A Generic Software Framework for Automated Negotiation

Claudio Bartolini, Chris Preist, Nicholas R. Jennings [1]
Trusted E-Services Laboratory
HP Laboratories Bristol
HPL-2002-2
January 23rd , 2002*

automated negotiation, electronic marketplaces, negotiation protocols, software framework

If agents are to negotiate automatically with one another they need to follow a shared protocol. The protocol specifies the way in which negotiation should take place, including the flow of messages to be used. To date, most research in this area has focused on defining specific protocols for different kinds of interaction. Here we propose an alternative approach. We define a simple interaction protocol which can be used in all circumstances, and a general interaction framework using this protocol. This framework can be parameterized with different negotiation rules. By choosing different sets of rules, different negotiation mechanisms can be implemented. We present a taxonomy of such rules, together with examples of specific negotiation mechanisms. We also describe our implementation of the framework using the Jade multi-agent platform integrated with the Java Expert System Shell (Jess).

* Internal Accession Date Only Approved for External Publication
[1] Dept. of Electronics & Computer Science, University of Southampton, Highfield, Southampton, UK

A Generic Software Framework for Automated Negotiation

Claudio Bartolini
HP Laboratories
Filton Road, Stoke Gifford
Bristol, UK
+ 44 117 312 8505
claudio_bartolini@hp.com

Chris Preist
HP Laboratories
Filton Road, Stoke Gifford
Bristol, UK
+ 44 117 312 8311
chris_preist@hp.com

Nicholas R. Jennings
Dept of Electronics & Computer Science
University of Southampton
Highfield, Southampton, UK
+44 23 8059 7681
nrj@ecs.soton.ac.uk

ABSTRACT

If agents are to negotiate automatically with one another they need to follow a shared protocol. The protocol specifies the way in which negotiation should take place, including the flow of messages to be used. To date, most research in this area has focused on defining specific protocols for different kinds of interaction. Here we propose an alternative approach. We define a simple interaction protocol which can be used in all circumstances, and a general interaction framework using this protocol. This framework can be parameterized with different negotiation rules. By choosing different sets of rules, different negotiation mechanisms can be implemented. We present a taxonomy of such rules, together with examples of specific negotiation mechanisms. We also describe our implementation of the framework using the Jade multi-agent platform integrated with the Java Expert System Shell (Jess).

Keywords

Automated negotiation; Electronic Marketplaces; Negotiation protocols; Software framework

1. INTRODUCTION

Recently there has been much interest in the role of dynamic negotiation in electronic business transactions. For such negotiation to be effectively automated, parties need to use a shared *negotiation protocol*. The protocol determines the flow of messages between participants and the rules by which they must abide during the negotiation. In addition, each agent needs a *negotiation strategy* which determines how it will act within the protocol to attempt to get a good outcome. The research we present in this paper focuses on the shared protocol, not the private strategy.

Various protocols are used for automated negotiation. They can be one-to-one (such as iterated bargaining [13]), one-to-many or many-to-many (such as auctions [18]). However, most state-of-the-art multi-agent systems are designed with a single negotiation protocol explicitly hard-coded in all agents (usually as finite state machines). This leads to an inflexible environment, only able to accept agents designed for it. An advance on this is provided by standardization activities such as FIPA [8] and the Open Agent Architecture [5]. These provide formal definitions of several standard negotiation protocols. A fully FIPA-compliant agent, for example, will be able to use any of these, and can be informed by another agent which is to be used in a given negotiation. This provides a limited degree of flexibility, but still requires hard-coding of all protocols.

In this paper, we propose an alternative approach. We define a generic interaction protocol and a general interaction framework using this protocol. This framework can be parameterized with different negotiation rules. Depending on the choice of rules, different negotiation mechanisms can be implemented. We present a taxonomy of such rules, together with examples of specific negotiation mechanisms. This approach has two important advantages over the state-of-the-art. Firstly, it is flexible. Only the general interaction framework needs to be agreed in advance and explicitly hard-coded in agents. Rules defining a specific protocol can be defined at any time. Secondly, protocol specifications (consisting of a small number of declarative rules) can be explicitly passed between agents and reasoned over. This means one agent can give another an explicit specification of the protocol it wishes to use. This also opens the door for future research into agents dynamically designing protocols to meet their needs, or negotiating with others over changes in a protocol specification.

In describing the architecture of our negotiation framework we take a layered approach (Figure 1). The bottom layer consists of a generic agent-oriented platform (such as the ones described in [5, 8]). This frees us from having to re-define basic services for agent communication, lifecycle management, and so on. On top of the agent-oriented platform, our negotiation framework defines: (i) a general negotiation protocol, (ii) a taxonomy of the rules of negotiation, (iii) a language to define the rules of negotiation and (iv) a language to express negotiation proposals.

In more detail, the *general negotiation protocol* (section 2.3) defines the way in which the agents interact during the negotiation. We base this protocol on an abstract model of negotiation, formed by analyzing what is common to many different forms of

AAMAS '02, July 15-19, 2002, Bologna, Italy.

negotiation (sections 2.1 and 2.2). The general protocol can be specialized to a specific negotiation mechanism by specifying *negotiation rule*s (section 2.4). We define a *language to express negotiation proposals* (section 3.3) that allows proposals to specify specific acceptable outcomes, or constraints on the space of acceptable outcomes. Finally, we introduce a *declarative language* to express negotiation rules (section 4) so that negotiation participants can reason over them. The declarative layer can then be mapped to reusable software components implementing the logic expressed by the rules.

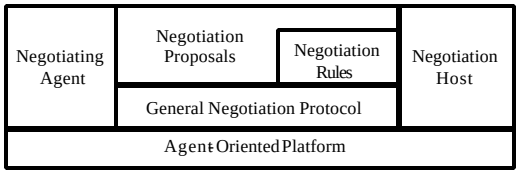

Figure 1: Abstract Architecture of Negotiation Framework

2. THE ABSTRACT ARCHITECTURE

In this section we present an abstraction of the negotiation process, developed from role-based analysis of many examples of different kinds of negotiations. From this, we identify the key abstract roles and their responsibilities and develop a general protocol for the negotiation framework.

2.1 An Abstract Negotiation Process

Negotiation takes place by parties communicating via a *negotiation locale*. This locale is an abstraction of the messaging system that is used by the negotiation participants to address each other. After admission to negotiation, a participant is given access to the locale. This locale may already exist, or may be specially created. Each participant can send proposals by sending a message to the negotiation host. Reliable delivery and security are enforced by the underlying messaging infrastructure. The negotiation host determines which of the participants should see the message and multicasts the message appropriately. This allows us to model one-to-one negotiation as a particular case of many-to-many.

To be able to negotiate with one another, parties must initially share a *negotiation template*. This specifies the different parameters of the negotiation (e.g. product type, price, supply date etc). Some parameters may be constrained (e.g. product type will almost always be constrained in some way), while others may be completely open. A negotiation locale has a negotiation template associated with it and this defines the object of negotiation within the locale.

As part of the *admission* process to the negotiation, participants must accept the negotiation template. A potential participant may also need to meet other admission policies, such as providing certain credentials, before starting negotiation.

The process of *negotiation* is the move from a negotiation template to an acceptable agreement. A single negotiation may involve many parties, resulting in several agreements between different parties and some parties who do not reach agreement. For example, a stock exchange can be viewed as a negotiation where many buyers and sellers meet to trade a given stock. Many agreements are formed between buyers and sellers, and some buyers and sellers fail to trade (see section 4.2 for more details).

During negotiation, the participants exchange *proposals* representing the agreements currently acceptable to them. Each proposal will contain constraints over some or all of the parameters expressed in the negotiation template. These proposals are sent to the negotiation host. However, before a proposal is accepted by the locale, it must be valid. To be valid, it must satisfy two criteria:

- It must be a valid restriction of the parameter space defined by the negotiation template. The constraints represent the values of parameters that are currently acceptable. Often, a constraint will consist of a single acceptable value.
- The proposal must be submitted according to the set of rules that govern the way the negotiation takes place. These rules specify (among other things) who can make proposals, when they can be made, and what proposals can be submitted in relation to previous submissions. (For example, auctions often have a 'bid improvement' rule that requires any new proposal to buy to be for a higher price than previous proposals). Such rules are specified and agreed at the admission stage.

An *agreement* is formed according to the agreement formation rules associated with the negotiation locale. When the proposals in the locale satisfy certain conditions, they are converted by these rules into agreements, and returned to the proposers. The end of a negotiation is determined by termination rules.

This abstract process can be specialised to many different negotiation styles. For example, in one-to-one bargaining, participants take turns in exchanging proposals in a previously agreed format. The rules in this case are simple. Any proposal can be made, as long as it is consistent with the negotiation template and made in turn. The negotiation terminates when the same proposal is returned unchanged (which we take as declaration of acceptance) or when one party leaves the negotiation locale. In the former case, an agreement identical to the last proposal is formed. In an English auction, the proposals specify the price of the good, every other parameter being fully instantiated in the negotiation template. Negotiation rules state that every new proposal (bid) will be valid only if it is an improvement over the current best proposal. Termination occurs at a deadline, and the agreement formed will contain the specification of the good as expressed in the negotiation template, at the price specified in the winning bid.

2.2 Roles in Negotiation

There are two main roles in negotiation – participant and host (Figure 2). The former are those who wish to reach agreement. The latter is the role responsible for enforcing the protocol and the rules of negotiation. The agent playing the host role may also play a participant role (e.g. in one-to-one negotiation) or may be non-participatory (e.g. the auctioneer in an auction). In some cases, the role of negotiation host may alternate between different entities as the negotiation progresses.

The *Negotiation Participant* can post proposals according to the rules provided by the negotiation host.

The *Negotiation Host* is responsible for the creation and enforcement of rules governing participation, execution, resolution and termination of a negotiation. It has the following sub-roles:

- Gatekeeper: Enforces policy governing admission to the negotiation.
- Proposal validator: Ensures that a proposal is well formed with respect to the negotiation template.
- Protocol enforcer: Ensures that participants' proposals are posted and withdrawn according to the negotiation rules.
- Agreement maker: Ensures that agreements are formed according to the rules.
- Information updater: Notifies participants of current state of the negotiation, according to the visibility and display rules.
- Negotiation terminator: Declares negotiation over according to what is specified in the termination rule.

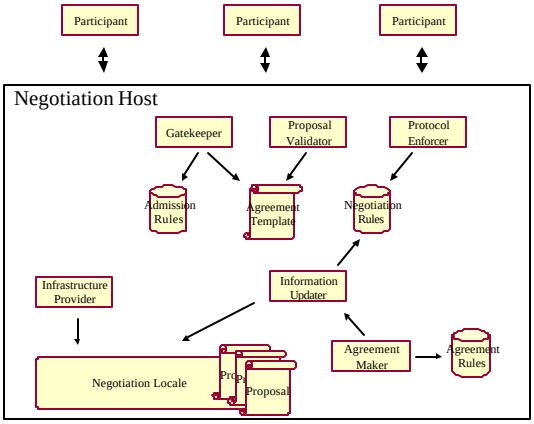

Figure 2: Abstract architecture: sub-roles & relationships

2.3 The General Negotiation Protocol

We now present the general negotiation protocol, showing how these roles interact. We do this using UML diagrams enhanced with swim-lanes (which give the views of each of the actors).

Figure 3 shows the negotiate activity.[1] We assume that a negotiation locale and a negotiation template exist. The negotiation host declares the negotiation open. Participants can then be admitted to the negotiation process if they meet the admission requirements.

The participants now submit proposals by posting them to the negotiation locale. This continues until termination is reached, as defined by the termination rules. Termination may occur after agreement formation (as in one-to-one bargaining), before agreement formation (as in a sealed-bid auction) or may be independent (as in a continuous double auction.) Each time a participant submits a proposal (Figure 4) the negotiation host, in the role of proposal validator, checks that it is a constrained form of the negotiation template and is syntactically well formed. If the proposal is not valid, it is rejected. If the proposal passes this first stage of validation, the negotiation host (playing here the role of protocol enforcer) checks that it satisfies the negotiation rules. These rules define the way in which the negotiation should take place and may include restrictions on when a proposal can be made (e.g. participants must take turns to submit) and semantic requirements on valid proposals (e.g. requirements that a proposal must improve on previous ones). If the proposal passes this second validation stage, the current set of proposals and associated data structures are updated accordingly and participants are notified. Who is notified, and the structure of the notification, is defined by the visibility rules and display rules.

[1] In this paper we will describe the negotiate activity only. See [2] for a full description of the general negotiation protocol, including the activities of admission, proposal withdrawal, initialization and finalization of the negotiation infrastructure.

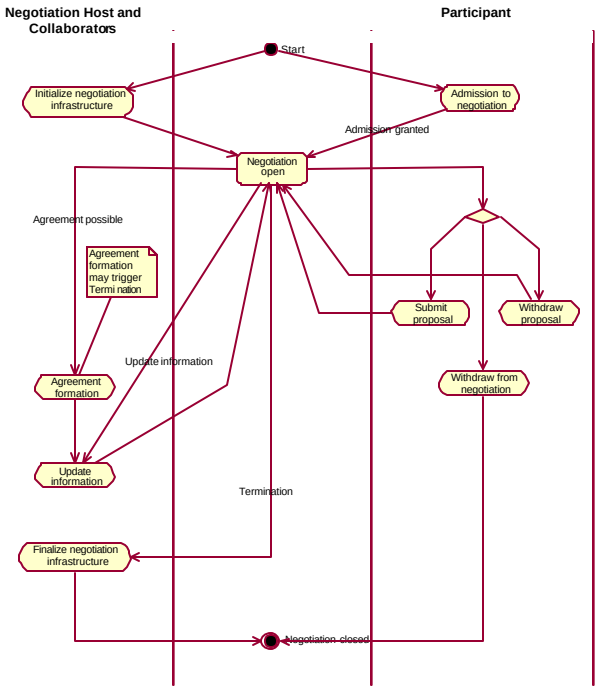

Figure 3: Negotiate Activity Diagram

An agreement formation process can be triggered at any time during negotiation, according to the agreement formation rules. The negotiation host (in the agreement maker role) then looks at the current set of proposals to determine whether agreements can be made. Agreements can potentially occur whenever two or more negotiating parties make compatible proposals. If this is the case, agreement formation rules determine exactly which proposals are matched and the final instantiated agreement that will be used.

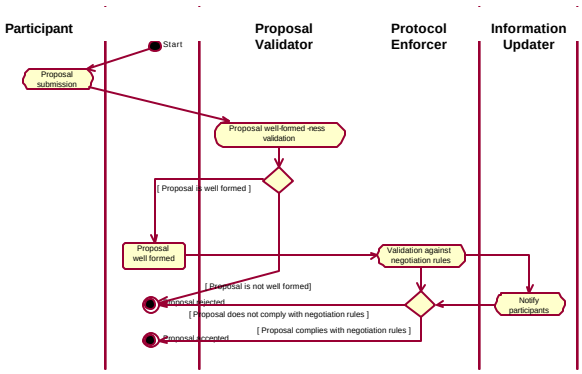

Figure 4: Proposal Submission Activity Diagram

Agreement rules may state, for example, that the highest priced offer to buy should be matched with the lowest priced offer to sell and that the final agreement will take place at the average price. Often, *tie breaking* agreement rules will be defined that will be used if the main agreement rules can be applied in several ways. For example, earlier posted offers may take priority over later ones. When the agreement formation rules have been applied to determine exactly which agreements are made, the negotiation host (information updater) notifies the participants. Figure 5 illustrates the agreement formation utility diagram.

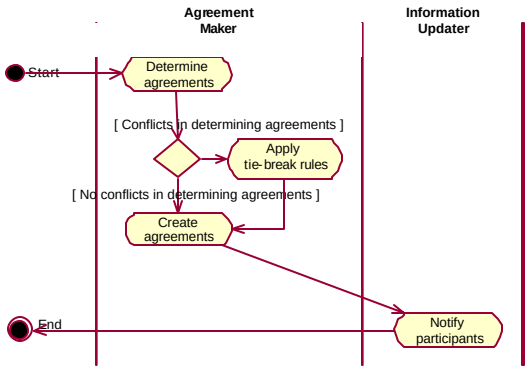

Figure 5: Agreement Formation Activity Diagram

Having defined the general protocol for negotiation, we now show how it can be specialized in a variety of different ways. We do this firstly by presenting a taxonomy of negotiation rules and then (in the context of our prototype implementation) example rules for different negotiation mechanisms.

2.4 A Taxonomy of rules for negotiation

Our analysis has identified the following categories of negotiation rules, together with the roles responsible for them.

Rules for admission of participants

Responsible role: *Gatekeeper*

Admission rules: Govern admission to negotiation

Rules for proposal validity

Responsible role: *Proposal Validator*

Validity rule: Ensures that any submitted proposal has to be compliant with the negotiation template

Rules for protocol enforcement

Responsible role: *Protocol Enforcer*

Posting rule: Determines circumstances in which a participant may post a proposal

Improvement rule: Specifies, given a set of existing proposals, what new proposals may be posted

Withdrawal rule: Specifies if and when proposals can be withdrawn, and policies over the expiration time of proposals

Rules for updating status and informing participants

Responsible role: *Information Updater*

Update rules: Specifies how the parameters of the negotiation change on occurrence of certain events

Visibility rule: Specifies which participants can view a given proposal

Display rule: Specifies if and how the information updater notifies the participants that a proposal has been submitted or an

agreement has been made – either by transmitting the proposal unchanged or by transmitting a summary of the situation

Rules for lifecycle of negotiation

Responsible role: *Negotiation Terminator*

Termination rule: Specifies when no more proposals may be posted (e.g. a given time, period of quiescence)

Rules for agreement formation

Responsible role: *Agreement Maker*

Agreement formation rules: Determine, given a set of proposals of which at least two are compatible, which agreements should be formed

Tie-breaking rule: Specific agreement formation rule applied after all others

3. IMPLEMENTING THE FRAMEWORK

In this section we present an exemplar embodiment of the abstract architecture described in section 2. In this case, the negotiation host and its sub-roles are implemented as a multi-agent system using a blackboard to communicate. The negotiation participants are autonomous agents which can access portions of this blackboard, though this access is mediated by the negotiation host.

The main task of the negotiation host agents is to evaluate negotiation rules and take actions as a consequence of the result. To do so, they use the blackboard that contains information about the negotiation as a whole (e.g. valid proposals, participants, status of the negotiation). Each of the agents is loaded with negotiation rules that it is responsible for enforcing. They execute rules either in response to a message or in response to changing data on the blackboard.

We have implemented the negotiation framework using the Jade multi-agent platform. Jade [4] is compliant with the FIPA abstract architecture [8]. The main abstractions in Jade are agents and behaviours (section 3.1) Agents communicate using messages in the FIPA Agent Communication Language (ACL) [7]. Jade provides tools for inspecting these messages and also provides a library of interaction protocols and generic agent behaviours, which we have used as the basis of our implementation. The natural way of designing the negotiation host agents is as a rule engine. To do this we use the Java Expert System Shell (Jess). Following [11], we associate a Jess rule engine with a Jade agent. We implement our negotiation rules in the Jess rule language. The agent's behaviour monitors changes on the blackboard and incoming messages, and executes rules in response to these events. Agents may write information about the negotiation on the blackboard (section 3.2). Proposals are also stored on the blackboard, provided they satisfy the negotiation template (section 3.3).

3.1 Agents and Behaviours

Our prototype system consists of a Negotiation Host agent and its sub-ordinate agents: *Gatekeeper*, *Proposal Validator*, *Protocol Enforcer*, *Information Updater*, *Negotiation Terminator* and *Agreement Maker*. Any agent can join as a negotiation participant, provided it conforms to the general negotiation protocol described in section 2.

The Negotiation Host initializes the blackboard and creates the sub-ordinate agents. It acts as a first level contact for the negotiation participants. It receives proposals and forwards them to the Protocol Enforcer. Upon termination of the negotiation, it performs finalization tasks such as putting the agents to sleep.

Each of the other agents has an associated Jess engine. When certain events occur (eg a new message or a change on the blackboard) they evaluate their rules and take the associated actions.

The Gatekeeper implements an agent-based version of a credentials-based access control system [2]. On receiving an ACL.REQUEST message from the Negotiation Host containing information on participant identity and credentials, it evaluates the admission rules to decide whether the participant should be admitted to negotiation.

The Proposal Validator receives proposals (ACL.PROPOSE) from the Negotiation Host. It validates them against the negotiation template. If a proposal is valid, it forwards it to the Protocol Enforcer. Otherwise, it informs the submitter with an ACL.REJECT_PROPOSAL message.

When the Protocol Enforcer receives a proposal from the Proposal Validator, it checks that the proposal satisfies the posting and improvement rules. It does this by invoking the Jess engine and accessing associated proposal data on the blackboard. If this succeeds, it declares the proposal valid and asserts it on the blackboard. The submitter is informed through an ACL.CONFIRM message with a proposal id. Otherwise it sends an ACL.REJECT_PROPOSAL message to the submitter. The Protocol Enforcer also processes withdrawal requests (ACL.REQUEST, where the payload is a proposal withdrawal referring to a valid proposal id), provided they satisfy the conditions of the withdrawal rules.

The Negotiation Terminator regularly checks the termination rule to determine whether the negotiation should end. The termination rule is a Jess rule stating the conditions under which termination should occur (e.g. a time-out or following agreement formation). On negotiation termination, it notifies the Negotiation Host.

At regular intervals or when a new proposal is posted on the locale, the Information Updater updates information on the blackboard appropriately. It may forward proposals to those participants eligible to see them (according to the visibility rules) and/or send a digest of the current state of the negotiation (according to the display rules).

The Agreement Maker applies the agreement formation rules to determine which agreement can be made, given the valid proposals on the blackboard. It then notifies the interested participants that an agreement has been formed (ACL.INFORM). Its action can be triggered by an internal clock, or by an event such as a new proposal arriving or the negotiation terminating.

3.2 Assertions on the Blackboard

We now give details of the knowledge base used by the agents and then give details of the negotiation proposal language and negotiation rule language which make use of this. This knowledge base is stored in the negotiation locale and is accessible by the negotiation host and its sub-agents. All examples are given as Jess assertions and rules.

3.2.1 Facts about the negotiation

The negotiation is assigned a unique ID at its start:

```
(negotiation (id Negotiation-Id))
```

Other parameters of the negotiation are asserted in the form

```
(negotiation
   (id Negotiation-Id)
   (negotiation-parameter Value))
```

For example, parameters associated with an English auction can be specified in the following way:

```
(negotiation
   (id auction-37)
   (seller-proposal Alice-37)
   (bid-increment 5)
   (termination-window 30min)
   (currently-highest-bid 0))
```

This states that auction-37 is selling a good described in proposal Alice-37 (See section 3.3), with an auction bid increment of 5. The first four fields will remain fixed, while the fifth will be updated regularly.

3.2.2 Facts about participants

When a participant is admitted, the gatekeeper asserts relevant facts in the knowledge base. The participant is assigned an ID, and associated with a negotiation.

```
(participant
   (id Participant-Id)
   (negotiation-id Negotiation-Id))
```

Other parameters of the participants are asserted in the format:

```
(participant
   (id Participant-Id)
   (negotiation-id Negotiation-Id)
   (participant-attribute-name, Value))
```

For example, based on a participant's credentials, the gatekeeper may assign them a credit limit:

```
(participant
   (id Claudio)
   (negotiation-id Auction37)
   (creditLimit 10000))
```

3.2.3 Facts about Proposal Status

Facts are asserted which specify the current status of proposals on the blackboard. For example, when a proposal is first received, its submission time is asserted by the Gatekeeper as:

```
(submission-time 01/10/01:18:37
   (proposal-id Proposal-Id))
```

When the proposal validator has checked a proposal, it asserts:

```
(valid-proposal
   (proposal-id Proposal-Id))
```

In a negotiation where new proposals can supersede old ones (such as an English auction), the Information Updater will assert facts specifying which are active currently (and retract this if the proposal is superseded.)

```
(active-proposal
   (proposal-id Proposal-Id))
```

3.3 Negotiation Proposals and Templates

The negotiation template is expressed as a collection of Jess facts and predicate constraints. In order to express complex objects, the facts may make reference to Jess templates. In them we declare which fields must appear in every proposal and which are optional. We also define the type of each field and constraints on its value. For example, a negotiation host wishing to conduct auctions of cars could define the parameters as:

```
(deftemplate proposal
   (slot submitter (type STRING))
   (slot role (type STRING))
   (slot automobile (type OBJECT))
   (slot price (type INTEGER)))
```

and constrain the initial parameter space as:

```
(proposal
   (submitter ?S&:(participant
                   (id ?S)
                   (negotiation-id ?NEG))
   (role Buyer|Seller)
   (automobile ?A)
   (price ?P))
```

Negotiation participant agents can send proposals as ACL.PROPOSE messages containing a collection of facts and predicate constraints. The Proposal Validator applies a variant of the subsumption algorithm described in [16] to determine whether the proposal is valid with respect to the negotiation template. An example of a proposal that is valid with respect to the template presented above is:

```
(proposal
   (proposal-id Alice-37)
   (submitter Alice)
   (role Seller)
   (object
      (automobile
         (make FIAT)
         (model Punto)))
   (price ?P&:(>= 3000 ?P)))
```

This states that Alice wishes to sell a Fiat Punto for at least £3000. The proposal ID is added by the Negotiation Host. In the next section we give guidelines on how to write negotiation rules for various negotiation mechanisms.

4. NEGOTIATION RULES

Agents have standard rule templates, where the rule asserts information in their private fact base. The agent responds to this information, executing appropriate actions and sending messages according to the General Negotiation Protocol.

The display rule in the Information Updater has the format:

```
(defrule display-rule
   (negotiation
      (extract_relevant_parameters))
      (process_relevant_parameters)
   => (assert
         (information-digest
            (processed_parameters)))
```

The visibility rules have a similar format, and act as filters on new proposals. They determine which participants can view which parameters of a new proposal. The information they assert is used by the Negotiation Host to mediate the view that different negotiation participants have on the blackboard.

```
 (defrule visibility-rule
   (valid-proposal
      (extract_relevant_parameters))
      (process_relevant_parameters)
   (test (required_condition))
   => (assert
         (visible-proposal
            (processed_parameters)))
```

The termination rule in the Negotiation Terminator has the format:

```
(defrule termination-rule
   (extract-required-parameters)
   (test (termination-condition))
   => (assert
         (terminate negotiation-id)))
```

Rules in the Protocol Enforcer have a different format. Both when receiving protocols and withdrawal requests, the agent must check whether a series of conditions are all true to determine its action. Because of Jess's cumbersome mechanism to support backward chaining, we implement these rules in the format:

```
(defrule rule-name
   (proposal
      (proposal-id ?Proposal-id)
      (extract_other_required_parameters))
   (test not(required_condition))
   (assert (failed rule-name ?proposal-id)))
```

The Protocol Enforcer has a meta-rule which rejects the proposal if there are any such assertions in the database after the rules have executed, and accepts it otherwise. It executes appropriate actions and sends messages as defined in the General Negotiation Protocol.

4.1 Single Item English Auction

Assume a Negotiation Host has advertised an agreement template as per section 3.3, and has been contacted by Alice to sell her Fiat Punto via auction. The Host starts a new negotiation, with id auction-37, to sell it. It generates an associated agreement template, which is a specialized version of the one in 3.3, with the automobile slot instantiated with details of her Fiat Punto. The Host asserts facts about the auction on the blackboard.

The negotiation rules which apply to the seller state that they make a single proposal, and then remain silent. In the interests of space, we omit these. The proposal Alice makes is as specified in section 3.3. This confirms the details of the good she is selling, and specifies her reservation price of £3000. Facts about the auction are updated, and now appear as in section 3.2.1.

After this, buyers place bids in the form of proposals that satisfy the buyer proposal validation rules. These are applied by the Protocol Enforcer, and have the format described above (beginning of this section). The conditions are:

[Posting rule] This tests that, if a buyer is posting a proposal, then the seller has already posted one.

```
(test (equal ?Role buyer)
   (exists
      (active-proposal
         (……)
         (role seller)))
```

[Improvement rule] The price field of the buyer's proposal must be a certain increment above the value of all previously posted buyer proposals. Hence the improvement rule contains the test:

```
(test (> ?Price
   (+ ?Currently-Highest-Price
      ?bid-increment)))
```

[Withdrawal rule] Auctions do not allow bids to be withdrawn once submitted. Hence, the withdrawal rule (in format specified in Section 5) contains `(test FALSE)` and so always fails when executed.

[Visibility rules] The seller's initial proposal is visible to all the buyers. However, the field in which the seller constrains the price to be above their reservation price cannot be viewed:

```
(defrule visibility-rule
   (active-proposal
      (proposal-id ?PID)
      (role seller))
   (test
      (TRUE))
   => (assert
         (visible-proposal
            (proposal-id
               (value ?PID)
               (visibility all))
            (price
               (value ?Price)
               (visibility none))
            (……)))
```

A similarly structured rule states that all active buyer proposals are visible to all participants. Optionally, the identity of a bidder can be maintained private.

[Display rule] The currently highest bid price is notified to all participants.

```
(defrule display-rule
   (negotiation
      (……)
      (currently-highest-bid ?CHB))
   => (assert
         (information-digest
            (currently-highest-bid ?CHB)))
```

[Termination rule] Termination occurs if the auction is inactive for longer than the termination window specified in the negotiation fact base. Hence the rule, in the format specified in the beginning of this section, contains the test:

```
(test (> ?Current-Time
      (+ ?Active-Proposal-Time
         ?Termination-Window))
```

Together with the information asserted in section 3.2, this results in Alice's auction terminating if it is inactive for 30 minutes.

Agreement formation rules

When negotiation terminates, an agreement is formed between the currently active buyer and the seller. The agreement states that the item specified in the template is sold to the buyer at the price specified in the currently active proposal.

```
(defrule agreement-formation-rule
   (active-proposal
      (proposal-id ?PID)
      (submitter ?BUYER)
      (role Buyer)
      (price ?PRICE))
   (active-proposal
      (proposal-id ?PID)
      (submitter ?SELLER)
      (role Seller)
      (price ?RES-PRICE))
   (test
      (> PRICE RES-PRICE))
   => (assert
         (agreement
            (buyer ?BUYER)
            (seller ?SELLER)
            (price ?PRICE))))
```

4.2 The Continuous Double Auction

A many-to-many Continuous Double Auction can be implemented in our framework by straightforward modification of the rules above. For example, the improvement rule requires new bids/offers to be higher/lower than the currently active bid/offer. We have one rule which matches with seller proposals, with test:

```
(test (> ?Price ?Currently-Lowest-Offer))
```

and a simlar rule for buyer proposals with test:

```
(test (> ?Price ?Currently-Highest-Bid))
```

The posting rule is modified to allow both buyer and seller proposals at any time. In addition to the highest bid, the information digest also contains the lowest offer. Termination occurs at a fixed time, so the test becomes:

```
(test (> ?Current-Time ?End-Time))
```

The only substantial change is in the agreement formation rule. Agreement is formed whenever there is a bid greater than an offer. Highest bids are matched with lowest offers, with the agreement at the midpoint.

```
(defrule agreement-formation-rule
  (active-proposal
     (proposal-id ?Seller-PID)
     (price ?Seller-price))
  (active-proposal
     (proposal-id ?Buyer-PID))
     (price ?Buyer-price))
  (currently-highest-bid ?Buyer-Price)
  (currently-highest-ask ?Seller-Price)
  => (assert
        (agreement
           (proposals
              (?Seller-PID ?Buyer-PID))
           (price (= (/ 2 (+ (?BP ?SP))…)))
```

After an agreement is made, the Information Updater will declare the next highest/lowest bid/offer to be active. This may result in more agreements being formed immediately.

5. RELATED WORK

Research on agent negotiation protocols has primarily focussed on the specification of specific protocols, often using conversations [1] specified as finite state machines. For example, Parsons et. al. define a flexible protocol for one-to-one bargaining using this approach [13]. The FIPA agent standardization effort has defined various interaction protocols, including English and Dutch auctions, as interchanges of messages in FIPA ACL [9]. These are effectively a set of one-to-one conversations which must be coordinated. Pitt et. al.[12] define a semantic framework around FIPA ACL to allow the easier specification of multi-party interactions by adding structured conversation identifiers and a richer representation of protocol states. Our approach differs from these in that rather than defining a library of protocols, we define a general protocol that can be parameterized with rules.

Esteva et.al. [6] have defined a formal approach to specifying *electronic institutions* in which agents interact. This goes beyond other work on protocols in the additional abstractions it provides. It associates different protocols to *scenes*, and provides means for specifying transition conditions from one scene to another together with normative rules associated with transition. Our work is complementary to this, in that our focus is primarily on a single scene (negotiation) and providing flexibility within it.

Wurman et. al. [17] carried out a thorough analysis of the auction design space, classifying auction mechanisms according to different parameters. This work, focussing primarily on auction

rules, provided valuable input to our analysis. Reeves et. al. [14] have also built on this to configure a general auction server with auction rules and contract templates. Their architecture is server-based, rather than agent-based, and participant agents must still be hard-coded with specific protocols. Our general negotiation protocol allows us to handle richer negotiation mechanisms than they support.

6. CONCLUSIONS AND FUTURE WORK

We have presented a general negotiation protocol and have shown how it can be parameterized with different rules to implement a variety of negotiation mechanisms. We believe this approach to agent-based negotiation has the potential to produce significantly more open and flexible multi-agent systems. Negotiation protocols no longer need to be hard-coded into the agents. Instead, agents can carry an explicit representation of a protocol (in the form of a small number of rules), passing it to new agents as they arrive. Furthermore, this explicit representation opens up the potential for future research into agents which dynamically manipulate these protocols, designing them on the fly and negotiating with other agents over which rules to use.

Because of our use of Jade and Jess to implement our system, we have presented these rules as Jess assertions. However, if our system is to be truly open, this is not adequate. We are currently working on a platform-independent specification of templates, proposals and rules in DAML+OIL [10]. (Our work on templates and proposals is presented in [16]). This will also provide additional expressivity. We are also working on extending our framework to cover multi-party agreements, linked negotiations such as the contract net [15] and hope to extend it to cover argumentation-based negotiation [13] in the future.

7. ACKNOWLEDGEMENTS

David Trastour helped with the design of the proposal language. Kannan Govindarajan provided valuable feedback during the ideas inception phase. We'd also like to thank David Bell, Naiem Dathi, Kemal Guler and Alan Karp for discussions.

8. REFERENCES

[1] Barbuceanu, M. and Fox, M.S. COOL: A language for describing coordination in multi-agent systems. In Proc. First International Conference on Multi-Agent Systems, MIT Press, 1995.

[2] Bartolini, C. and Casassa-Mont M, Digital Credentials and Authorization to Enhance Trust in Negotiation within E Services Marketplaces. In Proc. 7th HP Openview University Association Plenary Workshop, 2000.

[3] Bartolini, C. and Preist, C. A Framework for Automated Negotiation. HPL Technical Report 2001-90.

[4] Bellifemmine, F., Poggi, A., and Rimassa, G. Jade - A FIPA compliant Agent Framework. In Proc. 4th International Conference on Practical Applications of Intelligent Agents and Multi-Agent Systems, 1999.

[5] Cheyer, A. and Martin, D. The Open Agent Architecture. In Journal of Autonomous Agents and Multi-Agents Systems 4 (1/2), 143-148 2001.

[6] Esteva, M., Rodriguez, J. A., Sierra, C., Garcia, P., Arcos, J. L.,: On the formal specifications of electronic institutions, In (F. Dignum and C. Sierra eds.) Agent-mediated Electronic commerce (The European AgentLink Perspective), Springer LNAI, 2000.

[7] Foundation for Physical Agents. FIPA ACL Message Structure Specification, 2000.

[8] Foundation for Physical Agents. FIPA abstract architecture, 2000.

[9] Foundation for Physical Agents. FIPA Interaction Protocol Library Specification, 2000.

[10] van Harmelan, F. and Horrocks, I. Reference Description of the DAML+OIL Markup Language. Available from www.daml.org, 2000.

[11] Hoffmann, O., Stumptner, M. and Chalabi, T. A Perspective Based Approach to Design, in Proc. Workshop on Planning Scheduling and Configuration, KI2001, 2001.

[12] Pitt, J., Guerin, F. and Stergiou, C. Protocols and Intentional Specifications of Multi-Party Agent Conversations for Brokerage and Auctions. In Proc. Fourth International Conference on Autonomous Agents, ACM Press, 2000.

[13] Parsons, S., Sierra, C. and Jennings, N. R. Agents that Reason and Negotiate by Arguing. In Journal of Logic and Computation, 8 (3), 261-292, 1998.

[14] Reeves, D., Wellman, M. and Grosof, B. Automated Negotiation from Declarative Contract Descriptions. In Proc. Fifth International Conference on Autonomous Agents, 2001.

[15] Smith, R.G. The Contract Net Protocol: High-level communication and control in a distributed problem solver. IEEE. Trans. Computing 29, 1104-1113, 1980.

[16] Trastour, D and Bartolini C., A Semantic Web Approach to Service Description for Matchmaking of Services. In Proc. Semantic Web Working Symposium, The World Wide Web Consortium, 2001.

[17] Wurman, P, Wellman, M. and Walsh W. A Parameterization of the Auction Design Space, in Games and Economic Behavior, 35, 2001.

[18] Wurman, P, Wellman, M. and Walsh W. The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents. In Proc. Second International Conference on Autonomous Agents, 301-308, 1998.