Formalizing and Achieving Multiparty Agreements via Commitments

Feng Wan
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA
fwpub-ncsu@yahoo.com

Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7535, USA
singh@ncsu.edu

ABSTRACT

Multiparty agreements often arise in a multiagent system where autonomous agents interact with each other to achieve a global goal. Multiparty agreements are traditionally represented by messaging protocols or event-condition-action rule sets in which agents exchange messages in a predefined sequence to ensure both global and local consistency. However, these models do not readily incorporate agents' autonomy and heterogeneity, which limits their ability to help build a flexible open system. Commitments have been studied for modelling various agent interactions. They have also been used as the key elements for formulating multiparty agreements and centralized approaches for resolving potential conflicts. This paper extends the above results by refining the formalizations and the existing protocols and proposing a decentralized protocol which is more efficient in resolving conflicts. It also introduces the concept of protocol safety, which ensures that agents not only interact efficiently but also correctly. This approach is geared toward constructing business processes where agents are mutually constraints in a manner that preserves their autonomy and heterogeneity.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*; D.1.0 [**Software Engineering**]: Programming Techniques—*general*; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Design, Verification

Keywords

Agents, Commitments, Agreements, Multiagent Systems

AAMAS'05, July 25-29, 2005, Utrecht, Netherlands.

1. INTRODUCTION

In a multiagent system, agents autonomously decide on whether to perform a particular action. When agents coordinate with each other to achieve a global task, they need to first create a multiparty agreement, which would satisfy global goals as well as the agents' individual constraints. Multiparty agreements in agent communities are more subtle than in other software systems where agreements are represented by fixed protocols and each party follows a predetermined execution sequence.

Researchers have studied multiparty agreements from several perspectives, including developing standard languages and protocols such as FIPA [4], implementing domain-specific protocols such as auction protocols, and developing methodologies for building agents such as AUML [7]. These approaches tend to define interaction frameworks that limit agents' choices. There is a rich literature on how agents form teams and negotiate on global execution plans, e.g., by Tambe and colleagues [11], and on how agents with different attitudes communicate facts with each other to make the right decisions, e.g., by Parson *et al.* [8].

Consider an example where a buyer wants to buy some goods from a seller. The buyer may require the seller to ship the goods before he would pay. The seller may require the buyer to pay before he would ship. Various approaches exist to resolve such situations in the real world, e.g., the buyer can make an advance deposit, the buyer and seller can use an escrow service to ensure successful execution of all steps, and so on. In essence, the different approaches correspond to different protocols that the agents must follow in order to bring an agreement to a fruitful completion. This leads to two questions of interest. What is the principled basis for such protocols? What semantics must be incorporated into the agreements?

Our proposed approach begins from a representation of multiparty agreements based on commitments. Commitments represent the obligations made between pairs of agents and are used to model interactions in a multiagent system. Commitments help agents express promises and monitor each other's compliance without regard to internal implementational details.

This paper uses commitments as the basic elements to form multiparty agreements. For example, one of the agents may strengthen its conditional commitment, replace it by an unconditional commitment, or even perform the desired action. Doing so may induce the other agents to perform their desired actions resulting in overall progress. For example,

an agent A who is apparently deadlocked with agent B may help their collective deal progress by making an utterance such as "OK, I will ship if you promise to pay on receiving the goods." Then, B could say "I promise to pay if I receive the goods." Assuming sufficient trust, progress could be obtained.

More subtle such moves would be involved when the agreement structures were more complex, e.g., in terms of the number of participants involved and the richness of their relationships. The semantics of commitments provides a ready and rigorous basis by which the agents could decide their conversational moves and other agents could interpret these conversational moves with respect to the agreements at hand. Moreover, the semantics of commitments helps us specify particular protocols that apply in different settings, and to analyze the effectiveness and safety of such protocols.

2. COMMITMENTS

A commitment is an obligation from a debtor x to a creditor y about a particular condition p. A commitment has the following two basic forms.

- *Unconditional commitment* $C(x, y, p)$. A commitment whose condition p will be brought about unconditionally by the debtor x. The commitment behaves as a directed obligation from the debtor x to the creditor y; the creditor has special functions, including being able to release the debtor (we ignore this aspect here). For example, $C(\text{buyer}, \text{seller}, \text{pay})$ denotes that the buyer promises to pay the seller.
- *Conditional commitment* $C(x, y, e \rightarrow p)$. A commitment whose condition p will be brought about if the precondition e becomes true. For example, $C(\text{buyer}, \text{seller}, \text{ship} \rightarrow \text{pay})$ denotes that the buyer promises to pay the seller if the latter ships the goods to him.

The precondition e of a conditional commitment can be expressed as a compound predicate which could embed other commitments as needed. This allows us to define commitments recursively and to specify complex obligation dependencies. For example, we could define a commitment $C(\text{buyer}, \text{seller}, C(\text{seller}, \text{buyer}, \text{ship}) \rightarrow \text{pay})$ saying that the buyer promise to pay the seller if the latter promise to ship the goods. This commitment differs from the above conditional commitment in that the payment only depends on the promise of shipping and may be performed before the actual shipping.

We require that for any inner commitment appearing on precondition e, the creditor must be the debtor of the immediate outside commitment. In other words, we require that if a commitment has form $C(x, y, \ldots C(w, v, q) \ldots \rightarrow p)$, then $x = v$. The motivation is that the debtor x's commitment is conditioned on another party doing something for x.

Commitments support several operations that combine to capture mutual and multiparty scenarios [10]. For the sake of simplicity, this paper is limited to four main operations. The four operations, *create*, *update*, *discharge*, and *cancel*, drive the lifecycle of commitments. A commitment is initially created when an agent makes a promise to another agent. If the commitment has been fulfilled, e.g., the conditions have become true, then the commitment is discharged. However, before the commitment is discharged, the agents involved can possibly update the commitment. The update operation gives flexibility in manipulating agents' context to react to any potential requirement changes or exceptions. Agents can also cancel their commitments, e.g., to accommodate exceptions. However, to cancel a commitment, agents usually face penalties that compensate for whatever inconsistencies that they may have introduced. Figure 1 shows the state diagram of the commitment lifecycle.

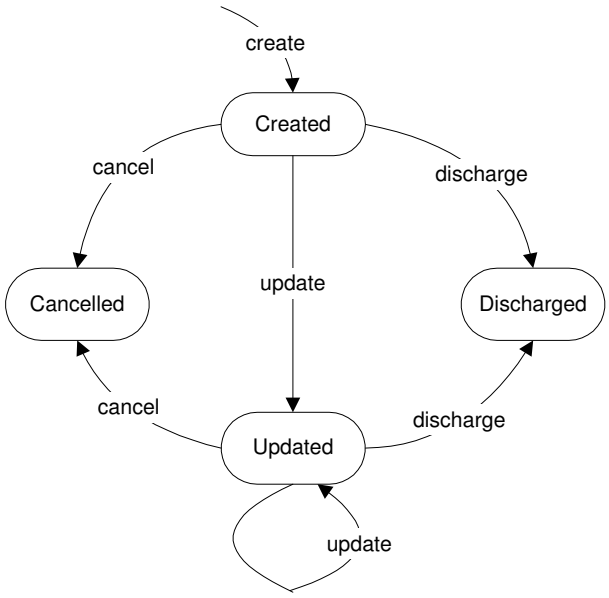

Figure 1: Commitment lifecycle

3. REPRESENTING MULTIPARTY AGREEMENTS

In a multiagent system, an important class of interactions involves agents making and fulfilling commitments to each other. The commitments themselves form protocol dependencies and coordination requirements among the agents concerned. Therefore, we define a multiparty agreement as follows:

DEFINITION 1. *A multiparty agreement A is given by a set of commitments $\{\mathsf{C}_1, \mathsf{C}_2, \cdots, \mathsf{C}_n\}$ where C_i is either an unconditional commitment or a conditional commitment.*

An example agreement $A = \{\mathsf{C}_1, \mathsf{C}_2\}$ is shown below. In this example, the buyer conditionally promises the seller if the latter ships the goods, then he will pay. The seller promises to ship the goods unconditionally. The outcome of this agreement is that the buyer pays to the seller after the latter ships the goods to him.

$$\begin{aligned} \mathsf{C}_1 &= \mathsf{C}(\text{buyer}, \text{seller}, \text{ShipGoods} \rightarrow \text{Pay}) \\ \mathsf{C}_2 &= \mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods}) \end{aligned}$$

In another example the buyer promises the seller that if the latter promises to ship a goods, then he will pay the seller. The seller promises to ship the goods. The outcome of this

agreement is the payment and the shipment can happen in arbitrary temporal orders.

$$\mathsf{C}_1 = \mathsf{C}(\text{buyer}, \text{seller}, \mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods}) \rightarrow \text{Pay})$$
$$\mathsf{C}_2 = \mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods})$$

Section 4 shows another agreement with deadlocking dependencies and describes how to resolve such cases.

3.1 Agreement Derivation Rules

Here we give a set of rules to reduce an agreement (or a commitment set) to a set of conditions that all the agents would eventually bring about. The purpose of the reductions is to show how the interactions progress given a commitment set. This also gives us a way to detect potentially deadlocking agreements. For simplicity, we do not consider the *cancel* and *update* operations, which usually digress from normal executions and do not help in detecting deadlocks introduced by the original commitment set.

$$E_1 : \mathsf{C}_i \in A \Rightarrow Create(\mathsf{C}_i)$$
$$E_2 : Create(\mathsf{C}(x,y,p)) \dashrightarrow Discharge(\mathsf{C}(x,y,p))$$
$$E_3 : Discharge(\mathsf{C}(x,y,p)) \Rightarrow p$$
$$E_4 : e \wedge Create(\mathsf{C}(x,y,e \rightarrow p)) \Rightarrow Create(\mathsf{C}(x,y,p))$$

E_1 shows that any commitment inserted into agreement A enters the Creation state immediately. We use $\Rightarrow$ to denote immediacy. E_2 shows that the creation of an unconditional commitment will eventually reduce to a discharge of the commitment. We use $\dashrightarrow$ to denote eventuality. This rule expresses the idea that when an agent makes an unconditional obligation to another, he must fulfill it eventually. E_3 shows that a discharge of an unconditional commitment makes the condition true immediately. Although rules E_2 and E_3 support concluding that the creation of an unconditional commitment can directly bring about the condition, the Discharge operation in between allows a designer to map concrete business activities corresponding to this operation. E_4 shows that after a conditional commitment is created, if its preconditions are satisfied, it will be converted to its corresponding unconditional commitment by removing the preconditions.

The following shows a derivation on the second example agreement above.

$$\begin{aligned}
A &\xrightarrow{E_1} Create(\mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods})) \wedge \\
&\quad Create(\mathsf{C}(\text{buyer}, \text{seller}, \\
&\qquad \mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods}) \rightarrow \text{Pay})) \\
&\xrightarrow{E_4} Create(\mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods})) \wedge \\
&\quad Create(\mathsf{C}(\text{buyer}, \text{seller}, \text{Pay})) \\
&\xrightarrow{E_2} Create(\mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods})) \wedge \\
&\quad Discharge(\mathsf{C}(\text{buyer}, \text{seller}, \text{Pay})) \\
&\xrightarrow{E_3} Create(\mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods})) \wedge \text{Pay} \\
&\xrightarrow{E_2} Discharge(\mathsf{C}(\text{seller}, \text{buyer}, \text{ShipGoods})) \wedge \text{Pay} \\
&\xrightarrow{E_3} \text{ShipGoods} \wedge \text{Pay}
\end{aligned}$$

The derivation generates the action sequence {Pay, ShipGoods} in which Pay happens before ShipGoods. However, it can also generate the sequence {ShipGoods, Pay}. Section 4 formally shows that this agreement is satisfiable.

3.2 Generating Agreement Diagram

An agreement diagram (AD) denotes the set of constraint dependencies among interacting agents. An AD is derived from commitment sets and changes dynamically as agents manipulate their commitments during business engagements. The construction of an AD not only helps monitoring runtime agent behaviors but also detecting any agreement deadlocks (see Section 4). Here we present Algorithm 1 that derives an agreement diagram from a given commitment set.

The algorithm creates nodes for debtor and creditor agents, and edges for conditions. It connects these nodes following the condition dependencies. If one of the preconditions of commitment C_1 is brought about by another outside commitment C_2, then we call it a hard dependency, since C_2 must be discharged before C_1 can be discharged. However, if C_2 is an inner commitment of C_1, then we call it soft dependency, since C_1 is discharged based on the promise of C_2, but not necessarily before the fulfillment of C_2. We use dotted lines to denote soft dependencies and solid lines for hard dependencies. The differentiation of the two types of dependencies enables us to capture important subtleties in agent interaction and thus generate flexible protocol executions.

The following is a list of some sample commitment sets and their corresponding diagrams (see Figure 2).

$$\begin{aligned}
A_1 &= \{C(x,y,p)\} \\
A_2 &= \{C(x,y,e \rightarrow p)\} \\
A_3 &= \{C(z,x,e),\ C(x,y,e \rightarrow p)\} \\
A_4 &= \{C(x,y,C(z,x,e) \rightarrow p)\} \\
A_5 &= \{C(w,x,p_3), \\
&\quad C(x,y,p_1 \wedge C(z,x,p_2) \wedge p_3 \rightarrow p)\} \\
A_6 &= \{C(w,x,p_1), \\
&\quad C(x,y,(p_1 \wedge p_2) \vee C(z,x,p_3) \rightarrow p)\}
\end{aligned}$$

4. BUILDING SATISFIABLE AGREEMENTS

In a multiagent system, each individual agent can have its local constraints. The agents' commitments not only specify their protocols, but also factor in their local constraints (which in essence limit what the agents can promise others). However, since the agents are autonomous, the constraints of different agents may form cyclic dependencies. For example, a buyer may want the seller to ship the goods first before he makes the payment, but the seller may want the buyer to pay first before he ships the goods. The two commitments can be expressed as below.

$$\mathsf{C}_1 = \mathsf{C}(buyer, seller, ShipGoods \rightarrow Pay)$$
$$\mathsf{C}_2 = \mathsf{C}(seller, buyer, Pay \rightarrow ShipGoods)$$

By executing Algorithm 1, we obtain an agreement diagram, as shown in Figure 3. Apparently, neither party will proceed because of the deadlocking dependencies. Our approach detects these cyclic constraint dependencies. We propose several protocols to resolve them and produce a satisfiable commitment set. First let us define what a satisfiable multiparty agreement is.

DEFINITION 2. *A multiparty agreement is satisfiable if and only if for any $\mathsf{C}(x_i, y_i, p_i)$, p_i will eventually become true; or for any $\mathsf{C}(x_i, y_i, e_i \rightarrow p_i)$, p_i will eventually become true if e_i becomes true.*

1 We first identify commitment entities based on the following rules
begin
(a) Each $C(...)$ is a commitment entity;
(b) $C_1(x_1, y_1, W_1)$ and $C_2(x_2, y_2, W_2)$ are the same if $x_1 = x_2$, $y_1 = y_2$, and $W_1 = W_2$;
(c) If $\exists\ C_1(x, y_1, W_1)$ and $C_2(x, y_2, W_2)$ where $y_1 \neq y_2$ or W_1 and W_2 have different conditions (regardless of preconditions), then rename them to $C_1(x_1, y_1, W_1)$ and $C_2(x_2, y_2, W_2)$
end
2 For each unique agent in the list of commitment entities, draw a node labeled with the agent name.
3 For each commitment $C(x, y, p)$ or $C(x, y, e \rightarrow p)$, draw an edge labeled with p from agent x's node. If the commitment does not form any hard dependencies, then use a dotted line, otherwise, a solid line.
4 **for** *each commitment* $C(x, y, e \rightarrow p)$ **do**
5 Convert e to DNF (Disjunctive Normal Form);
switch e **do**
case $P_1 \wedge P_2 \wedge \cdots \wedge P_n$
6 Use an AND connector to connect edges p_1, p_2, ..., p_n to node x;
case $T_1 \vee T_2 \vee \cdots \vee T_n$
7 Use an OR connector to connect edge T_1, T_2, ..., T_n to node x;
8 For each T_i, if it has more than one atomic proposition, then use an AND connector and perform step 5;
9 For each edge p created above, if there exists an agent node generated from step 3 with edge p, then merge the two edges;
10 If a node from step 3 has not been picked up by step 9, then attach its original creditor node as the end node (If the creditor is one of the debtors x in step 4, then create a new node with a different name).

Algorithm 1: Building an agreement diagram

4.1 Detecting Agreement Deadlocks

We have an intuition that, if the agreement diagram derived from a commitment set has a solid cycle (formed by all solid lines), then it forms a cyclic hard dependency chain. This means that no condition would be brought about. In other words, the commitment set is not satisfiable. Here we introduce two theorems to show what kinds of agreement diagram are not satisfiable.

THEOREM 1. *If there exists a solid cycle in a given agreement diagram and any node on the cycle only has AND preconditions, then the agreement is not satisfiable.*

PROOF. *1. First we define $p(t)$ as true or false at a time t*

2. Assume the cycle is $x_1 \xrightarrow{p_2} x_2 \xrightarrow{p_3} \ldots x_n \xrightarrow{p_1} x_1$, where p_i is one of the AND preconditions of each node x_i, then based on steps (4) through (8) in Algorithm 1,

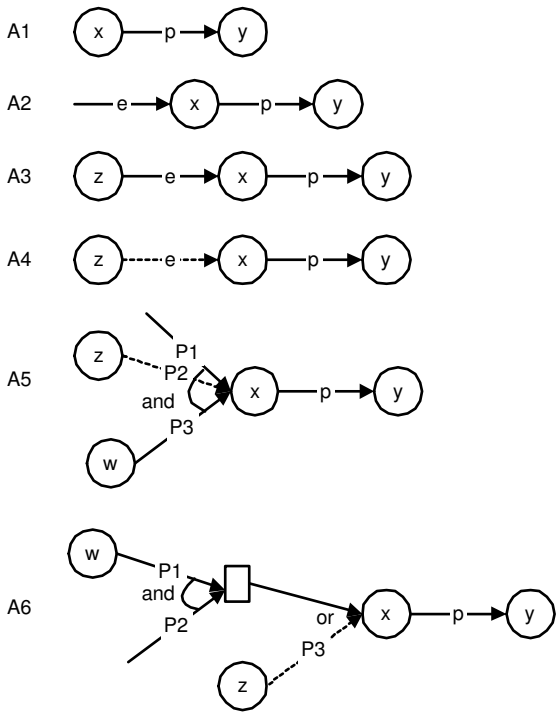

Figure 2: Agreement diagram examples

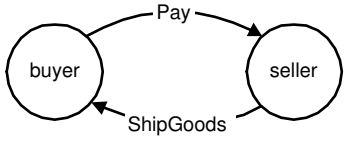

Figure 3: A deadlock agreement

we obtain a commitment set $\{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_n\}$ where

$$\begin{aligned} \mathsf{C}_1 &= \mathsf{C}(x_1, x_2, p_1 \wedge W_1 \rightarrow p_2) \\ \mathsf{C}_2 &= \mathsf{C}(x_2, x_3, p_2 \wedge W_2 \rightarrow p_3) \\ &\ldots \\ \mathsf{C}_n &= \mathsf{C}(x_n, x_1, p_n \wedge W_n \rightarrow p_1) \end{aligned}$$

W_i are the rest of the AND preconditions of node x_i

3. Assume C_1 is eventually discharged. Then we can find a time t_m, such that $p_2(t_m)$ (i.e., p_2 holds at t_m);

4. Based on derivation rule E_4, we can find a time t_1, $t_1 < t_m \wedge p_1(t_1)$, and $\forall t,\ t \leq t_1 \wedge \neg p_2(t)$.

5. For the same reason, we can find a time t_2, $t_2 < t_1 \wedge p_n(t_2)$ and $\forall t,\ t \leq t_2 \wedge \neg p_1(t)$.

6. Repeating step (5), we can find a time t_n, $t_n < t_{n-1} \wedge p_2(t_n)$ and $\forall t,\ t \leq t_n \wedge \neg p_3(t)$.

7. From (4) through (6), we have $t_n < t_1 \wedge p_2(t_n)$. This result conflicts with "$\forall t,\ t \leq t_1 \wedge \neg p_2(t)$" in step (4) above. Therefore C_1 cannot be discharged and the same conclusion applies to C_2, ..., C_n.

8. *Since no commitment on the cycle can be discharged, the multiparty agreement is not satisfiable.*

□

To consider cycles that involve nodes with OR preconditions, we have the following theorem.

THEOREM 2. *If there exists a set of connected solid cycles in a given agreement diagram, and for any node that has OR preconditions, each edge of the OR preconditions is also on one of the cycles, then the agreement is not satisfiable.*

PROOF. *1. Assume that a condition p_1 on a cycle becomes true. Then the cycle must have at least one node with OR preconditions. Otherwise, it conflicts with Theorem 1.*

2. *In all the nodes with OR preconditions, there must exist a node whose precondition becomes true. Otherwise, no condition will be satisfied on the cycle, which conflicts with the assumption in (1) above.*

3. *Assume the satisfied precondition is p_2. Based on the premise, p_2 is also on one of the cycles.*

4. *If p_2 is on the same cycle that p_1 belongs to, by applying steps (2) through (7) in Theorem 1, we can prove that neither of them can be satisfied.*

5. *If p_2 is on a different cycle and the cycle has other OR nodes, then repeat steps (2) and (3) above.*

6. *If p_2 is on a different cycle and the cycle has no other OR nodes, the proof is same as for step (4) above, so p_2 cannot be satisfied.*

7. *From the conflicts generated from steps (4) or (6) above, we conclude that p_1 cannot be satisfied. Therefore, the agreement is not satisfiable.*

□

There could also exist dotted cycles (formed by all dotted lines). That is, even promises themselves may have deadlock dependencies. For example, the two commitments below contain more subtle constraints. The analysis of this scenario is not within the scope of this paper.

$$\mathsf{C}_1 = \mathsf{C}(x, y, \mathsf{C}(y, x, q) \rightarrow p)$$
$$\mathsf{C}_2 = \mathsf{C}(y, x, \mathsf{C}(x, y, p) \rightarrow q)$$

4.2 Resolving Agreement Deadlocks

Deadlocking constraints imposed on a group of agents do not mean that these agents cannot engage activities at all. Autonomous agents can negotiate to serve their interests. To break these deadlocking dependencies, all the agents may choose to commit what they promise to do for the others regardless of what constraints they impose on the others, or some of the agents may concede to satisfy the others first before their own constraints are satisfied. All these approaches lead to a variety of protocols for forming satisfiable multiparty agreements. For the sake of simplicity, we only show how to resolve a solid cycle that only has nodes with AND preconditions. We use the following three commitments as the example agreement:

$$\mathsf{C}(x, y, p \rightarrow q), \mathsf{C}(y, z, q \rightarrow r), \mathsf{C}(z, x, r \rightarrow p)$$

By applying Algorithm 1, we can tell that its agreement diagram is a deadlocking cycle. The following protocols describe different approaches to resolve this deadlock.

4.2.1 Two-Phase (2PC) Protocol

The two-phase (2PC) protocol is inspired by a similar protocol, called *two-phase commit*, which is widely used in distributed database systems where task executors either all commit or all abort their transactions to ensure task atomicity and preserve system consistency [5]. For the present purpose, we apply this protocol to agents who have deadlocking constraints, which prevents them from discharging any commitment to each other. The goal of the protocol is to make sure that all the involved agents commit first before their preconditions are met. The following shows the steps of the 2PC protocol.

```
1 A coordinator tells all the agents that are involved in a
  solid cycle that a 2PC protocol is started;
2 Each agent sends yes or no to indicate whether it wants
  to unconditionally discharge its commitments;
  if all the agents answer yes then
3 |  The coordinator sends yes to all agents;
4 |  Each agent replaces its conditional commitment
  |  with a corresponding unconditional commitment by
  |  removing the preconditions;
  else
5 L  No solution can be found.
```

Algorithm 2: 2PC protocol

By executing the 2PC protocol, q, r, and p will be unconditionally performed by x, y, and z, respectively. Once these conditions become true, they would also satisfy each precondition in the above commitments. In terms of this aspect, the 2PC protocols essentially convert all the conditional commitments to their corresponding unconditional commitments provided all agents agree. Therefore, the above three commitments become

$$\mathsf{C}(x, y, q), \mathsf{C}(y, z, r), \mathsf{C}(z, x, p)$$

An assumption of the 2PC protocol is that, for any commitment $\mathsf{C}(x, y, p \rightarrow q)$, p is not required to happen before q, but must happen eventually. However, there may be other constraints which require that p happens before q can happen (Section 4.3 returns to a discussion of protocol safety). The 2PC protocol does not apply in such a case and we need other protocols to resolve such conflicts.

4.2.2 Unconditional Yield

If an agent is willing to convert its conditional commitment to an unconditional commitment, we say that this agent yields unconditionally. In the above example, agent x may promise y to perform q without being satisfied by p first. This usually happens when the debtor of p, which is z in this example, has developed enough credit with x, which makes the latter believe that p will be eventually performed by z, even after x's unilateral concession. This protocol differs from the 2PC protocol in that it is based on trust whereas 2PC is based on an unanimous agreement. Here we construct a protocol to convey x's intention and propagate it to other agents to make corresponding commitment changes.

1 A coordinator notifies all the agents that are involved in a solid cycle that an Unconditional Yield protocol has been initiated;
2 Each agent sends yes or no to indicate whether it is willing to unconditionally discharge its commitment;
if *at least one agent answers yes* **then**
3 pick the first agent (say agent x) who answers yes and forward its answer to all agents;
4 Agent x will replace its conditional commitment with a corresponding unconditional commitment by removing the preconditions.
else
5 No solution can be found.

Algorithm 3: Unconditional yield protocol

By executing the protocol on the above example, the three commitments are changed to

$$\mathsf{C}(x, y, q),\ \mathsf{C}(y, z, q \rightarrow r),\ \mathsf{C}(z, x, r \rightarrow p)$$

in which case agent x will commit q unconditionally to y (intuitively, based on its implicit belief that agent z will eventually commit p to it if r happens.)

4.2.3 Conditional Yield

This is a more complicated scenario in that the agent willing to make an unconditional commitment does not place enough trust on the other agents. It must conditionally rely upon other agents' promises to it before it can perform its action. Conditional yield usually involves two agents. For example, let them be agent x and z. Agent x will bring about q if agent z promises x to bring about p. Agent z may make the promise, but may not fulfill it until its precondition r is satisfied. However, in the meantime, agent x can bring about q because of z's promise. This protocol is described in Algorithm 4.

1 A coordinator notifies all the agents involved in a solid cycle that a Conditional Yield protocol has been started;
2 Each agent sends yes or no to indicate whether it is willing to conditionally discharge its commitments;
for *each agent x who answers yes* **do**
3 Let z be the agent that x depends on;
4 Contact z to see if it can make a promise to x;
if *z answers yes* **then**
5 The coordinator picks agent x and z, and notifies the result to all the agents;
6 Agent z converts its conditional commitment to an unconditional commitment;
7 Agent x converts its hard dependency commitment to a soft dependency commitment, which is based on z's promise;
8 Stop the protocol;
9 No solution can be found.

Algorithm 4: Conditional yield protocol

By executing the protocol on the above example, the three commitments are changed to

$$\mathsf{C}(x, y, \mathsf{C}(z, x, p) \rightarrow q),\ \mathsf{C}(y, z, q \rightarrow r),\ \mathsf{C}(z, x, p)$$

Note that, although agent z creates an unconditional commitment, it may wait for r to happen before bringing about p.

4.2.4 Decentralized Protocol

The above three protocols are centralized and they require a coordinator. In such protocols, the agents are forced to wait for the voting results from the coordinator before proceeding. This would potentially delay the commitment fulfillment for the agents who are willing to yield, which in turn reduces the protocol efficiency. Also, a centralized coordinator would create a single point of failure and affect robustness. To overcome the above disadvantages, we devised a decentralized protocol, which allows an agent to construct partial agreement diagrams locally. If a deadlocking cycle is detected by the agent, it can yield at its own will without coordinating with other agents. This protocol yields more agent autonomy and generates more flexible executions. Due to the space limitation, we only describe the skeleton of the protocol in Algorithm 5.

for *each commitment made to others* **do**
if *the commitment has preconditions* **then**
1 Find the agents who can satisfy the preconditions;
2 Send a message to each such agent informing them that this agent depends on those preconditions;
3 On receiving a message informing of a dependency,
4 Place the sending agent in the dependency list of the corresponding conditions;
if *this agent relies upon other agents to bring about the conditions* **then**
5 Propagate the sending agent's name (along with own name) to those agents;
if *received dependency request refers to this agent* **then**
6 A deadlock is detected;
7 Apply local policies to decide whether to yield or not, or whether yield unconditionally or conditionally.

Algorithm 5: Decentralized protocol (one copy for each agent)

Here we give an example to illustrate the algorithm. If there exists a solid cycle, assume it represents the following commitment set.

$$\begin{aligned}
\mathsf{C}_1 &= \mathsf{C}(x_2, x_1, p_2 \wedge W_2 \rightarrow p_1) \\
\mathsf{C}_2 &= \mathsf{C}(x_3, x_2, p_3 \wedge W_3 \rightarrow p_2) \\
&\ldots \\
\mathsf{C}_{n-1} &= \mathsf{C}(x_n, x_{n-1}, p_n \wedge W_n \rightarrow p_{n-1}) \\
\mathsf{C}_n &= \mathsf{C}(x_1, x_n, p_1 \wedge W_1 \rightarrow p_n)
\end{aligned}$$

Figure 4 shows a partial protocol execution that starts from agent x_2, in which x_2 sends message to x_3 saying it

depends on p_2 to fulfill its commitments to others; x_3 then puts x_2 in p_2's dependency list. Since x_3 relies on p_3 to bring about p_2, it sends a message to x_4 saying x_2 should be in p_3's dependency list. The propagation eventually ends at x_2, in which p_1's dependency list is filled with $\{x_2, \ldots, x_n, x_{n-1}\}$. When x_2 finds that itself is in the dependency list of p_1, the condition that it is expected to bring about, it can conclude that it is involved in a dependency deadlock.

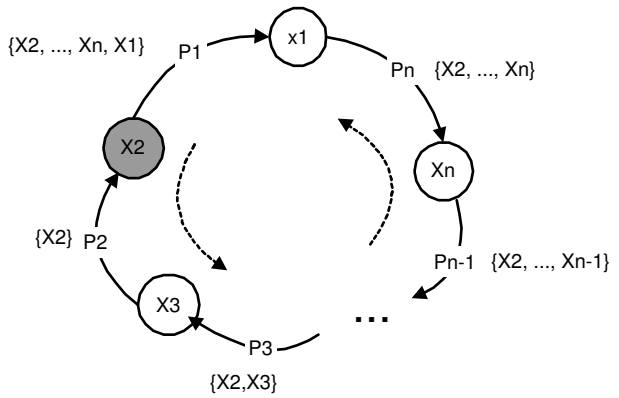

Figure 4: A decentralized protocol

4.3 Protocol Safety

Commitment-based protocols differ from other protocols in that promises can be used as dependency constraints and these constraints can be manipulated on the fly. This characterization is extremely useful in a multiagent system where agents interact autonomously and system behaviors can only be validated by the commitments that agents make to each other. However, when considering the flexibility of protocol execution, we also need to take the correctness of execution into account. In other words, a flexibly generated execution sequence should not violate global business policies. Since the policies may or may not fall into the local view of agents, it is possible that agents behave incorrectly. Here we give two definitions to show what safe executions are.

DEFINITION 3. *A* global policy *is defined as* $GP = \bigcup_i^n e_i \rightarrow p_i$ *where both* e_i *and* p_i *are events and* p_i *can happen only when* e_i *happens.*

DEFINITION 4. *An agreement is* protocol safe *(with respect to a given policy GP) if for any of its derivation sequence* $\{p_1, p_2, ..., p_n\}$, $\forall p_i, p_j$ *where* p_i *happens before* p_j, $p_j \rightarrow p_i \notin GP$.

Protocol safety ensures that the manipulation of commitments and the resulting execution sequences should comply with a global policy. This requires all the protocols presented above should take safety into account. The validation is relatively straightforward for centralized protocols, since the coordinator has the global view and can make right decisions. However for decentralized protocols, we may need to take extra steps to pass global static constraints to each agent. The research on an efficient solution is still undergoing.

5. DISCUSSION

Argumentation-based dialogue theory has been used for building multiagent systems. In recent work, Parsons *et al.* [8] have studied a set of agent attitudes by categorizing agents' truth telling behavior as *confident*, *careful*, and *thoughtful*. These attitudes are further used to define a set of agent dialogues such as *Information-Seeking*, *Inquiry*, and *Persuasion*. The motivation behind this work is to enable agents with different level of knowledge and credibility to communicate facts with others and to make right decisions. As a counterpart of their work, our approach enables agents to convey constraints and promises instead of their knowledge so that our agents care more about whether a promise is fulfilled instead of whether a fact is true.

Research of constraint satisfaction problems (CSP) in distributed systems, e.g.,by Liu *et al.* [6], has shown much promise. The models behind CSP are computational since the entire multiagent environment and individual agents are represented by variables, formulas, and constraints, which are made ready to compute based on mathematical rules. However, these models in most cases require homogeneous agents who sense and act in exactly the same manner, so they are not suitable for a real business-to-business world where parties are heterogeneous and loosely coupled. Our approach trades off complexity in the agent models with flexibility of the agents' behaviors. The outcome of our problem-solving is a set of satisfiable commitments.

Economou *et al.* study obligations among agents [1]. In their theory, if deontic states are entered, then the commitments have to be fulfilled. This is similar to our approach. A key difference is that we respect agent autonomy and allow agents to manipulate commitments on the fly. Doing so not only produces optimal executions but can also help avoid deadlocks as discussed above. In this manner, our approach complements deontic theory. It reconciles obligations and autonomy to enable modeling and enactment of practical agent applications.

Commitments are widely recognized as the key elements to capture the interactions among agents; this goes back to work by Singh [9] and Castelfranchi [3]. The essence of commitments is to create a structure to specify the obligations that each agent makes to others. By tracking the lifecycles of these commitments, one can monitor agents' external behaviors and detect any violation and system inconsistency without knowing the agents' internal structure [12]. Current research emphasizes how commitments are fulfilled or whether they are violated after they have been created. We are not aware of work that directly addresses how commitments can coexist from the very beginning. This is the aspect studied by this paper. In other words, we develop a means to detect deadlocking commitments and resolve them to ensure the progress of agent interactions.

Business-to-business applications are the main motivation for our approach. By looking at the existing approaches that model business processes, such as Agent UML (AUML) [7] and Business Process Execution Language (BPEL) [2], we can observe that they all impose inflexible protocols in which actions must happen in a predefined order. Agents get no opportunity to express whether they want to perform tasks differently than explicitly specified. Commitments enable agents to tell each other what they are going to do and what conditions that have to be satisfied to make it happen. This not only enables autonomous agents to perform their

tasks based on their local preferences, but can also avoid potentially deadlocking interactions caused by improper assumptions made by each agent. Our approach incorporates commitments into protocols and thus enable an open and flexible environment for various parties doing business.

This paper introduced commitments as the key elements for formulating a multiparty agreement from which we can derive agent interactions, detect potential commitment deadlocks, and resolve these deadlocks. The approach is a natural extension of our recent work [13] on commitment causal relations where the interaction among business agents are modelled via commitments and causal relations among commitments while agent autonomy and heterogeneity are still preserved.

Key future directions include the following. One, create a complete list of agreement patterns to cover most of today's e-business interactions. Two, extend the decentralized protocol described above to give more negotiation capabilities to agents so as to maximize their autonomy. Three, more deeply study the role of trust in the above kinds of protocols.

6. REFERENCES

[1] G. Economou, M. Tsvetovat, K. Sycara, and M. Paolucci. Implicit commitments through protocol-level semantics. In *Proc. 2nd Workshop on Norms and Institutions in MAS*, 2001.

[2] BPEL. Business process execution language for web services, version 1.1, May 2003.

[3] C. Castelfranchi. Commitments: From individual intentions to groups and organizations. In *ICMAS*, pages 41–48, 1995.

[4] FIPA. FIPA interaction protocol specifications, 2003.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo, 1993.

[6] J. Liu, H. Jing, and Y. Tang. Multi-agent oriented constraint satisfaction. *Artificial Intelligence*, 136:101–144, 2002.

[7] J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In *AOIS 2000*.

[8] S. Parsons, M. Wooldridge, and L. Amgoud. Properties and complexity of formal inter-agent dialogues. *Logic and Computation*, 13(3):347–376, 2003.

[9] M. P. Singh. Social and psychological commitments in multiagent systems. *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, 104–106, 1991.

[10] M. P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 7:97–113, 1999.

[11] M. Tambe. Agent architectures for flexible, practical teamwork. In *AAAI*, pages 22–28, 1997.

[12] M. Venkatraman and M. P. Singh. Verifying compliance with commitment protocols. *Autonomous Agents & Multi-Agent Systems*, 2(3):217–236, 1999.

[13] F. Wan and M. P. Singh. Commitment and causality in multiagent design. In *AAMAS 2003*.