Concurrent Multiple-Issue Negotiation for Internet-Based Services

Negotiation is a technique for reaching a mutually beneficial agreement among autonomous entities. In an Internet-based services context, multiple entities will likely be negotiating simultaneously. The concurrent negotiation protocol extends existing negotiation protocols, letting both service requestors and service providers manage several negotiation processes in parallel. Colored Petri nets, which have greater expressive power than finite state machines and offer support for concurrency, represent the negotiation protocol and facilitate the analysis of desirable properties.

Jiangbo Dang and Michael N. Huhns
University of South Carolina

Negotiation is a process by which autonomous entities communicate and compromise to reach agreement on matters of mutual interest while maximizing their individual utilities. In e-commerce, Web services, and online supply chains, participants negotiate about the properties of the services they request and provide to enter into binding agreements and contracts with each other. To meet negotiation requirements, participants must account for multiple issues, including functionality and quality issues, such as response time, cost, and price.

In an Internet-based service environment, multiple service requestors and providers will likely be negotiating simultaneously. Concurrent negotiation is both time efficient and robust when an entity (a software agent) negotiates with multiple other entities to choose the best offer, and essential when an agent requests a service involving multiple agents, as in a supply-chain problem.

In many-to-many negotiations, each agent can be involved in multiple negotiations with different participants at the same time. However, each individual negotiation involves only two agents, so these negotiations are bilateral. Accordingly, we call these *many-to-many bilateral negotiations*. In this article, we consider a competitive environment and assume the agents are self-interested and know only their own negotiation preferences. We present a protocol that supports many-to-many negotiation in which many agents negotiate with many other agents simultaneously.[1]

Petri nets are a graphical and mathe-

 Published by the IEEE Computer Society 1089-7801/06/$20.00 © 2006 IEEE

matical modeling tool applicable to many systems. They're a promising tool for describing and studying concurrent, asynchronous, distributed, and parallel information-processing systems.[2] Because colored Petri nets (CPNs) support concurrency and have greater expressive power than finite state machines (FSMs), we use them as a modeling language for our concurrent negotiation protocol.

Negotiation Protocol

We combine the two-phase commit protocol[3] from database transaction theory and the extended contract-net protocol (see the "Related Work in Service Negotiation" sidebar) to introduce two phases of accept and reject into the alternating-offers protocol[4] and thus support concurrent multi-issue negotiation. During a negotiation session, an agent can use several messages to communicate with its opponent. Table 1 defines the negotiation performatives.

In multiple-issue negotiation, different agents have different preferences over the negotiation issues. Their preferences are usually represented as utility functions with issues as variables. In alternating-offer protocols, an agent makes an offer that gives it the highest utility at the beginning of the negotiation, and then incrementally concedes by offering its opponent a proposal that reduces the utility as the negotiation progresses.

Let a and b represent the negotiating agents and I a set of n negotiation issues, where $I = \{I_1, I_2, \ldots I_n\}$. Given $O_{b\to a,k}$ representing an offer from b to a at negotiation round k, we define a's utility as $U_a(O_{b\to a,\,k})$. We define agent b's utility similarly.

Definition 1: In a negotiation in which agent a negotiates with a set of agents $B = \{b_1, b_2, \ldots, b_n\}$ concurrently, agent b_i's offer $O_{bi\to a,k}$ is better than agent b_j's offer $O_{bj\to a,k}$ if and only if $U_a(O_{bi\to a,k}) > U_a(O_{bj\to a,\,k})$.

Definition 2: In a negotiation in which agent a negotiates with a set of agents $B = \{b_1, b_2, \ldots, b_n\}$ concurrently, agent b_i's offer is *acceptable* to agent a at round k if

$$U_a(O_{bi\to a,k}) \geq U_a(O_{a\to bi,\,k+1}) \text{ and}$$
$$U_a(O_{bi\to a,\,k}) = \text{argmax}\, U_a(O_{bi\to a,\,k}) \text{ for } b_i \in B,$$

where the argmax function returns the max value from a set.

As Figure 1 illustrates, a requestor agent a_1 locates two provider agents, b_1 and b_2, and initiates two negotiation threads simultaneously by sending each agent its proposal. After evaluating the proposal, b_2 sends its counterproposal to a_1. Although b_1 is negotiating with another requestor agent a_2 when it receives a_1's proposal, b_1 sends its counterproposal to a_1, because b_1 has not yet reached an agreement with a_2. After evaluating b_2's counterproposal, a_1 finds it acceptable and pre-accepts it, while prerejecting other counterproposals at the same time. Agent b_1 receives the preject from a_1 and a pre-accept from a_2, so it sends the formal proposal to a_2 and prerejects all other requestors. While pre-accepted b_2 sends a_1 its formal proposal, other prerejected agents send their counterproposals to a_1. If b_2's formal proposal is still acceptable, a_1 accepts b_2 and rejects all other providers. Similarly, a_2 sends the accept message to b_1 and the reject message to other providers.

Few existing protocols can handle this situation properly. In some protocols, for example, if b_1

Table 1. Negotiation performatives.

Messages	Descriptions
Propose	A requestor initiates the negotiation by proposing an offer for a service.
Counterpropose	An agent offers a new proposal in response to the previous proposal.
Formally propose	An agent formalizes its pre-accepted proposal.
Pre-accept	An agent temporarily accepts a proposal.
Prereject	An agent temporarily rejects a proposal.
Accept	An agent accepts a proposal.
Reject	An agent rejects a proposal.

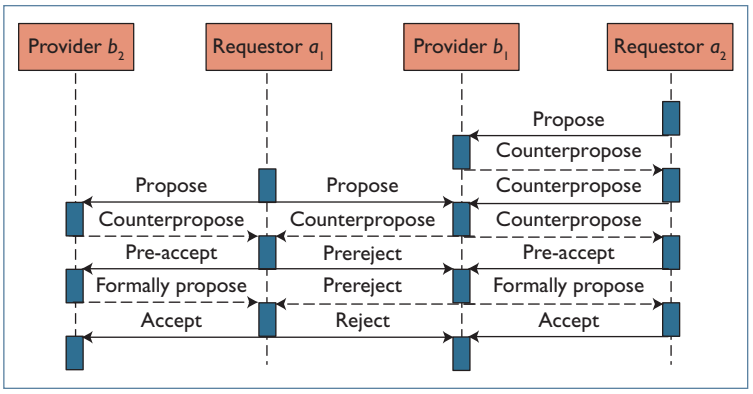

Figure 1. A concurrent negotiation sequence diagram. Requestor agents a_1 and a_2 negotiate for services with two provider agents, b_1 and b_2. Negotiation continues among requestor and provider agents until two agents reach an acceptable agreement and reject all other proposals.

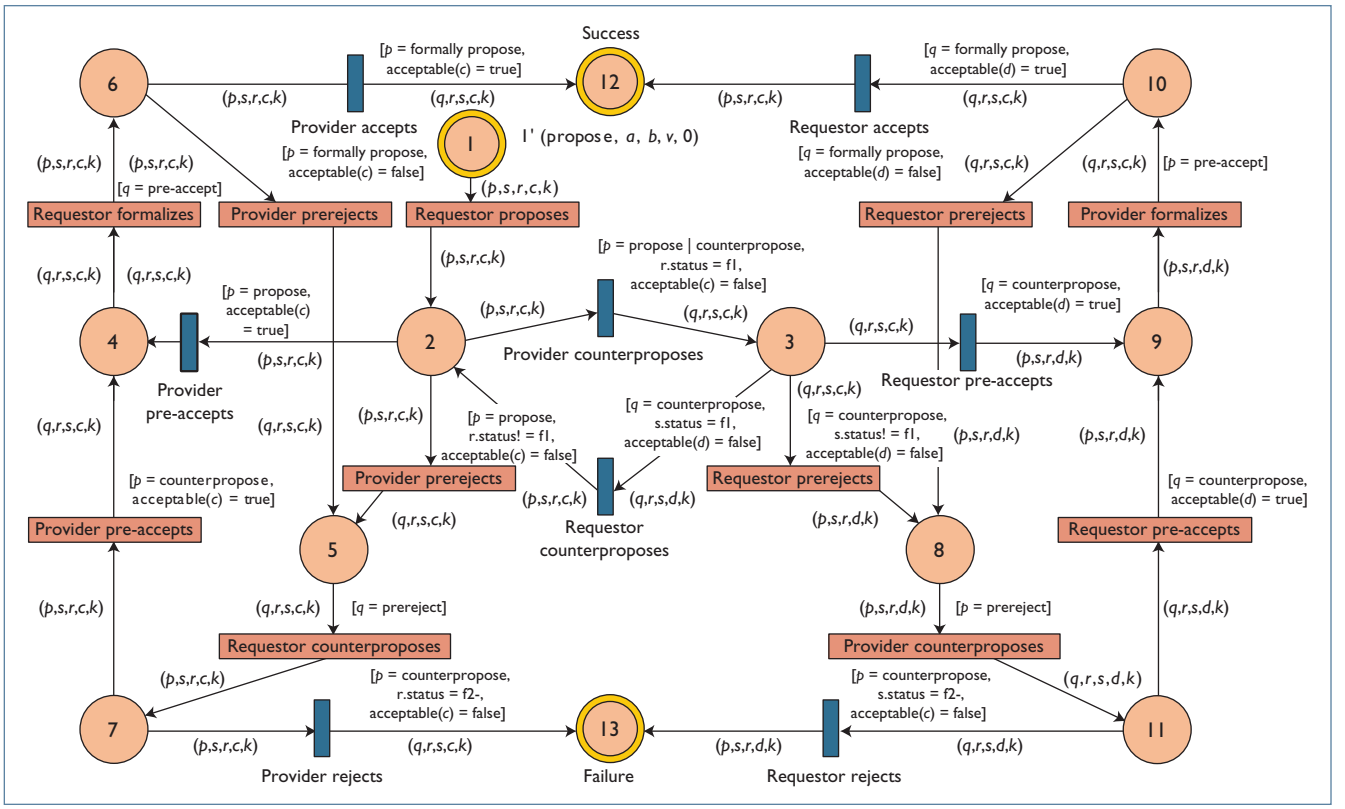

Figure 2. Colored Petri net for concurrent negotiation. Circles represent places, or colored Petri net (CPN) states; rectangles represent transitions between states.

is one of the providers negotiating with a_1 concurrently, b_1 must wait until all of a_1's negotiation threads end. Even if b_1 reached an agreement with a_1 earlier, a_1 can still reject it; the agreement isn't finalized until a_1 finishes all of its negotiation threads. These protocols bind b_1 to a one-sided commitment and cause it to lose time and reduce its chances of reaching a contract with other agents. Such protocols are biased in favor of a_1, but still cause a_1 the trouble of a likely decommitment to the previously agreed proposals, and lead to a loss of utility (*decommitment penalty*) and reputation that would harm an agent interested in long-term cooperation and gains.

Colored Petri Nets

To successfully negotiate in an Internet-based environment, service requestors and provider agents must comply with an interaction protocol. The protocol should be correct, unambiguous, complete, and verifiable. The ability to express correct protocols depends on the specification language or tool used to model the protocol. FSMs can express a variety of protocols in a conceptually simple and intuitive way. However, they aren't adequately expressive to model more complex interactions, especially those with some degree of concurrency. Petri nets were originally a response to FSMs' limited modeling power. They provide the benefits of FSMs while allowing greater expressivity and concurrency.

A Petri net is a directed, bipartite graph whose nodes represent a process's possible states and actions and whose arcs represent possible transitions among states and actions. Multiple tokens that move through the graph and indicate its current status provide concurrency. In a CPN, each token is extended with a value referred to as *color*, which is a schema or type specification.

CPNs have great value for modeling a concurrent protocol because they provide a relatively simple and precise formal model, an intuitive graphical representation, full expressiveness with explicitly represented states, support for concurrency, and a firm mathematical foundation for investigating and verifying properties.

Negotiation Process

Figure 2 describes the concurrent negotiation protocol for services. For simplicity, we omitted some

constraints such as time-out and exception handling from the figure. The service requestor (state 1) initiates a negotiation process (place 1) by sending an initial proposal to the service provider. (*Places* represent the states of a CPN, and are drawn as circles or ellipses. *Transitions* represent the CPN's actions, and are drawn as rectangles. We will discuss them later.) The provider evaluates the proposal (place 2) and pre-accepts it if it's acceptable (place 4); otherwise, it counterproposes (place 3). Two entities exchange counterproposals before they find an acceptable offer (places 2 and 3). A provider can prereject a proposal if it has pre-accepted a counterproposal from another requestor or another requestor has pre-accepted its proposal (place 5). The pre-accepted requestor then sends its formal proposal to the provider (place 6). If this formal proposal is acceptable, the provider accepts it (place 12). Otherwise, the provider prerejects this proposal (place 5), and the requestor can send an improved counterproposal (place 7), which the provider could pre-accept (place 4) or reject finally (place 13).

Agents need two-phase commitment (pre-accept and accept) and the corresponding two-phase rejection (prereject and reject) to deal with concurrent encounters. Our protocol's concurrent negotiation process has two stages.

In stage one, service agents exchange counterproposals after service requestors initiate the negotiations. When an agent receives an acceptable proposal, it announces the start of negotiation stage two by sending a pre-accept to the agent who sent the acceptable proposal and a prereject to the rest of the negotiating opponents.

In stage two, the negotiation enters a process similar to a last-round first-price auction. The pre-accepted entity returns its formal proposal while prerejected agents send their counterproposals for their final tries. If the former proposal is still acceptable, the agent accepts it formally and rejects other offers, ending the negotiation. Otherwise, the agent will prereject it (we discuss this later). In Figure 2, states 1, 2, and 3 belong to negotiation stage one, and the remaining states belong to negotiation stage two.

In this model, colored tokens have an attached data value, which can be of an arbitrary complex type. Color sets determine tokens' possible values. We use the CPN modeling language (CPN ML) to make CPN declarations. Figure 3 defines the CPN color set. The CPN ML descriptions in Figure 3 also give us a formal way to express the system's negotiation process. We base the messages used on the proposed negotiation performatives.

```
color PERFORMATIVE = with propose | counterpropose | formally propose |
  prereject | pre-accept | reject | accept;
color AGENTNAME = string;
color AGENTTYPE = with requestor | provider
color STATUS = with f1 | f2– | f2+
color ID = product AGENTTYPE * AGENTNAME * STATUS;
color VALUE = with float | int | string ;
color CONTENT = list VALUE;
color MESSAGE = product PERFORMATIVE * ID * ID * CONTENT * ROUND;
color ROUND = int

var m : Message;
var p, q : PERFORMATIVE;
var c, d : CONTENT;
var s, r : ID;
var k : ROUND;

fun acceptable(c: CONTENT);
```

Figure 3. Color set declaration. In the colored Petri net modeling language, color sets determine possible values for each token.

We define the type `agentname` as a string and `agenttype` as an enumeration type containing two possible values: requestor and provider. `Status` indicates different negotiation statuses — that is, f1 denotes the first negotiation stage, in which none of the agents is prerejected or pre-accepted yet; f2– denotes the status in which one agent has prerejected other agents; and f2+ denotes the status in which one agent has pre-accepted another agent.

The type `id` is the product of the types `agentname`, `agenttype`, and `status`. The `content` type represents the possible negotiation offer in a multi-issue negotiation; we define it as a list of values of integer, real, or string types. We also define several variables with the declared types and a Boolean function `acceptable( )`, which takes a `content` variable as the argument and produces a true/false value, as explained in Definition 2.

Each place, or CPN state, has an associated type (color set) determining the kind of data that the place can contain. A CPN's colors can be arbitrarily complex, such as a message with integer, real, string, and list fields as in Figure 3. We write the place type to the lower left or right of the place. All places have the same type — `message` — so we omit type definitions for them, as in Figure 2.

A CPN state is called a *marking*, which consists of several tokens distributed among the CPN's places. The tokens on a particular place are that place's marking. Each token carries a value (color), which belongs to the type of the place on which the token resides. For example, a possible marking of place 1 is 1′(*propose*, *a*, *b*, *v*, 0). This marking contains one token with value `message` = (*propose*, *a*, *b*, *v*, 0), where *propose* is the performative, *a* is the sender agent, *b* is the receiver agent, and *v* is a list representing an offer at round *k* = 0. By convention, the prime symbol (′) denotes the number of token appearances. An initial marking describes the system's initial state and is written on the upper left or right of the place by convention. In our CPN, place 1 has an initial marking consisting of a single token with the value (*propose*, *a*, *b*, *v*, 0). Initially, the remaining places contain no tokens.

To enable a transition in a marking, we must be able to bind data values to the variables appearing on the surrounding arc expressions and in the transition's guard such that

- each of the arc expressions evaluate to tokens that are present on the corresponding input place, and
- the guard (if any) is satisfied.[5]

As Figure 3 shows, the `requester propose` transition has five variables:

- *p* of type `performative`,
- *s* of type `id`,
- *r* of type `id`,
- *c* of type `content`, and
- *k* of type `round`.

We assign data values to these variables by creating the following binding: $p \leftarrow proposal$, $s \leftarrow a$, $r \leftarrow b$, $c \leftarrow v$, $k \leftarrow 0$. In addition to the arc expressions, we can attach a list of Boolean expressions (with variables) to each transition. This list of expressions — or *guard* — specifies that we only accept bindings for which all Boolean expressions evaluate to true.

For example, the `provider counterpropose` transition uses a guard with three Boolean expressions: (*p* = proposal|counterproposal, *r*.status = f1, acceptable(*c*) = false). The result is true only if all expressions are true. Except for the operator "." — which we use to extract `status` information from a variable of type ID in the arc expression r. status — the expressions' meanings are straightforward.

An occurrence of the `provider counterpropose` transition removes a token with value (*propose*, *a*, *b*, *v*, 0) from place 2 and adds a token to output place 3. We determine the values of the tokens removed from an input place by evaluating the arc expression on the corresponding input arc. Similarly, we determine the values of tokens added to an output place by evaluating the arc expression on the corresponding output arc. After evaluating the guard, the `provider counterpropose` transition updates the token by changing the performative to counterpropose, switching the sender and receiver, updating its offer, and incrementing round *k*.

An occurrence sequence describes a CPN execution and specifies the markings that are reached and the steps that occur. In the initial marking, the `requestor propose` transition is enabled in a binding. Hence, the occurrence sequence can be continued. This leads to a new marking in which evaluating the binding and guard values enables one of the transitions (`provider counterpropose`, `provider pre-accept`, and `provider prereject`). An *infinite occurrence sequence* consists of an infinite number of markings and steps and corresponds to a nonterminating system execution.

Figure 2 models concurrent pairwise negotiation between a service requestor and a provider. Our proposed protocol lets service requestors and service providers manage several negotiation processes in parallel.

Negotiation Algorithm

Figure 4 illustrates the proposed negotiation mechanism for service requestors. `Evaluating CounterProposal` deals with all counterproposals in phase one. It evaluates the counterproposals from the providers and sends a pre-accept, prereject, or counterpropose regarding the evaluation result. `EvalutingCounterProposal2` evaluates all counterproposals in phase two. It sends a pre-accept to the sender if its proposal is acceptable; otherwise, it sends a reject. `Evaluating-FormalProposal` evaluates the formal proposal from the pre-accepted entity. It sends an accept if the formal proposal is acceptable and a prereject otherwise. The CPN diagram for a service provider is similar to the diagram in Figure 4. However, it starts with a `ReceiveProposal` transition that leads to an `Evaluating` place, which is equivalent to `EvaluatingCounterproposal` in Figure 4.

Because we assume that the agents are self-interested, an agent can propose a good offer in stage one to scare off its competition and then send a lower formal proposal later. It won't suc-

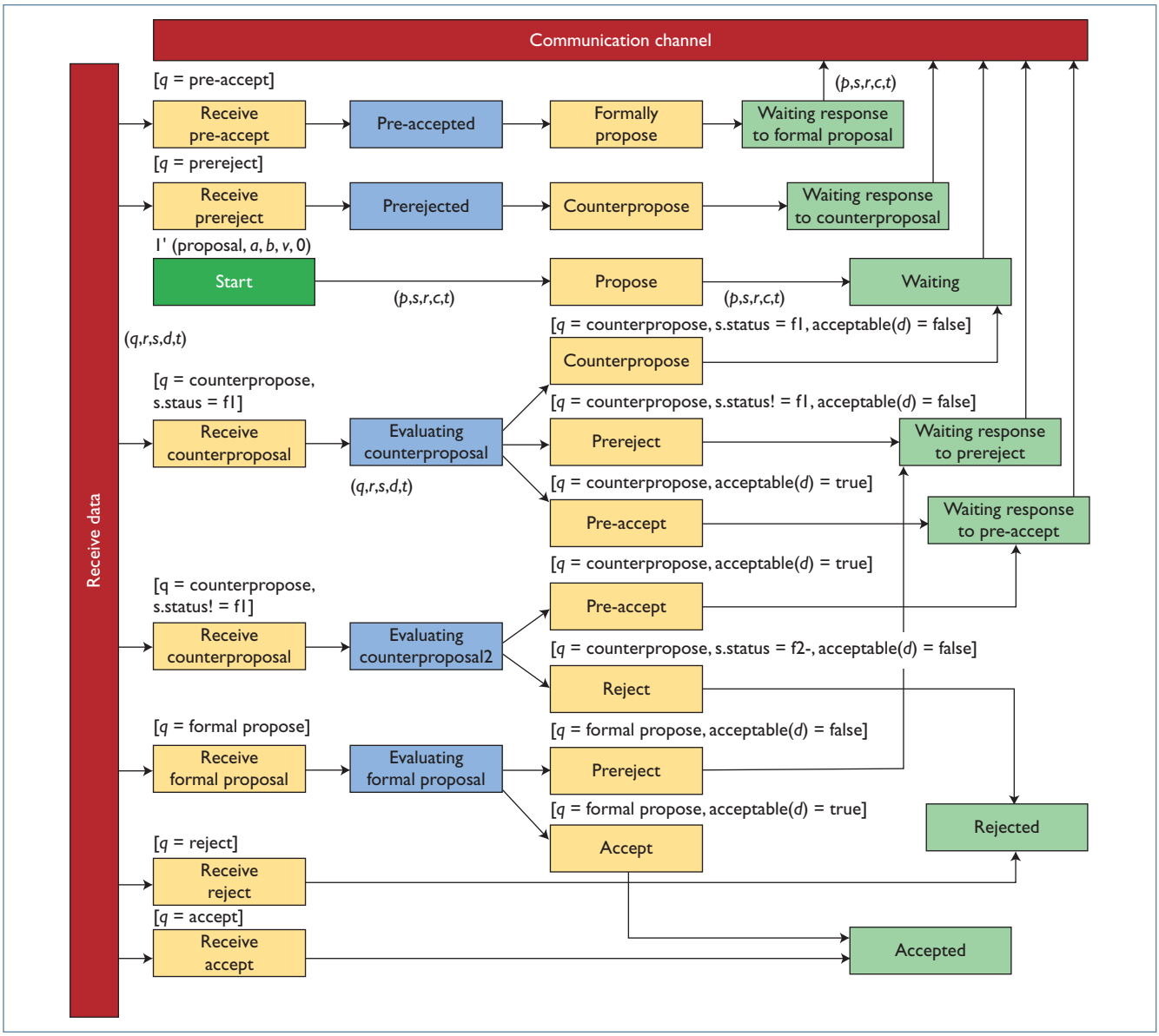

Figure 4. Colored Petri net for a service requestor. A service requestor initiates a negotiation, exchanges counterproposals with its opponents, and then ends a negotiation through two phases of accept and reject.

ceed because other agents' counterproposals will likely beat this lower proposal. In addition, we enforce a negotiation strategy that avoids this situation. Thus, the system peremptorily rejects any formal proposals that are worse than their pre-accepted proposals. In this case, the requestor will pre-accept the best offer from the received counterproposals as a replacement.

Theoretical Analysis

Any negotiation protocol should guarantee that a negotiation process following it will eventually terminate.

We define a *termination property* as follows: given a set of service requestors A and a set of service providers B, a negotiation process engaged by the entities from A and B using our concurrent protocol ends after a finite number of steps.

Proof: From Figure 2, we can see that three loops can occur during the negotiation process:

- a loop on states 2 and 3;
- a loop on states 5, 7, 4, and 6, and
- a loop on states 8, 11, 9, and 10.

Related Work in Service Negotiation

Negotiation for services involves a sequence of information exchanges among parties to establish a formal agreement among them, whereby one or more parties will provide services to one or more other parties.

Iyad Rahwan, Ryszard Kowalczyk, and Ha Hai Pham consider negotiation as a distributed constraint-satisfaction problem.[1] Their framework supports one-to-many negotiation by coordinating several concurrent one-to-one negotiations and giving the coordinator several possible negotiation strategies. However, this framework can't handle the difficult issues arising in many-to-many negotiation, such as consistency, coordination, and decommitment risk.

Thuc Duong Nguyen and Nicholas Jennings present a heuristic model for coordinating concurrent negotiation and an integrated commitment model, which lets agents reason about when to commit or decommit.[2] However, their model is obviously biased in the buyer's favor. To mitigate the problem, they let the seller decommit, or stray from its commitment, by paying a precomputed decommitment penalty. Both the buyer and seller can then renege on the previous deal. Incorporating seller decommitment into their model has no advantage, because in one-to-many negotiation a seller has no other buyers, so it has no incentive to decommit and will never do so rationally. On the other hand, breaking a commitment is always a hard decision because it typically involves issues beyond a decommitment penalty, such as reputation and user feedback.

Sakir Aknine, Suzanne Pinson, and Melvin Shakun present an extended version of the contract-net protocol to support concurrent negotiation processes for a task contractor.[3] Their protocol is more efficient and failure tolerant than the basic contract-net protocol. However, it doesn't allow counterproposals, which are important in negotiations involving time constraints, especially when multiple issues are involved.

A conversation-based approach's value is largely determined by the conversational model it uses. Petri nets are a well-known graphical and mathematical modeling tool that researchers have used to model agent communication interactions. Wil van der Aalst applies message sequence charts to specify the interaction between organizations by using Petri nets to model the workflow inside the organization.[4]

References

1. I. Rahwan, R. Kowalczyk, and H.H. Pham, "Intelligent Agents for Automated One-to-Many E-Commerce Negotiation," *Proc. 25th Australasian Conf. Computer Science*, Australian Computer Soc., 2002, pp. 197–204.
2. T.D. Nguyen and N.R. Jennings, "Reasoning about Commitments in Multiple Concurrent Negotiations," *Proc. 6th Int'l Conf. E-Commerce*, ACM Press, 2004, pp. 77–84.
3. S. Aknine, S. Pinson, and M.F. Shakun, "An Extended Multi-Agent Negotiation Protocol," *Proc. Int'l Conf. Autonomous Agents and Multi-Agent Systems*, Kluwer Academic, 2004, pp. 5–45.
4. W.M.P. van der Aalst, "Interorganizational Workflows: An Approach Based on Message Sequence Charts and Petri Nets," *Systems Analysis, Modeling, Simulation*, vol. 34, no. 3, 1999, pp. 335–367.

To prove that the protocol will end in a finite number of steps, we must prove that none of the three loops could have an infinite number of steps.

In loop 2–3, service entities exchange counterproposals through the alternating-offer protocol, in which an entity must concede to offer deals that its opponents are more likely to accept if it prefers an agreement to the conflict deal. These principles apply to all concurrent negotiation threads. Many existing mechanisms can guarantee that agents will continue progressing during the negotiation. With a predefined minimum hop, one entity will eventually pre-accept the counterproposal from its opponents and exit from loop 2–3 or time out. Also, an agent can be prerejected out of loop 2–3 when its opponent pre-accepts a proposal from a peer in another concurrent negotiation thread.

In loop 5–7–4–6, the prerejected requestor sends its new counterproposal to the provider and is pre-accepted; however, its formal proposal is then prerejected and it must send a new counterproposal. Assume a service provider b negotiates concurrently with a set of n service requestors: a pre-accepted requestor a_0 and a set A' $\{A - a_0\}$ of $n - 1$ prerejected requestors. After the provider receives the formal proposal from a_0 and the counterproposals from A', let a_i be the one with the best offer from A'. By comparing the offers from a_0 and a_i, we have two possible situations.

If $U_b\left(O_{a_i \to b}\right) \leq U_b\left(O_{a_0 \to b}\right)$, requestor a_0 will be accepted and reach state 12, all requestors from A' will end with rejections and reach state 13, and the negotiation will end.

If $U_b\left(O_{a_i \to b}\right) > U_b\left(O_{a_0 \to b}\right)$, requestor a_0 will be prerejected and reach state 5, requestor a_i will be pre-accepted and enter state 4, and all other agents will be rejected and out of the loop. This leaves only a_i and a_0. The negotiation ends if one of them can overbid the other in two consecutive rounds. An infinite loop occurs when a_i and a_0 keep overbidding each other by a tiny amount. We can prevent this by defining a minimum increment ε or enforcing a time constraint on the protocol. Because both sides would benefit from reaching a contract earlier, the provider should

consider the time factor for proposals received in phase two. For example, before comparing two proposals, we can apply a time discount function $\delta(k) < 1$ (for example, a normalized function whose value decreases exponentially with the time) to the counterproposal that must evolve two more states to be a formal proposal. So, the counterproposal must overbid the formal proposal by an exponentially increasing amount to overrule it along the time, and negotiation will end in a finite number of steps.

As R. Scott Cost and his colleagues describe, we can check CPNs for a variety of properties.[6] Given a CPN, we might be interested in reachability: Does the initial marking result in the correct negotiation result? We can also perform a liveness test: Does the negotiation process enter a "dead" state in which no further activity can occur? CPNs offer several techniques and tools for formal analysis and verification of such properties.

Related Issues in Concurrent Negotiation

We can use our protocol to perform the concurrent negotiation process at two levels:

- The upper level (a coordinator) coordinates all of the threads and solves conflicts among them.
- The lower level (negotiation threads) deals directly with the various opponents and decides which counterproposals to send and which proposal to pre-accept.

In each round, the threads report their status to the coordinator, and the coordinator updates the other threads' status and uses one thread's progress to alter the agent's behavior in the other threads.

A *commitment* is a symmetric relationship that binds the participants in a negotiation. One-sided commitment problems occur when, for example, a service requestor reaches an agreement with one provider but continues negotiating with other providers, reneging on the earlier agreement if it receives a better offer later. As Figure 2 shows, our protocol is neutral to both service requestors and service providers. Therefore, it can eliminate the decommitment situations that arise in one-sided commitment.

In negotiation phase two, a service agent hosts a procedure that is similar to a last-round first-price auction. Because it's the prerejected agents' final try to stay in the negotiation, it makes truth-telling about the reserve offer the dominant strategy for agents whose counterproposals are close to their reserve offers. At this time, they don't want to lose by providing an offer worse than the reserve offer, and they don't want to win the negotiation with negative gains by providing an offer better than the reserve offer. Our protocol makes the negotiation more efficient.

Several possible directions exist for future work. First, we'll further investigate our protocol's effect on an agent's negotiation strategy and develop a precise commitment model. Second, we can extend the protocol to support negotiation for a composed service with different service agents under constraints such as quality-of-service and interagent dependency issues.

References

1. J. Dang and M.N. Huhns, "An Extended Protocol for Multiple-Issue Concurrent Negotiation," *Proc. 20th Nat'l Conf. Artificial Intelligence* (AAAI), AAAI Press, 2005, pp. 65–70.
2. T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE*, IEEE Press, 1989, pp. 541–580.
3. J. Gray, "Notes on Data Base Operating Systems," *Operating Systems, An Advanced Course*, vol. 60, Springer-Verlag, 1978, pp. 393–481.
4. M.J. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, 1994.
5. K. Jensen, *Coloured Petri Nets*, vol. 1, 2nd ed., Springer, 1996.
6. R.S. Cost et al., "Using Colored Petri Nets for Conversation Modeling," *Issues in Agent Comm.*, F. Dignum and M. Greaves, eds., Springer-Verlag, 2000, pp. 178–192.

Jiangbo Dang is a technical member at Siemens Corporate Research. His research interests include multiagent systems, service-oriented computing, business process and workflow management, knowledge discovery, data mining, and machine learning. Dang has a PhD in computer science from the University of South Carolina. Contact him at dangj@engr.sc.edu.

Michael N. Huhns is the NCR professor in the Computer Science and Engineering Department at the University of South Carolina, where he also directs the Center for Information Technology. His research interests include information technology, including distributed artificial intelligence and multiagent systems, machine learning, enterprise modeling and integration, and distributed database systems. He is a fellow of the IEEE and a member of the ACM and AAAI. Contact him at huhns@sc.edu.