Consenting Agents: Designing Conventions for Automated Negotiation

Jeffrey S. Rosenschein
Institute of Computer Science
Hebrew University
Givat Ram, Jerusalem
Israel
jeff@cs.huji.ac.il

Gilad Zlotkin
Center for Coordination Science
Sloan School of Management, MIT
1 Amherst Street, E40-179
Cambridge, MA 02139 USA
gilad@mit.edu

This is the adapted text of an invited lecture given by the first author at IJCAI93, in Chambery, France, on September 2, 1993. Slides that were used during the talk are included as figures in the text below. First-person phrasing has also been maintained in the text.

Abstract: 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 here 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.

Keywords: multi-agent systems, negotiation, distributed artificial intelligence

1 Introduction

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 home or office, has some kind of "information appliance" that can handle the wealth of resources that are available.

Now, artificial intelligence 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 in Seattle is building what he calls SoftBots, software robots, to handle the interface between humans and network resources. And there are lot of good, meaty, classic AI problems that need to be solved in this context, like knowledge representation and planning problems.

What I'm going to talk about today has a very strong connection to those efforts at building software agents. What I've been interested in over the last few years has been to look at the ways in which these software agents, automated agents, will be dealing with one another. In other words, it's all very well and good to have agents residing in your home computer that help you deal with all that information, but that agent of yours is going to have to deal with agents of other people, either private agents if you're trying to do some job like setting up a meeting with a group of people, or agents belonging to a company 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, negotiating with each other?

1.1 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 in concert with other machines. Computers that control electrical grid networks, and need to load 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 telecommunication networks, routing. Information or packets may pass over one network, controlled by one company, and then pass onto another network controlled by another company, or pass through one country and then pass on through another country. Computers that control a telecommunications network might find it beneficial to enter into agreements with other computers, controlling other networks, about routing packets more efficiently from source to destination. The point is, 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.

We're also seeing the emergence of things like personal digital assistants, small, hand-held computers, like the Newton, which more and more of us will start to carry around with us all the time. These PDAs are going to take over a lot of roles in managing our daily lives, as notebooks, as communicators, as fax machines, telephones, and automated schedulers. We're going to have some kind of agent software on that PDA, and it's ultimately going to get part of its work done interacting with other agents on other PDAs.

Another example is the proliferation of shared databases, where there's information spread all over the world. It's something we've seen spring up with a vengeance in the last decade. You've already got agents that are going out there to gather information, like Etzioni's SoftBots, that go into a shared database to gather information about, for example, a person's Internet address. Finally, even something like traffic control, coordination of vehicular traffic, or air traffic control, stands as an example 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 that resource with other computers. With PDAs, the resource might be a person's time, while with a telecommunications network the resource might be communication lines, switching nodes, or short and long-term storage. But in each situation, the computers that control these resources can do their own job better by reaching agreements with other computers.

1.2 Heterogeneous, Self-motivated Agents

Now, the agents that I'm 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 and 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 your idea of utility, is interested in furthering 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 very flexible, it's very dynamic, new PDAs are coming in, even new *types* of PDAs are being built and coming in to the system and have to interact with other agents.

And in particular, very importantly, they'll not act benevolently unless it's in their interest to do so. They will not necessarily share information, they will not necessarily do things that other agents ask them to do unless they have a good reason for doing so. So 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.

1.3 The Aim of the Research

Now, 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. What do I mean by "social engineering"? You know what social engineering is when we talk about communities of people. It 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 have decided ahead of time are good ways for them to act. At the very least, it constrains their behavior.

Well 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 of looking at the research is that we're trying to exploit formal tools, in this case game theory 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, those PDA agents to act in certain ways.

1.4 Broad Working Assumption

Those 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. But who's "we," right? What do I mean by "we"? Well, "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, Sony all come together, and they say, "OK, we've got this domain. The domain is personal digital assistants doing scheduling, time scheduling. Given that 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 that are going to be like a high-level protocol that determines the kind of deals, let's say, that agents can make among themselves." We'll get more into the details of deal-making later.

So these designers come together, they try to agree on standards for how their automated agents interact. And an important part of this is that it's in a given domain. They might decide on different protocols for different domains. But given the domain, like scheduling among PDAs, they're going to decide on standards for how the agents reach agreements.

Now, what are they actually doing in this meeting? See, this is a standards committee meeting, they sit around a big table, and they start discussing the tradeoffs of different decisions. And one of them says, "You know, if we have this kind of protocol, the agents will be able to come to agreement very quickly. The agreements probably won't be optimal, but we'll be able to get to them very fast." And somebody else at the table says, "Yes, but it's very 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." And 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, we the researchers here, are not trying to impose a group of decisions that

these company representatives are going to make. Instead, what we're trying to do is come forward, elucidate a variety of protocol decisions and show how certain protocols have certain desirable attributes. Once we have shown that protocol A has a certain attribute, the designers of those PDAs may decide to choose it and they might not, they might choose something else. But the idea of the research is to come forward and elucidate the possibilities, elaborate on the possibilities. Protocol A has this set of attributes, 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.

1.5 Attributes of Standards

Now, what are the sorts of things we're looking at when we try to design a standard? Well, we might look at things like efficiency of the system, things like Pareto Optimality, meaning that the agreement that's 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. So that might be a broadly acceptable doctrine of general goodness.

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

Simplicity: this is certainly a very important one, by the way, for computer science, much, much less important for game theorists. We certainly want our protocols to have low computation and communication costs. This is an attribute of the standard that we might propose and those designers might say, "This is absolutely important, it's very important that our small processor not have a very heavy load in doing negotiation," let's say.

We'd like the protocol to be distributed, usually, 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.

So the idea is to design protocols for specific classes of domains that satisfy some or all of these attributes. As I present protocols, or design decisions, it may be the case that it's very simple but it's not distributed, or that it's stable but it's not efficient, or efficient but not stable. That's a classic tradeoff.

1.6 Distributed Artificial Intelligence

How does this work relate to other research that people are doing out in the field? Well, it fits into the broad area of distributed artificial intelligence, which itself can be broken down into two, related areas. Now these two areas constitute distinctions between research agendas; they're not really appropriate as descriptions of running systems.

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

This contrasts with another stream of research within distributed artificial intelligence called "Multi-Agent Systems." In multi-agent 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 don't assume that the individual agents have a group sense of utility, a group notion of utility. Each of the agents in the system may be working at different goals, even conflicting goals. So you have to deal with a system made up of multiple agents where there's going to be possibly competition between the agents, possibly cooperation. In Multi-Agent Systems, there is assumed to be **no single body** who is able, at design time, to directly influence the preferences of all agents in the system. The agents' preferences arise from distinct designers.

In particular, if a DPS researcher can show that acting in a particular way is good for the system as a whole, he can *impose* this behavior on all the agents in the system at design time. For the MAS researcher, such an alternative is unavailable. At best, he 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 MAS research from DPS research.

The work that I am describing today is Multi-Agent Systems research.

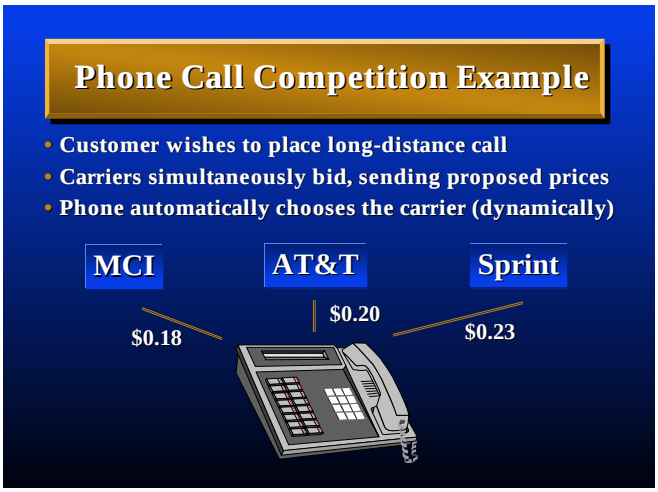

Figure 1: A System for Placing a Call

2 The Phone Call Competition Example

I've mentioned how we're trying to design protocols for agent interactions. To illustrate this point, I want to go through an example of a hypothetical environment, and 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 phone customer sends in a postcard to hook up with one or another of them. That company becomes their default carrier, and you have to dial extra numbers to place a long-distance call with some other company. What if we imagined another kind of system, one that is perfectly well within the current technology, and has certain benefits over the way things are done now? What if, when a customer lifts the handset and dials a long-distance call, a microprocessor within the phone automatically collected bids from the various carriers? Each company's computer automatically and simultaneously declares the price per minute that it's willing to carry the call. So here we see the MCI computer relaying 18 cents, while the AT&T computer bids 20 cents, and so on [see Figure 1]. This can all be done in a split second, and without really delaying the call significantly.

2.1 Best Bid Wins

Now, our phone's microprocessor collects these bids. Assume that the protocol involves our phone choosing the company with the lowest bid, which is completely

reasonable, and placing the telephone call with that company. The winning carrier receives a price per minute equal to the amount that it bid, and that's that. OK, so we have a pretty interesting system set up here. 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 it 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 a great motivation to sink their budgets into research on how to lower their costs, rather than into advertising campaigns to grab consumers. Now, I'm not really analyzing all the ramifications of this scenario, but I'm bringing it mainly to illustrate a point, which we'll get to in a moment.

So far, so good. But this bidding mechanism has a serious flaw [see Figure 2].

Figure 2: Rules of the Game, and Resulting Attributes

2.2 Attributes of the Mechanism

We have managed to make the long-distance carrier selection be distributed and symmetric. But the protocol, the rules by which the agents play to win the telephone call, does not encourage stable, simple, nor efficient behavior on the part of the telephone company computers. To see why this is so, 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 that profit by bidding higher. It might

think that the next highest bidder won't go below 22 cents, and consequently make its own bid be 21 cents. Now, that's a risky strategy, but the point is that the carriers have a great incentive to invest 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, how much can carrier X *really* afford to carry a call for *right now*? They'll invest effort in trying to find out relevant information about their opponents, which switching stations are down, how's the other company's profit and loss for this quarter, all sorts of things that might affect the bid the opponent puts in. Ultimately, that sort of effort drains resources that might be better spent elsewhere. Equally important, the actual bidding procedure may result in an inefficient outcome. In this example, MCI may 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.

2.3 Best Bid Wins, Gets Second Price

Let's say that in our new protocol, all the company computers put their bids in, and our phone's computer again automatically chooses the lowest bidder as the winner. However, this time, the carrier that wins gets paid a price per minute equal to the second lowest bid. This bidding system, called Vickrey's Mechanism, has the attractive property that it provides no incentive for a company to underbid, nor to overbid. A company has an incentive only to provide the true, minimum acceptable price. A company won't bid lower than it's minimum acceptable price, because it fears that some other company might bid in the gap that it's opened up (and 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 else might bid 10 cents; then MCI would be forced to carry the call for less than its minimal acceptable 18 cents per minute. On the other hand, 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've fundamentally altered the way in which computers should play the game [see Figure 3].

Figure 3: Different Protocol, Different Attributes

2.4 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 be using. Of course, we've bought these wonderful attributes at a cost; the consumer has to pay a hopefully small premium on each call to make things work, in this example paying 20 cents rather than 18 cents per minute. With many carriers, this effect will be minimized, but in any case, the whole point of showing this example are not its details, but rather to illustrate the overall idea, of 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, and we're interested in more complicated situations, with computers controlling and sharing resources, real-world situations.

3 Domain Theory

I mentioned before that it is very 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, that motivates agents to act in a certain way in one type of domain, 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 more general than the last. This is not exhaustive; there are other, more general categorizations of domains that go beyond this. But let me go over these three because they're both interesting, and cover a lot of the real-world domains in which we're interested.

The lowest level, the simplest kinds of domain that we've looked at, are called Task Oriented Domains (TODs). A Task Oriented Domain exists when agents have non-conflicting jobs to do, and these jobs or tasks can be redistributed among the agents. So, the agents receive some list of jobs that they have to accomplish and the object of negotiation in these kinds of environments is to redistribute tasks among the agents, for everyone's mutual benefit, if that is possible. Most of the talk will be on this first area.

The next higher level we call State Oriented Domains. State Oriented Domains are a superset of Task Oriented Domains. State Oriented Domains have goals that specify acceptable final states, in the classic artificial intelligence way. Very importantly, and this is probably the critical aspect of State Oriented Domains in contrast to Task Oriented, actions can have side effects. In Task Oriented Domains there are *never* any side effects. State Oriented Domains have side effects, and an agent doing one action might hinder another agent, or might help the other agent. The object of negotiation is to develop joint plans and schedules for the agents. What they're trying to do is to figure out when each agent should do each action, mainly to stay out of each other's way but also to help one other if appropriate.

Finally, we come to Worth Oriented Domains, which are a superset of State Oriented Domains. Worth Oriented Domains assume, like State Oriented Domains, that there are goals that specify final states, but that fact is encoded in a function that rates states' acceptability. Every state in the world is better or worse, but it's not this 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 also goal relaxation. In other words, agents may not be able to get to the state that is their ultimate objective, but be willing to arrive at a state that is a little bit worse. Because the agents have a function that rates states' acceptability they're able to evaluate gradations among goal states, which they couldn't do in State Oriented Domains.

3.1 Examples of Task Oriented Domains

3.1.1 Postmen Domain

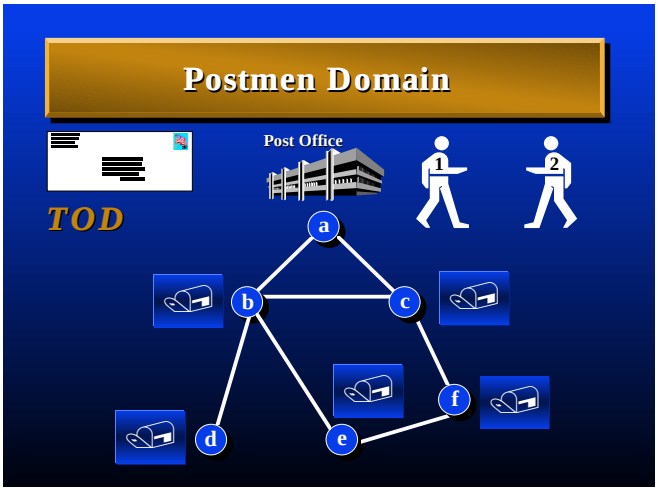

Figure 4: Example of Task Oriented Domain

Let's look at some examples, to see what I mean by this hierarchy [see Figure 4]. Here's a classic Task Oriented Domain that we've looked at quite a bit, called the Postmen Domain. In this case two agents arrive at the post office early in the morning and 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, and f, and e, and then return to the post office, while the other agent might have to go to c, b, and d, and then return to the post office. That's a Task Oriented Domain. The cost of carrying out a delivery is only in the travel distance. There are no side effects to what they are doing, there is no limit to the number of letters they can carry, there's no limit to how many letters they can put into a mailbox. There's no worry about hitting each other. They just have these tasks they're going to do. No possibility of getting in each other's way.

So what the agents want to do, is that they would like to cooperate, before they start out on their journey, 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? No extra cost to you, the cost is only in the travel distance, and I'll have a shorter trip." And what they're trying to do is to evaluate these deals. There are different possible divisions of the tasks, some are better for one, some are better for the other, and

we would like the agents to come to some agreement about how they are going to divide up the tasks.

3.1.2 Database Domain

Let's look at another domain, it's 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 get all the female employees making over $50,000 a year, and return with the names, the other one is supposed to get all female employees with more than three children, and bring back the names. In this domain, each subquery to the database costs money. Now these two agents approach the database, and they 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 that subset of names. We don't have to *both* ask for all female employees." So this is another Task Oriented Domain. No side effects, no getting in each other's way, just the possibility for cooperation [see Figure 5].

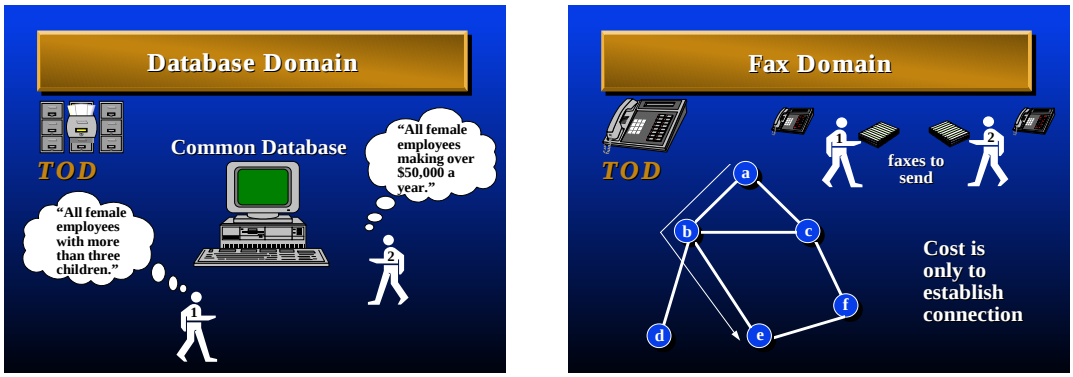

Figure 5: More Examples of Task Oriented Domains

3.1.3 Fax Domain

One final example of a Task Oriented Domain, one similar to the Postmen Domain, is called the Fax Domain. In the Fax Domain two agents arrive in the morning and they are given lists of faxes they have to send all over the world. The agents fortunately only have to pay for connecting to this other fax machine, and

once they connect they are allowed to download as many faxes as they want. The cost is all in establishing the connection.

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

3.2 Slotted Blocks World—State Oriented Domain

Figure 6: State Oriented Domain and Worth Oriented Domain Examples

Now what do I mean by a State Oriented Domain? Well, the Blocks World, that's an example of a State Oriented Domain. An agent comes and he wants the blocks in a certain configuration. The other agent comes and *he* wants the blocks in a certain configuration. Now these goals may be identical, or they may be in conflict with one another. This agent may want the orange block on top of the blue block, and this agent might want to have the blue block on top of the orange block. There's a possibility of real conflict [see Figure 6].

There are other possibilities too, like accidental cooperative action, where one agent inadvertently does something that's good for the other without the other one having to ask for it. This is not 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 there are side effects that really affect our analysis of the domains.

3.3 The Multi-Agent Tileworld—Worth Oriented Domain

Let's take one more example, of a Worth Oriented Domain. This is a multi-agent version of the Tileworld, originally introduced by Martha Pollack. The basic idea is that 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. Now it's true that there is a most desirable state that each agent has, where all of the tiles are in his holes and so on, but there are also other states that are good, though less good than that most desirable state. So each agent is able to rank different states, and agreements can more easily reflect the possibility of compromise.

4 Task Oriented Domain (TOD)

Let's go back now and look at those Task Oriented Domains. Here's a more formal definition. A TOD 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, then, 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 his set. You might as well also call that task set his "goal." That's an encounter, a group of agents coming together, each one with a list of tasks.

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

4.1 Building Blocks

In doing this analysis, we have three things we would like to look at. The first is, a precise **specification of the domain**, a definition of what a goal is and what agent operations are. We just did that in a broad sense for a Task Oriented Domain.

There are two other pieces of what we would like to do. The first thing is to design a **negotiation protocol**, for the domain. Now 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 being sort of like the

"Rules of the Game." An analogy would be in chess: if I told you what kind of move each piece could make, well that's the negotiation protocol. I define what all the moves are. What are the possible moves in this negotiation, what kinds of deals can be offered, and so on.

Ah, but what's missing? A **negotiation strategy**. A negotiation strategy is how an agent should act given the set of rules. Think about it, think about a chess game. Think about breaking apart on the one hand, the rules that describe the game from, on the other hand, the technique that an agent is going to use 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.

4.2 Deal and Utility in 2-Agent TOD

So first we would like to have a definition of a deal, and 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 $\Theta \equiv (T_1, T_2)$
- Utility$_i(\delta) =$ Cost$(T_i) -$ Cost(D_i)

A nice simple definition. A deal in a two-agent TOD, is a pair, $D1, D2$, 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. You don't reach agreement, you deliver your original sack of letters.

Utility of a deal for an agent we define to be the cost of his original work minus the cost of his new work, given the deal. The difference is how much he has gained from the deal. He used to have to walk five miles, now he only has to walk three miles, his utility is two.

4.3 Negotiation Protocols

Now as far as the protocol that the agents are going to use, actually there are lots of good choices. For the purposes of the rest of this discussion, we're going to

assume that the agents are using some kind of product maximizing negotiation protocol, like in Nash bargaining theory. It really doesn't matter which one they use, for our purposes, as long as it's symmetric, maximizes the product of the utilities. There are all sorts of examples of different protocols, different rules, that will bring the agents to come 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 and that will be the point they jump to. Or you can instead have some kind of a monotonic concession protocol, where the agents first start with a deal that's best for them and then iteratively make compromises to the other agent. There are a lot of different product maximizing protocols.

But now we return to the last item on our Building Blocks list. Given that we have set up a Task Oriented Domain specification, a definition of a deal, utility and the conflict deal, and an overall protocol, what is the **negotiation strategy** that the agents should use? Now, given the discussion above, you could say, "Well, strategy is not really very 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. There a well-defined point that tells them when they've maximized the product of their utilities. They can just move to that point, one way or another. it doesn't really matter how."

And that's true, when the agents have complete information.

5 Negotiation with Incomplete Information

But what about the case where they 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 to f, and the second agent has to go to e. They don't know each other's letters, so they can't simply 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. So one very simple, straightforward technique for doing this is to set up a -1 phase game, a sort of pre-game exchange of information [see 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 he 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 8.

Agent 1 would certainly go all the way around, and 2 might go half way around and back, but it's equivalent for him to going all the way around. In this particular case, because of the structure of the problem, the agents will eventually come to the agreement where they flip a coin, and one of them travels all the way around while the other one stays in the Post Office. And assuming it's a fair coin, they've divided up the work equally.

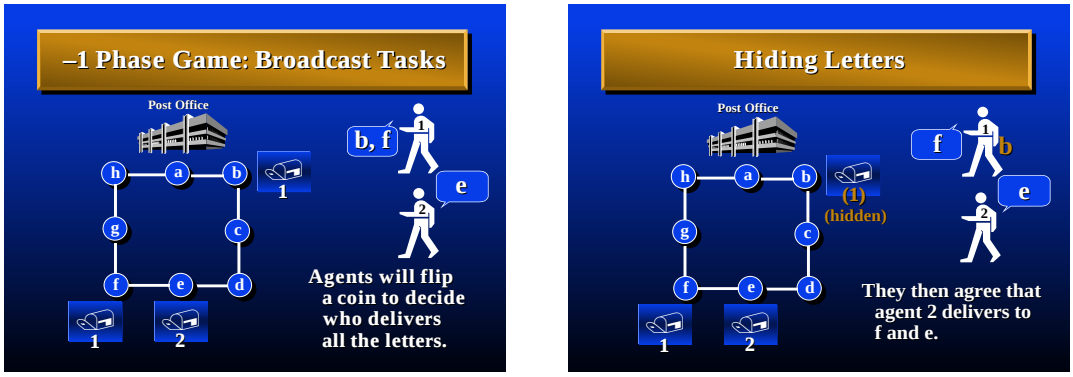

Figure 7: Dealing with (and Exploiting) Incomplete Information

5.1 Hiding Letters

See, but our intrepid agent has been built by a smart group of designers, and he makes the following claim. Agent 2 honestly says "I have to go to e," agent 1 says "I have to go to f" and he hides his letter to b [see the right side of Figure 7]. Now, the negotiation situation has changed. The negotiation situation has changed because agent 1 is purporting to say here that he only has to travel 6, and he 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, while agent 1 supposedly does nothing. The reason this is the agreed-upon deal is that it would not be rational for agent 1 to agree to carry letters all the way around the loop. Then it would be doing 8 units of work, more than the 6 units of work it would supposedly be doing by itself. And it can't be expected to agree to a deal that makes it do extra work; that wouldn't be rational. On the other hand, agent 2 doesn't benefit from this deal, it still travels 8, but it isn't harmed, either. So 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 his hidden letter to b, at a cost of 2 units. So agent 1 has really made off very well with this manipulation,

guaranteeing himself 2 units of work instead of 8 if he were alone, or even 4 units if he were honest in his deal-making.

5.2 Phantom Letters

Let's look at another possibility for deception [see 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. But 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.

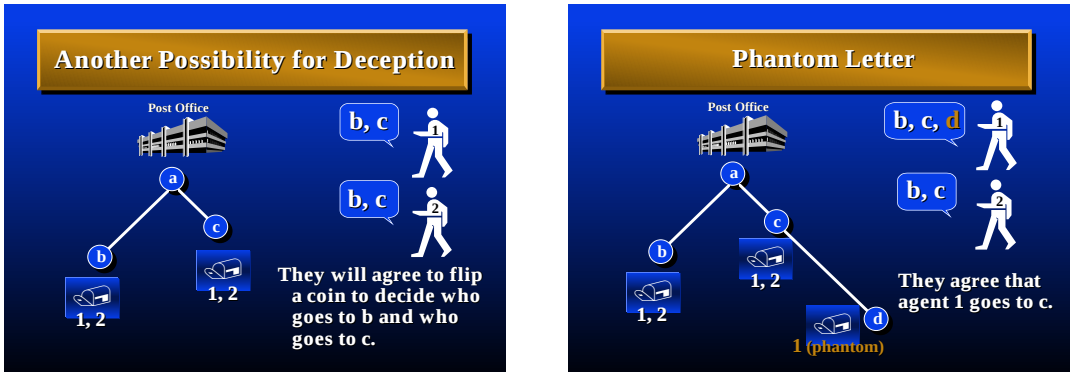

Figure 8: Creating a Phantom Task

But agent 1 again decides to manipulate the agreement. He announces that he has a letter to deliver to node d, which is a long way off in the direction of node c. So now, it only makes sense for agent 1 to go to c, and presumably continue on 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, and that wouldn't be acceptable to agent 2. So they have to agree that agent 2 goes to the left side, and 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 his 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 that way. But it's really this fact that our deal space is discrete that gives rise to some of these possibilities for deception. In other words, you have a certain limited number of ways of dividing up tasks between agents. And depending on

the particular encounter, an agent might be able to maneuver his way to a certain deal that's better for him, exploiting the fact that there are only certain ways of dividing the tasks.

5.3 Negotiation over Mixed Deals

So 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 task set D_1, and with probability $1 - p$ it does task set D_2 (and vice versa for agent 2). This 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, meaning the deal where all tasks are put into one big set, and a weighted coin is tossed to decide which agent does *all* the tasks. This kind of agreement, an all-or-nothing agreement, will in our examples always be a potential deal; there may be others, of course.

So by adding this probability into the deal definition, we've managed to make the space of deals be continuous, instead of discrete, the way it was originally.

And that means that at least some of the agents' possibilities for deception have vanished. Let's revisit our original Postmen Domain example, but now the protocol has agents negotiating over mixed all-or-nothing deals [see Figure 9]. *Now*, if agent 1 hides his letter to node b, he still has a certain probability of going all the way around the loop. Which means that he doesn't get off for free, the way he did in the original example. There is some possibility (less than $1/2$, but it's there) that he might deliver the declared letters. In any case, he still has a guaranteed trip of 2 to node b, even if he wins the coin toss. If you work out the numbers, you see that agent 1 has not benefited anymore from his hiding one of his 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. He'll end up doing extra work, because he had the audacity to claim that he came into the encounter with extra work to begin with. The logic of maximizing the product of utilities here means that if he came into the encounter with extra work, he has to bear more of the burden of the final deal. So in both of these specific cases, we've done pretty well by just introducing probability into the deal definition.

But what's really going on here? Does adding in probability really solve all

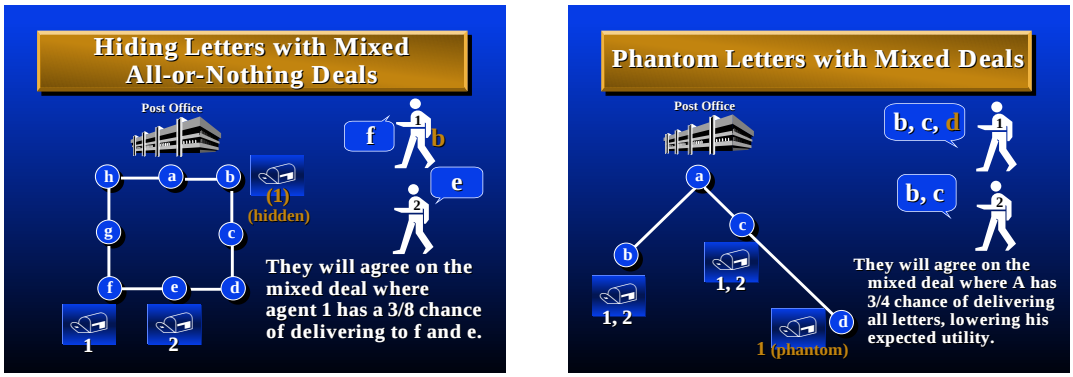

Figure 9: Mixed All-or-Nothing Deals Discouraging Deception

our problems? And the answer is no, it removes problems for specific kinds of encounters. So we have to understand the kinds of Task Oriented Domains that exist; not all Task Oriented Domains are the same.

6 SubAdditive TODs

You see, all the examples we've given so far, have been examples of what are called "subadditive" Task Oriented Domains. A TOD 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 the diagram [see Figure 10]. Putting the sets of tasks together lowers the overall cost of the combined set. That's the meaning of subadditivity.

Now, all the examples we've looked at so far have been subadditive, the Postmen Domain, the Database Domain, and the Fax Domain. But not all TODs are necessarily 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 that case, we don't have a subadditive domain anymore [see the right side of Figure 10]. 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 1 unit. The combined set of tasks costs 3 units, down one side, back to the original node, then down the other side. So the combined tasks costs more than the sum

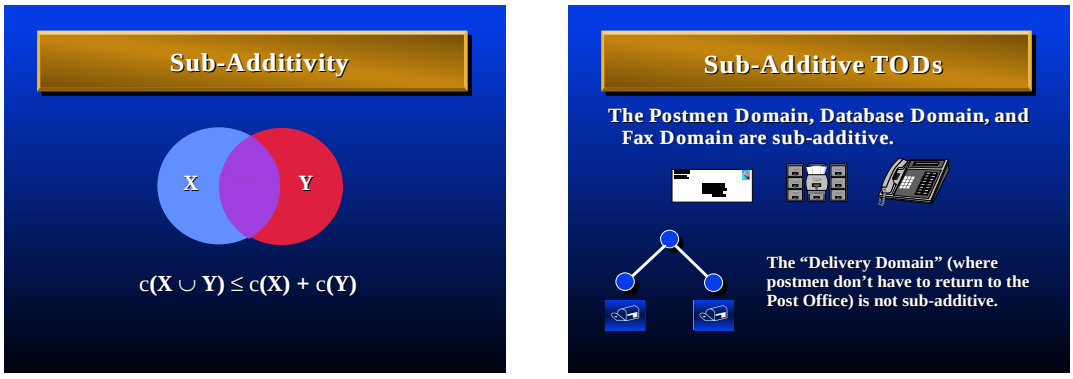

Figure 10: The Nature of Subadditivity

of the individual tasks. This is not subadditive. If this were the Postmen Domain, each separate delivery would cost 2 units, the combination would cost 4 units, and that *is* subadditive.

6.1 Incentive Compatible Mechanisms

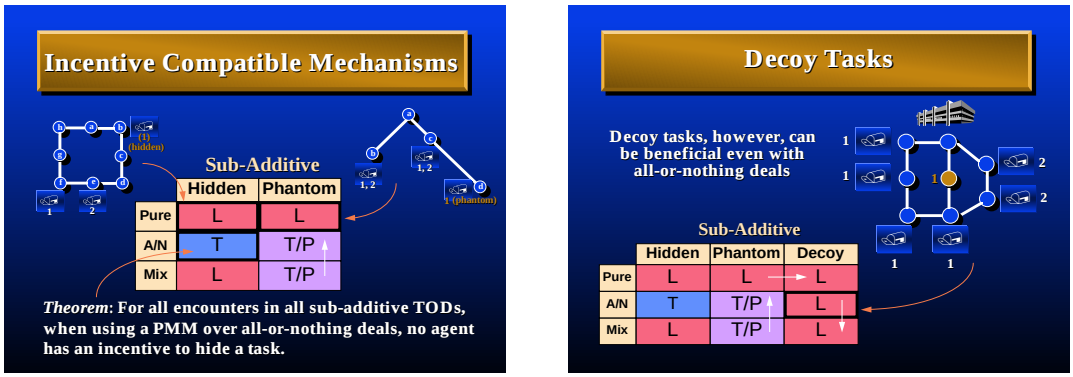

Figure 11: Tables Summarizing Strategies, Given Protocols

Now we can summarize what we know so far into a table, that illustrates when lying is potentially advantageous, and when truth-telling is the best policy [see Figure 11]. Here we have, for subadditive TODs, three possible protocols, the original deal definition which we call Pure Deals, the new deal definition that uses probability, Mixed Deals, and the All-or-Nothing protocol that always results in this special kind of mixed deal where one agent does everything with some probability. The original loop example was an instance where hiding a letter might

be beneficial to an agent, when a Pure Deal was being used in the protocol. So we put an "L" there, to signify that there exist encounters where lying is beneficial. Similarly, our second example was an instance of this other box, where a phantom task was beneficial; it gets an "L" also. A "T" in a box means that honesty is the best policy. An agent's best strategy is always to tell the truth. So, for example, when an all-or-nothing protocol is being used in a subadditive TOD, no agent has any incentive to hide a task. The best strategy is to reveal all tasks.

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

Now, you'll also notice that there's a relationship between table entries, denoted by that white arrow. These are sort of 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 truth-telling is the best strategy in mixed deals, it is certainly also the best strategy in all-or-nothing deals, which are a subset.

6.2 Decoy Tasks

There's one more kind of lie that's worth looking at, and that's what we call decoy tasks. A decoy task is like a phantom, it's a fake task, but an agent can create it on demand, if necessary. So a postman might claim he has a letter to some particular node, and if required to hand it over when using a probabilistic deal, he quickly jots a note down, "Dear Resident: You may have already won the sweepstakes," and hands it over. So a simple penalty mechanism won't work, and in fact as the table shows [right side of Figure 11], decoy lies in subadditive domains can sometimes be beneficial for an agent even when all-or-nothing protocols are used. As an example, look at this graph, 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. So it sort of deceptively "locks itself into" its original path. Even with an all-or-nothing deal, agent 1 benefits from the deception, so this is an example of that table entry on the right. Again, notice the white arrows. Since lying can be shown to be beneficial using an all-or-nothing protocol, it must be sometimes beneficial when using mixed deals, which are a superset. The point is, having filled out one entry in the table, other entries may be implied automatically.

Subadditive TODs are an important class of domains, but there are other, more restrictive classifications that we can use to understand Task Oriented Domains.

7 Concave TODs

Concave Task Oriented Domains are a subset of subadditive TODs, where we have the following situation. 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. Then the cost Z adds to X, the subset, is greater than 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 that implies that it's also subadditive [see Figure 12].

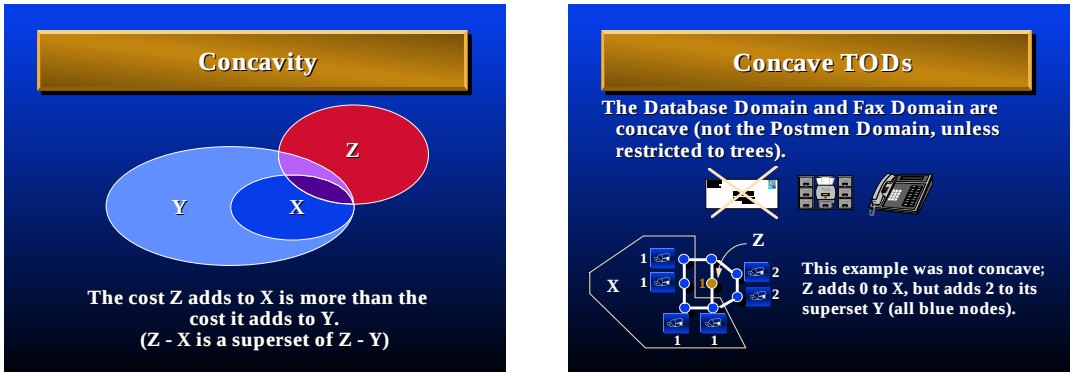

Figure 12: Concave Task Oriented Domains

To see what's going on, look at the diagram. X is a subset of Y. We come along with some arbitrary set of tasks Z. And it seems obvious, according to the diagram, that the cost Z adds to X will be more than the cost it adds to Y. So that illustrates the property of a domain that's concave.

Now, the diagram is actually a bit misleading, because it appears like such an obvious property, that it ought to hold of all reasonable Task Oriented Domains. But it doesn't [see the right side of Figure 12].

Of the three TODs we introduced originally, only the Database Domain and the Fax Domain are concave. Our trusty old friend, the general Postmen Domain, is *not* concave, unless graphs are restricted to trees. To see an example of a non-concave encounter in the general Postmen Domain, consider the example we gave for a beneficial decoy task. Agent 1 has to travel around these left nodes, let's call that X, and agent 2 has to travel around the right nodes. Let's call Y the set of all

dark gray nodes (i.e., excluding the middle node marked Z). So X, the left nodes, the ones marked with a "1," is a subset of Y, all the dark gray nodes. Agent 1 then lies with a decoy task to the node in the middle. Let's take that decoy task as set Z. Now the amount of work that Z, that middle node, adds to the set X, is 0. The agent visits Z on the way, no extra cost. But 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 of Y, then visit Z. So this example is not concave.

7.1 Three-Dimensional Incentive Compatible Mechanism Table

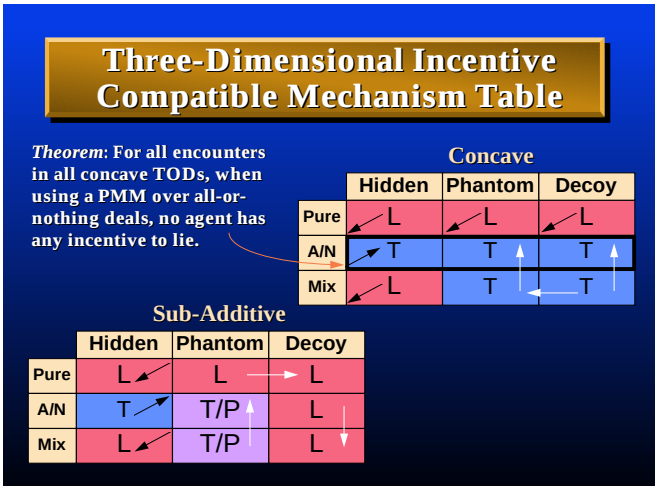

Figure 13: The Enlarged Strategy/Protocol Table

If we return to the table that we've been setting up, we can examine an entire new set of possibilities [see Figure 13]. The table's now become three-dimensional, and you can see from the black arrows that there are also relationships among Concave and 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 when a Pure Deal is used. That's because a Concave Domain is always also 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, and in particular, we've proven a theorem that says that in all Concave TODs,

when using all-or-nothing deals, no agent has any incentive to hide tasks, nor to create phantom or decoy tasks. There's absolutely no incentive to lie. Now, that theorem is what allows us to put the middle row of T's into the Concave part of the table.

We can make an even more precise classification of Task Oriented Domains with the following definition.

8 Modular TODs

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: $c(X \cup Y) = c(X) + c(Y) - c(X \cap Y)$. That's because you don't want to count the tasks that appear in both sets twice. And any modular domain is also a concave domain, and in turn also a subadditive domain, of course.

Figure 14: Modular Task Oriented Domains

This diagram shows exactly what's going on, and here it's pretty clear [see 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, so that middle region isn't counted twice in the cost calculation.

Of the three original TODs that I introduced, only the Fax Domain is modular. The Database Domain is not modular, and the general Postmen Domain is also not modular, unless you restrict the graphs that the postmen visit to have a star topology, with the Post Office in the middle and all the other nodes connected to it like spokes on a wheel. Modular TODs are the most restrictive categorization that we've looked at, and the best behaved with regard to lying, but even here

hiding tasks can sometimes be beneficial if the agents are using general mixed deal protocols.

8.1 Three-Dimensional Incentive Compatible Mechanism Table

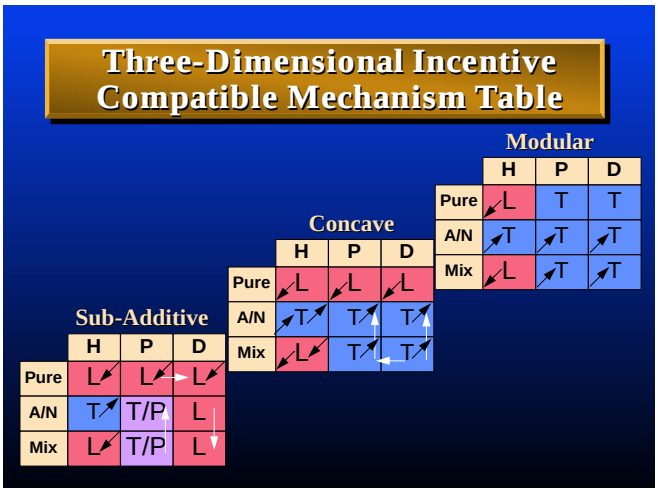

Figure 15: Categorizing Strategies Based on Domain and Protocol

Returning to our evolving table, we add another layer into the third dimension [see Figure 15]. This modular layer has more T's. You can see that in moving from subadditive to concave to modular we're increasing the percentage of T boxes, where telling the truth is always the best policy for an agent, but there are still some residual L's lurking in the table.

Designers who came together and 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, confident in the knowledge that none of the individual companies building the agents will have an incentive to conceal their tasks, nor create false ones. The best policy here is really just to tell the truth. Simple, efficient, and stable.

I have here a copy of an article that appeared in the Boston Globe a few months ago. It's called "A new dimension in deception" (written by Michael Schrage), and it talks about software agents that might choose to lie in order 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 in order to force a group of people to set a

meeting when *it* wants to have the meeting, and not at some other time. Now, the article's pretty good at laying out the scenario, but it missed the punchline, and that punchline is what this talk has been all about: sometimes, it's possible to design the rules of encounter so that lying is simply not in anyone's interest. If, for example, we have a Concave Domain, the protocol that the agents use might be set up to negotiate over all-or-nothing deals. And then, it's not that we're going to legislate that agents won't deceive—it just won't be rational for them to deceive.

9 Related Work

What I've spoken about today is really the tip of the iceberg. First of all, Task Oriented Domains themselves are only a small class of encounters between agents, and if you remember I presented two more general classes, State Oriented Domains and Worth Oriented Domains. We've carried out similar kinds of analysis of protocols in these more general types of domains, and the situation becomes more complicated. Lying, for example, is harder to prevent in those more general encounters, but we can still analyze properties of protocols, like efficiency and stability, and provide guidelines for how agent designers would 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 2 agents, general research into mechanism design (Ephrati, Kraus, Tennenholtz), research on other models of negotiation among agents (Kraus, Sycara, Durfee, Lesser, Gasser, Gmytrasiewicz), and research on other consensus mechanisms, such as voting techniques, explored in work that I've carried out with Ephrati, and economic models, such as those being examined by Wellman.

10 Conclusions

What have we been arguing? That 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 careful design of the negotiation protocol.

Second, we've pointed out that the interaction mechanism can and should be designed to ensure efficiency of the multi-agent system.

Third, to maintain efficiency over time of dynamic multi-agent systems, the rules must also be stable. It is not enough to figure out a strategy that has good properties, like efficiency—the agent designers have to feel that this is the strategy they should stick with, they shouldn't have any incentive to move to another strategy. So stability is a very important part of multi-agent systems.

Finally, we've done our analysis through 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. It's these kinds of tools that give us the leverage we need to design interaction environments for automated negotiation.

Other Reading

The work discussed in this talk can be found in the following articles and book: [23, 24, 26, 25, 27, 29, 20].

Related work can be found in the following articles: [16, 14, 15, 6, 7, 8, 10, 1, 3, 4, 5, 12, 11, 28, 18, 22].

For good introductory treatments of game theory and mechanism design, refer to the following books: [2, 17, 9, 13, 19, 21].

Acknowledgments

This research has been partially supported by the Leibniz Center for Research in Computer Science at the Hebrew University of Jerusalem, and by the Israeli Ministry of Science and Technology (Grant 032-8284). The authors wish 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.

References

[1] M. Avouris and Les Gasser. *Distributed Artificial Intelligence: Theory and Praxis*. Kluwer Academic Publishers, Boston, 1992.

[2] Ken Binmore. *Fun and Games, A Text on Game Theory*. D. C. Heath and Company, Lexington, Massachusetts, 1992.

[3] Susan E. Conry, Robert A. Meyer, and Victor R. Lesser. Multistage negotiation in distributed planning. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufmann Publishers, Inc., San Mateo, California, 1988.

[4] Keith S. Decker and Victor R. Lesser. An approach to analyzing the need for meta-level communication. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 360–366, Chambery, France, August 1993.

[5] Keith S. Decker and Victor R. Lesser. A one-shot dynamic coordination algorithm for distributed sensor networks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 210–216, Washington, DC, July 1993.

[6] E. Ephrati and J. S. Rosenschein. The Clarke Tax as a consensus mechanism among automated agents. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 173–178, Anaheim, California, July 1991.

[7] E. Ephrati and J. S. Rosenschein. Reaching agreement through partial revelation of preferences. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 229–233, Vienna, Austria, August 1992.

[8] E. Ephrati and J. S. Rosenschein. Distributed consensus mechanisms for self-interested heterogeneous agents. In *First International Conference on Intelligent and Cooperative Information Systems*, pages 71–79, Rotterdam, May 1993.

[9] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, Cambridge, Massachusetts, 1992.

[10] L. Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence*, 47(1–3):107–138, 1991.

[11] Piotr J. Gmytrasiewicz, Edmund H. Durfee, and David K. Wehe. A decision-theoretic approach to coordinating multiagent interactions. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 62–68, Sydney, Australia, August 1991.

[12] Piotr J. Gmytrasiewicz, Edmund H. Durfee, and David K. Wehe. The utility of comunication in coordinating intelligent agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 166–172, July 1991.

[13] John C. Harsanyi. *Rational Behavior and Bargaining Equilibrium in Games and Social Situations*. Cambridge University Press, Cambridge, 1977.

[14] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent negotiation under time constraints. Computer Science Technical Report Series CS-TR-2975, University of Maryland, College Park, Maryland, October 1992.

[15] Sarit Kraus. Agents contracting tasks in non-collaborative environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 243–248, 1993.

[16] Sarit Kraus and Jonathan Wilkenfeld. Negotiations over time in a multi-agent environment: Preliminary report. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 56–61, Sydney, August 1991.

[17] R. Duncan Luce and Howard Raiffa. *Games and Decisions*. John Wiley & Sons, Inc., New York, 1957.

[18] Y. Moses and M. Tennenholtz. On cooperation in a multi-entity model. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 918–923, Detroit, Michigan, August 1989.

[19] M. J. Osborne and A. Rubinstein. *Bargaining and Markets*. Academic Press Inc., San Diego, California, 1990.

[20] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, Cambridge, Massachusetts, 1994. To appear.

[21] Alvin E. Roth. *Axiomatic Models of Bargaining*. Springer-Verlag, Berlin, 1979.

[22] Y. Shoham and M. Tennenholtz. On the synthesis of usful social laws for artificial agent societies. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 276–281, San Jose, California, July 1992.

[23] G. Zlotkin and J. S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 912–917, Detroit, Michigan, August 1989.

[24] G. Zlotkin and J. S. Rosenschein. Negotiation and conflict resolution in non-cooperative domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 100–105, Boston, Massachusetts, August 1990.

[25] G. Zlotkin and J. S. Rosenschein. Cooperation and conflict resolution via negotiation among autonomous agents in noncooperative domains. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1317–1324, December 1991.

[26] G. Zlotkin and J. S. Rosenschein. Incomplete information and deception in multi-agent negotiation. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 225–231, Sydney, Australia, August 1991.

[27] G. Zlotkin and J. S. Rosenschein. A domain theory for task oriented negotiation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 416–422, Chambery, France, August 1993.

[28] G. Zlotkin and J. S. Rosenschein. The extent of cooperation in state-oriented domains: Negotiation among tidy agents. *Computers and Artificial Intelligence*, 12(2):105–122, 1993.

[29] G. Zlotkin and J. S. Rosenschein. Negotiation with incomplete information about worth: Strict versus tolerant mechanisms. In *Proceedings of the International Conference on Intelligent and Cooperative Information Systems*, pages 175–184, Rotterdam, May 1993.