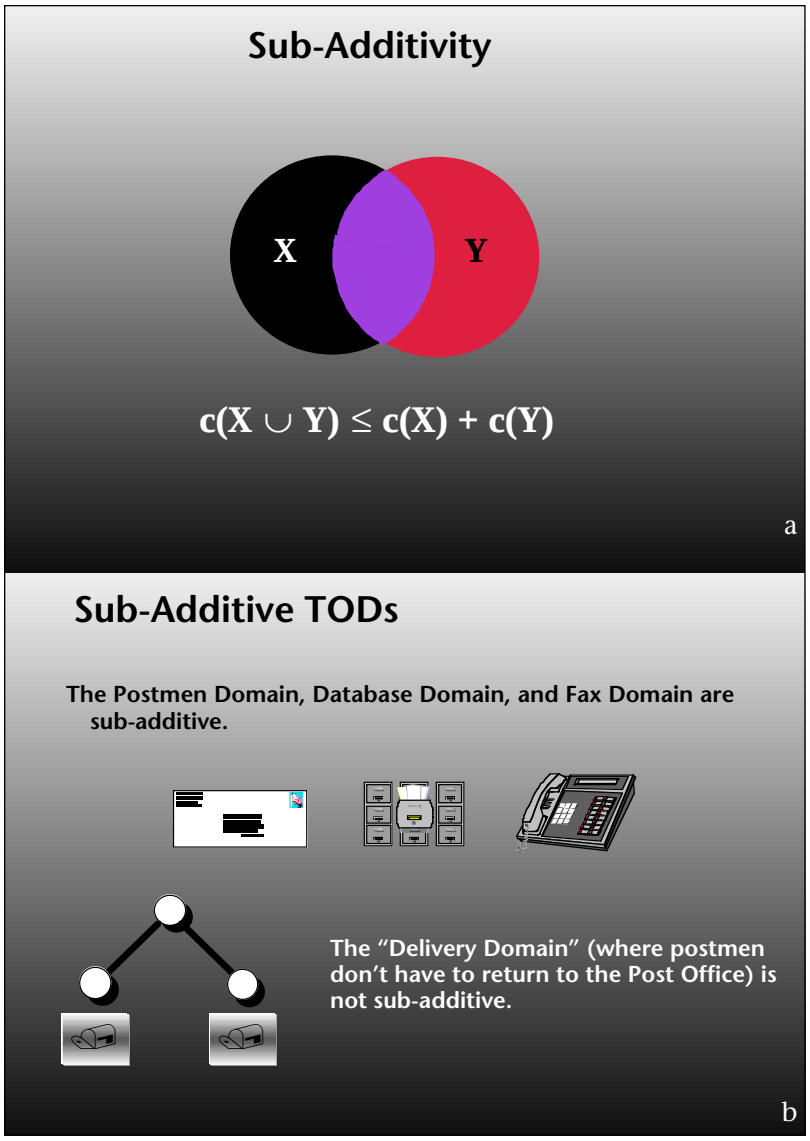

Figure 10. The Nature of Subadditivity.

Decoy Tasks

One more kind of lie that's worth looking at is the *decoy task*. A decoy task is like a phantom: It's a fake task, but an agent can create it on demand if necessary. A postman might claim it has a letter to some particular node. If required to hand it over when using a probabilistic deal, the agent quickly jots down a note, "Dear Resident: You might have already won the sweepstakes," and hands it over. Thus, a simple penalty mechanism won't work, and, in fact, as the table shows (figure 11b), decoy lies in subadditive domains can sometimes be beneficial for an agent even

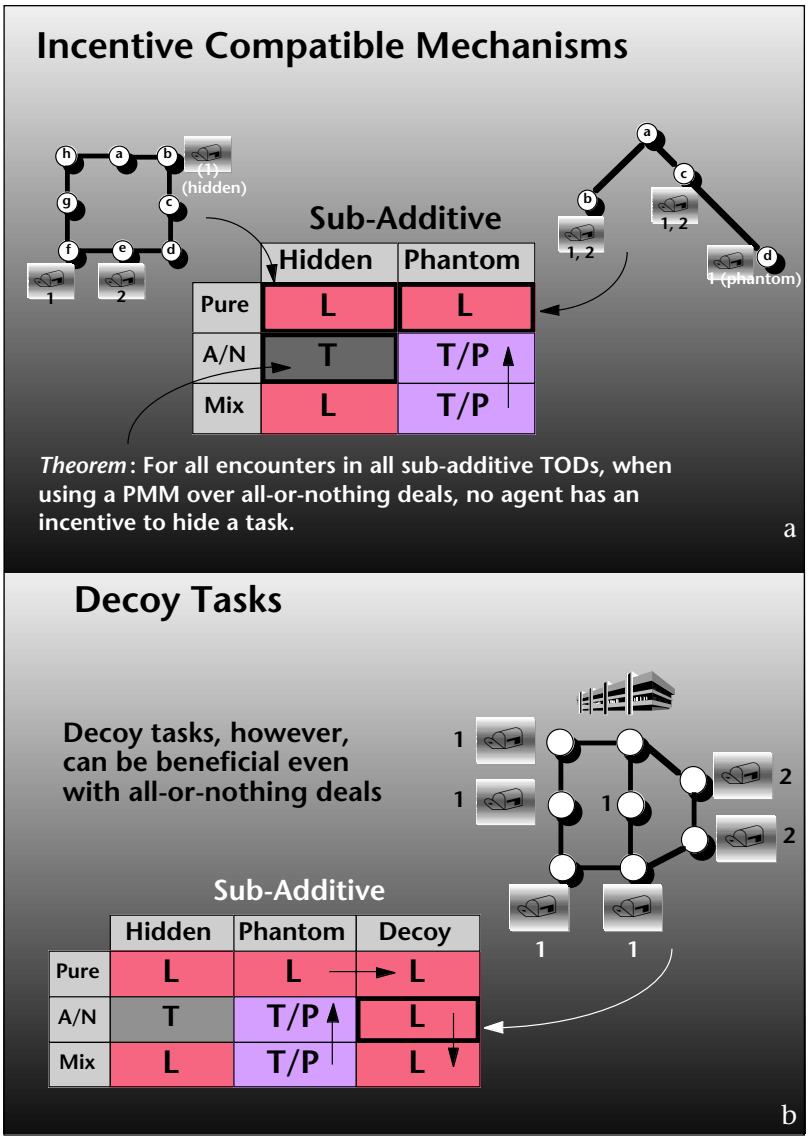

Figure 11. Tables Summarizing Strategies, Given Protocols.

when all-or-nothing protocols are used. Look at figure 11, where agent 1 would prefer to just deliver its own letters and return to the post office. By creating this decoy letter in the middle node, it claims that it would have to do a lot of extra work to carry out agent 2's delivery. It deceptively locks itself into its original path. Again, notice the white arrows. Because lying can be beneficial using an all-or-nothing protocol, it must also be beneficial sometimes when using mixed deals, which are a superset. Thus, having filled out one entry in the table, other entries can be implied automatically.

Subadditive task-oriented domains are an important domain class, but we can use other, more restrictive classifications to understand task-oriented domains.

Concave Task-Oriented Domains

Concave task-oriented domains are a subset of subadditive task-oriented domains that satisfy the following condition: Imagine that we have two sets of tasks, X and Y, where X is a subset of Y, and we come along with some other set of tasks, Z. The cost Z adds to X, the subset, is greater than or equal to the cost Z adds to Y, the superset: $c(X \cup Z) - c(X) \geq c(Y \cup Z) - c(Y)$. If a domain is concave, then it is also subadditive (figure 12).

Figure 12 is actually a bit misleading because it appears like such an obvious property that it ought to hold for all reasonable task-oriented domains, but it doesn't (see figure 12b).

Of the three task-oriented domains we introduced originally, only the database domain and the fax domain are concave. The general postmen domain is not concave unless graphs are restricted to trees. To see an example of a nonconcave encounter in the general postmen domain, consider the example we gave for a beneficial decoy task. Agent 1 has to travel around the left nodes; let's call it X. Agent 2 has to travel around the right nodes; let's call Y the set of all dark-gray nodes (that is, excluding the middle node marked Z). Thus, X, the left nodes marked with a 1, is a subset of Y, all the dark-gray nodes. Agent 1 lies with a decoy task to the node in the middle. This decoy task is set Z. The amount of work that Z adds to the set X is 0. The agent visits Z on the way at no extra cost. However, the amount of work that Z adds to Y, a superset of X, is 2. An agent would have to make a special trip to visit all Y, then visit Z; so, this example is not concave.

Three-Dimensional Incentive-Compatible Mechanism Table: If we return to the table that we've been setting up, we can examine an entirely new set of possibilities (figure 13). The table is now three dimensional; the black arrows show that relationships exist among the concave and the subadditive dimensions of the table. For example, if hiding letters is sometimes beneficial in concave domains when a pure deal protocol is used, then it will also sometimes be beneficial in subadditive domains because a concave domain is also always a subadditive domain.

The main thing to notice here is that concave domains are considerably better behaved

with regard to lying. There are more T's in the concave part of the table; in particular, we proved a theorem that says that in all concave task-oriented domains, when using all-or-nothing deals, no agent has any incentive to hide tasks or to create phantom or decoy tasks. There's absolutely no incentive for lying. It is the theorem that allows us to put the middle row of T's into the concave part of the table.

Modular Task-Oriented Domains

We can make an even more precise classification of task-oriented domains with the following definition: A *modular task-oriented domain* is one in which the cost of combining two sets of tasks, X and Y, into one large set is exactly the sum of their separate costs minus the cost of their intersection (you don't want to count the tasks that appear in both sets twice): $c(X \cup Y) = c(X) + c(Y) - c(X \cap Y)$. Any modular domain is also a concave domain as well as a subadditive domain.

In figure 14, the cost of the combined set of X union Y is exactly the cost of the set X plus the cost of the set Y minus the cost of the intersection of X and Y; the middle region isn't counted twice in the cost calculation.

Of the three original task-oriented domains we introduced, only the fax domain is modular. The general postmen domain is modular if you restrict the graphs that the postmen visit to a star topology (the post office in the middle and the other nodes connected to it like spokes on a wheel). Modular task-oriented domains are the most restrictive category and the best behaved with regard to lying, but even with modular task-oriented domains, hiding tasks can sometimes be beneficial if the agents are using general mixed-deal protocols.

Three-dimensional incentive-compatible mechanism table: Returning to our evolving table, we add another layer to the third dimension (figure 15). This modular layer has more T's. You can see that in moving from subadditive to concave to modular, we increase the percentage of T boxes, where telling the truth is always the best policy for an agent; however, there are still some residual L's lurking in the table.

Designers who together could determine that their domain was, for example, a modular task-oriented domain could look at this table and decide, perhaps, to use a protocol that has agents negotiating over all-or-nothing deals; the designers would be confident in the knowledge that none of the individual companies building the agents would have an incentive to conceal their tasks or create false ones. The best policy here is really just to tell the truth: simple, efficient, and stable.

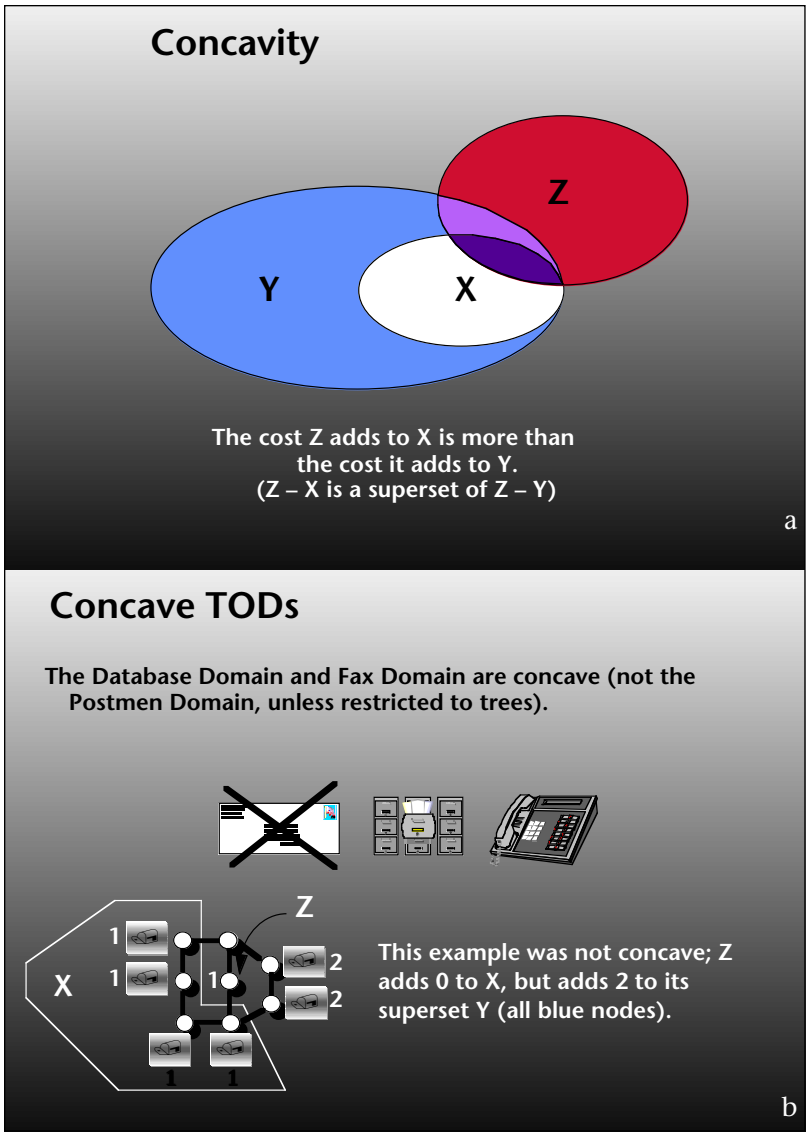

Figure 12. Concave Task-Oriented Domains.

In an article that appeared recently in the *Boston Globe* ("A New Dimension in Deception"), Michael Schrage talked about software agents that might choose to lie to further the aims of their owners. For example, a scheduling agent might falsely claim that its owner has an appointment at a certain time to force a group of people to set a meeting at a time the agent wants to have the meeting and not at some other time. Schrage was pretty good at laying out the scenario, but he missed one

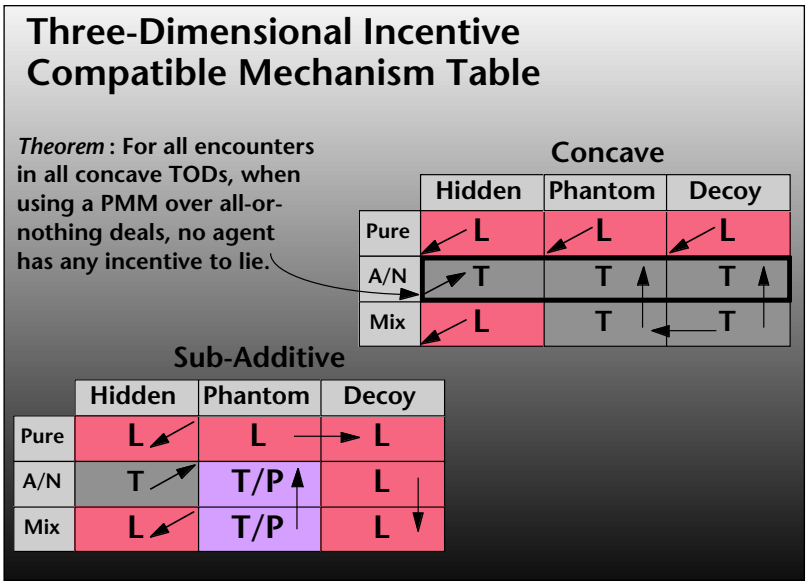

Figure 13. The Enlarged Strategy-Protocol Table.

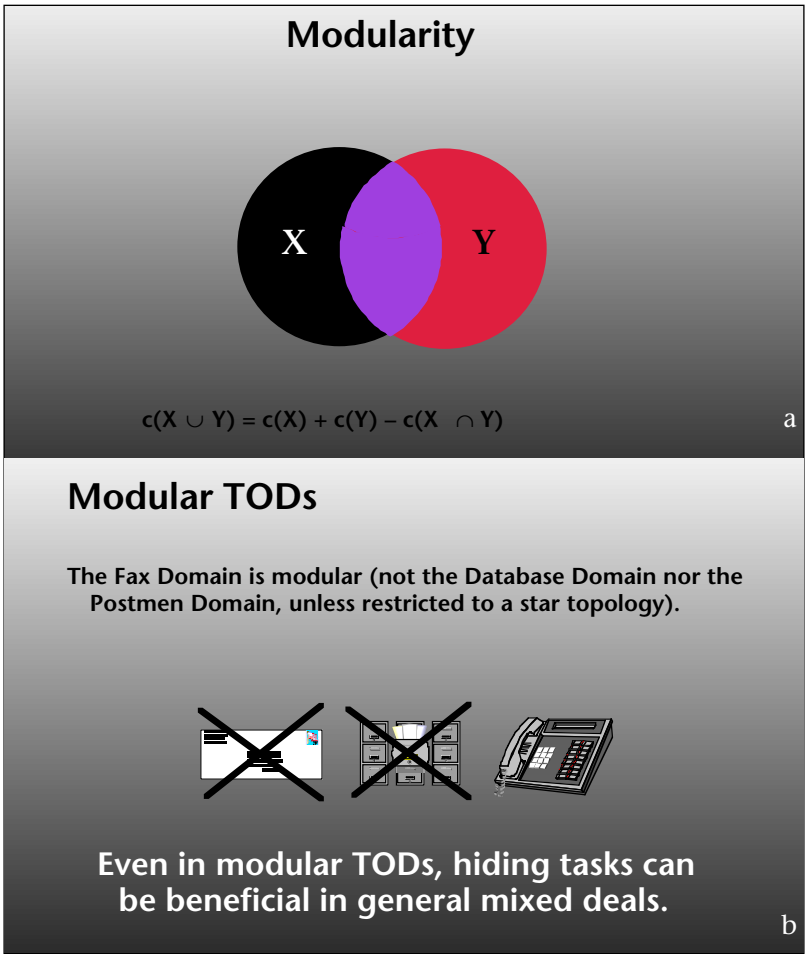

Figure 14. Modular Task-Oriented Domains.

crucial point: Sometimes, it's possible to design the rules of encounter so that lying is simply not in anyone's interest.

Related Work

This article only covers the tip of the iceberg. First, task-oriented domains are only a small class of encounters between agents; remember, we presented two other general classes: state-oriented domains and worth-oriented domains. We have carried out similar kinds of protocol analysis in these more general types of domain, and the situation becomes more complicated. Lying, for example, is harder to prevent in the more general encounters, but we can still analyze properties of protocols, such as efficiency and stability, and provide guidelines for how agent designers might want to build their systems.

Other work that's going on in this general direction within AI includes several recent papers on coalition formation where there are more than two agents (S. Ketchpel, S. Kraus, J. Rosenschein, and G. Zlotkin); general research into mechanism design (E. Ephrati, S. Kraus, and M. Tennenholtz); research on other models of negotiation among agents (S. Kraus, K. Sycara, E. Durfee, V. Lesser, L. Gasser, and P. Gmytrasiewicz); and research on other consensus mechanisms, such as voting techniques, explored in work that Jeffrey Rosenschein has carried out with E. Ephrati, and economic models, such as those being examined by M. Wellman.

Conclusions

What did we argue? First, by appropriately adjusting the rules of encounter by which agents must interact, we can influence the private strategies that designers will rationally build into their machines. When we can't have direct control of how multiple agents will be built, we can exert indirect influence by carefully designing the negotiation protocol.

Second, we pointed out that the interaction mechanism can and should be designed to ensure efficiency of the multiagent system.

Third, to maintain efficiency over time of dynamic multiagent systems, the rules must also be stable. It is not enough to figure out a strategy that has good properties, such as efficiency. The agent designers have to feel that they should stick with this strategy; they shouldn't have any incentive to move to another strategy. Stability is an important part of multiagent systems.

Finally, we did our analysis with the use of formal tools. Our commitment to the formal design and analysis of protocols both makes us more sensitive to issues such as efficiency and stability and gives us the ability to make definitive statements about them. Such tools give us the leverage we need to design interaction environments for automated negotiation.

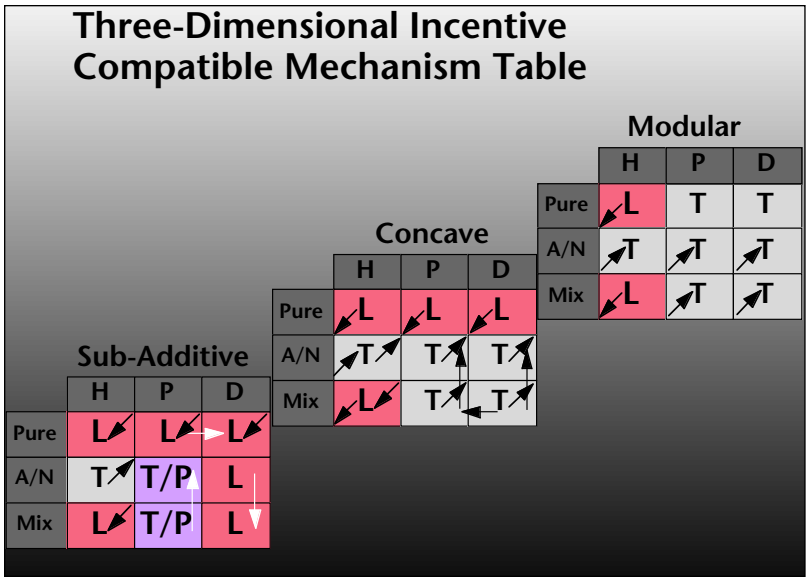

Figure 15. Categorizing Strategies Based on Domain and Protocol.

Acknowledgments

This research has partially been supported by the Leibniz Center for Research in Computer Science at the Hebrew University of Jerusalem and the Israeli Ministry of Science and Technology (grant 032-8284). The authors want to thank their many colleagues who have helped in the refinement of this research, including Erik Brynjolfsson, Edmund Durfee, Eithan Ephrati, Les Gasser, Mike Genesereth, Barbara Grosz, Robin Hanson, Sarit Kraus, Daniel Lehmann, Nati Linial, Ariel Rubinstein, Moshe Tennenholtz, Mario Tokoro, and Avi Wigderson.

Suggestions for Further Reading

Avouris, M., and Gasser, L. 1992. *Distributed Artificial Intelligence: Theory and Praxis*. Boston, Mass.: Kluwer Academic.

Binmore, K. 1992. Fun and Games: A Text on Game Theory. Lexington, Mass.: D. C. Heath.

Conry, S.; Meyer, R.; and Lesser, V. 1988. Multistage Negotiation in Distributed Planning. In *Readings in Distributed Artificial Intelligence,* eds. A. Bond and L. Gasser, 367–384. San Mateo, Calif.: Morgan Kaufmann.

Decker, K., and Lesser, V. 1993a. An Approach to Analyzing the Need for Meta-Level Communication. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 360–366. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Decker, K., and Lesser, V. 1993b. A One-Shot Dynamic Coordination Algorithm for Distributed Sensor Networks. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 210–216. Menlo Park, Calif.: American Association for Artificial Intelligence.

Ephrati, E., and Rosenschein, J. 1993. Distributed Consensus Mechanisms for Self-Interested Heterogeneous Agents. In First International Conference on Intelligent and Cooperative Information Systems, 71–79. Washington, D.C.: IEEE Computer Society.

Ephrati, E., and Rosenschein, J. 1992. Reaching Agreement through Partial Revelation of Preferences. In *Proceedings of the Tenth European Conference on Artificial Intelligence,* 229–233. Chichester, U.K.: Wiley.

Ephrati, E., and Rosenschein, J. 1991. The Clarke Tax as a Consensus Mechanism among Automated Agents. In Proceedings of the Ninth National Conference on Artificial Intelligence, 173–178. Menlo Park, Calif.: American Association for Artificial Intelligence.

Etzioni, O.; Leash, N.; and Segal, R. 1993. Building Softbots for UNIX, Preliminary Report, 93-09-01, Dept. of Computer Science, Univ. of Washington.

Fudenberg, D., and Tirole, J. 1992. *Game Theory.* Cambridge, Mass.: MIT Press.

Gasser, L. 1991. Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics. *Artificial Intelligence* 47(1–3): 107–138.

Gmytrasiewicz, P.; Durfee, E.; and Wehe, D. 1991a. A Decision-Theoretic Approach to Coordinating Multiagent Interactions. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 62–68. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Gmytrasiewicz, P.; Durfee, E.; and Wehe, D. 1991b. The Utility of Communication in Coordinating Intelligent Agents. In Proceedings of the Ninth National Conference on Artificial Intelligence, 166–172. Menlo Park, Calif.: American Association for Artificial Intelligence.

Harsanyi, J. 1977. Rational Behavior and Bargaining Equilibrium in Games and Social Situations. Cambridge, U.K.: Cambridge University Press.

Kraus, S. 1993. Agents Contracting Tasks in Non-Collaborative Environments. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 243–248. Menlo Park, Calif.: American Association for Artificial Intelligence.

Kraus, S., and Wilkenfeld, J. 1991. Negotiations over Time in a Multi-Agent Environment: Preliminary Report. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 56–61. Menlo Park, Calif.: International Joint Con-

ferences on Artificial Intelligence.

Kraus, S.; Wilkenfeld, J.; and Zlotkin, G. 1992. Multiagent Negotiation under Time Constraints, Technical Report, CS-TR-2975, Dept. of Computer Science, University of Maryland.

Luce, R., and Raiffa, H. 1957. *Games and Decisions*. New York: Wiley.

Moses, Y., and Tennenholtz, M. 1989. On Cooperation in a Multi-Entity Model. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 918–923. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Osborne, M., and Rubinstein, A. 1990. *Bargaining and Markets*. San Diego, Calif.: Academic.

Rosenschein, J., and Zlotkin, G. 1994. Rules of Encounter: Designing Conventions for Automated Negotiation among Computers. Cambridge, Mass.: MIT Press.

Roth, A. 1979. *Axiomatic Models of Bargaining*. Berlin: Springer-Verlag.

Shoham, Y., and Tennenholtz, M. 1992. On the Synthesis of Useful Social Laws for Artificial Agent Societies. In Proceedings of the Tenth National Conference on Artificial Intelligence, 276–281. Menlo Park, Calif.: American Association for Artificial Intelligence.

Zlotkin, G., and Rosenschein, J. 1993a. A Domain Theory for Task-Oriented Negotiation. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 416–422. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Zlotkin, G., and Rosenschein, J. 1993b. Negotiation with Incomplete Information about Worth: Strict versus Tolerant Mechanisms. In Proceedings of the International Conference on Intelligent and Cooperative Information Systems, 175–184. Washington, D.C.: IEEE Computer Society.

Zlotkin, G., and Rosenschein, J. 1993c. The Extent of Cooperation in State-Oriented Domains: Negotiation among Tidy Agents. *Computers and Artificial Intelligence* 12(2): 105–122.

Zlotkin, G., and Rosenschein, J. 1991a. Cooperation and Conflict Resolution via Negotiation among Autonomous Agents in Noncooperative Domains. *IEEE Transactions on Systems, Man, and Cybernetics* 21(6): 1317–1324.

Zlotkin, G., and Rosenschein, J. 1991b. Incomplete Information and Deception in Multi-Agent Negotiation. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 225–231. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Zlotkin, G., and Rosenschein, J. 1990. Negotiation and Conflict Resolution in Non-Cooperative Domains. In Proceedings of the Eighth National Conference on Artificial Intelligence, 100–105. Menlo Park, Calif.: American Association for Artificial Intelligence.

Zlotkin, G., and Rosenschein, J. 1989. Negotiation and Task Sharing among Autonomous Agents in Cooperative Domains. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 912–917. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Jeffrey S. Rosenschein received his B.A. in applied mathematics from Harvard University (1979) and his M.S. (1981) and Ph.D. (1986) in computer science from Stanford University. He is currently a senior lecturer at the Institute of Computer Science at the Hebrew University, Jerusalem, Israel. His research interests revolve around issues of cooperation and competition among agents and the use of economic theory, voting theory, and game theory to establish appropriate foundations for distributed AI. He and Gilad Zlotkin cowrote a book for MIT Press (1994) on their research entitled "Rules of Encounter: Designing Conventions for Automated Negotiation among Computers."

Gilad Zlotkin received his Ph.D. in computer science from the Hebrew University in Jerusalem, Israel. He is currently a research fellow at the Center for Coordination Science at the Sloan School of Management at the Massachusetts Institute of Technology. His research involves the use of game and group decision theories to design automated coordination mechanisms. His current research activities include coordination theory within the scope of business process reengineering, mechanisms for coalition formation, and meeting-scheduling systems.