Architecting for Reuse: A Software Framework for Automated Negotiation

Claudio Bartolini[1], Chris Preist[1], and Nicholas R. Jennings[2]

[1] HP Laboratories, Filton Rd, BS34 8QZ, Bristol, UK, +44 117 312 8311,
{claudio_bartolini,chris_preist}@hp.com
[2] Dept. of Electronics and Computer Science, University of Southampton, UK,
nrj@ecs.soton.ac.uk

Abstract. If agents are to negotiate automatically with one another they must share a negotiation mechanism, specifying what possible actions each party can take at any given time, when negotiation terminates, and what the resulting agreements will be. The current state-of-the-art represents this as a negotiation protocol specifying the flow of messages. However, they omit other aspects of the rules of negotiation (such as obliging a participant to improve on a previous offer), requiring these to be represented implicitly in an agent's design, potentially resulting in compatibility, maintenance and re-usability problems. In this paper, we propose an alternative approach, allowing all of a mechanism to be formal and explicit. We present (i) A taxonomy of declarative rules which can be used to capture a wide variety of negotiation mechanisms in a principled and well-structured way. (ii) A simple interaction protocol, which is able to support any mechanism which can be captured using the declarative rules. (iii) A software framework for negotiation, implemented in JADE [2] that allows agents to effectively participate in negotiations defined using our rule taxonomy and protocol.

1 Motivation

Recently there has been much interest in the role of dynamic negotiation in electronic business transactions. For negotiation to take place between two or more parties, they need to agree on what economists refer to as a *market mechanism* or *negotiation mechanism.* This defines the rules of the "game" which the parties are engaged in and so determines the space of the possible actions that they can take. Within this game, each party adopts a *strategy* which determines exactly which actions they make (in response to actions by other parties or external events) in an effort to maximise their gain. The mechanism must be public and shared by all parties, while an individual's strategy stays private, and is only revealed implicitly through the actions they take. For example, consider a simple market mechanism for an English auction. It is defined by the following rules: (i)The buyers can post bids at any time. (ii) A bid is only valid if it is higher than the currently highest bid. (iii) Termination occurs when no buyer has posted a bid in the last five minutes. (iv) The good is sold to the buyer with the current highest bid at the price bid.

The participants in the auction are constrained by these rules, but have a free choice of what action to take within them. A simple strategy for a buyer in such an auction is to set a maximum limit to the price they are willing to pay for the good, and to bid whenever the current highest bid is held by another buyer and is lower than their price limit.

In this paper *we consider mechanisms not strategies*. If a negotiation is to be automated, all agents need a shared understanding of the mechanism. This is done through the specification of a negotiation protocol. The protocol determines the flow of messages between participants, specifying when an agent can send a message, and what messages it can send as valid responses to specified incoming messages. For example, a negotiation protocol for the English auction states that (among other things) that potential buyers send messages specifying their bids to the auctioneer, and receive an accept or reject message in response. When the auction terminates, all participants receive a message informing them of who the winner is, and the winning bid.

Various protocols are used for automated negotiation. They can be one-to-one (such as iterated bargaining [7]), one-to-many or many-to-many (such as auctions [8]). 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 [4]. FIPA provides formal definitions of several standard negotiation protocols. The FIPA protocol for an English auction, described informally above, is shown in [4].

However, these negotiation protocols only formalise the interactions between the agents involved. They specify the permissable flow of messages, but omit information regarding other aspects of the *rules of negotiation* in a market mechanism. For example, the FIPA English Auction protocol does not specify the criteria for a bid being acceptable (i.e. that it must be greater than the current highest bid) or the conditions under which the auction will terminate (i.e. that no bids have arrived in the last few minutes). Hence, because the multi-agent environment does not make these explicit, the designer of an agent using the protocol must be aware of these negotiation rules and design their agent taking them into account. As a result of this, with the exception of the interaction aspects, the negotiation mechanism is implicit in the design of the multi agent system [6].

From the perspective of good software engineering practice, this approach has several severe disadvantages:

1 Because a negotiation protocol underspecifies a negotiation mechanism, it is not sufficient to merely standardise the protocol. For example, two agents accurately implementing the FIPA English auction protocol may differ in their criteria as to what makes a bid acceptable. One may accept any bid greater than the current highest bid, while another may require the bid to be at least a minimum increment over it. A standard must

either make decisions on exactly which negotiation rules apply, or provide a standardised way of agreeing them.

2 The designer of an agent must make themselves aware of the implicit assumptions in any given negotiation protocol, to ensure that their agent behaves appropriately. For this to happen, the designers of a multi-agent system must effectively document these, and ensure that all agent designers comply with them.

3 If the standard specification of a market mechanism is changed, all the agents that comply with it need to be updated to cope with the changes. As the mechanism rules are implicit rather than explicit, unless the agents has been very carefully designed for generality at the beginning, this will be a non-trivial task.

4 Let us assume that a multi-agent system contains several mechanisms which are similar (for example English auctions which have different bid increments or closing conditions). There is no standard way of an agent determining these details and adjusting its behaviour appropriately. This makes reuse of agents between multi-agent systems more difficult.

Given these observations, we propose an alternative to that currently adopted by FIPA. Our approach allows negotiation rules to be explicitly specified and categorised both at the design and at the implementation stage of agent oriented software development. We carry out an analysis of a generic negotiation process, which is able to capture common aspects of a wide variety of types of negotiation.[1] From there we derive: (i) A taxonomy of declarative rules which can be used to capture a wide variety of negotiation mechanisms in a principled and well-structured way. (ii) A simple interaction protocol, which is able to support any mechanism which can be captured using the declarative rules. This approach has the following advantages:

1 The generic negotiation process and rule taxonomy provide valuable conceptual tools for software engineers designing multi-agent systems which involve negotiation mechanisms. Their application will result in the mechanisms being represented in a more modular and explicit way than current approaches.

2 A set of rules together with an interaction protocol will fully specify a negotiation mechanism. Because of this, all information required for the design of agents using the negotiation mechanism is explicit and well-structured. This makes agent design and implementation easier, and reduces the risks of unintentional incorrect behaviour. This also opens the door for future research into creation and analysis of novel market mechanisms through exploration of new combinations of rules.

3 Because the rules specifying the negotiation mechanism are explicitly represented in the system, it is possible for an agent to reason over them

[1] We cover most of the bargaining and auctioning protocols that are found in literature. We will extend the framework to cover argumentation-based negotiation as future work.

to determine its behaviour and strategy. Ideally, an agent would be able to participate effectively in an arbitrary negotiation mechanism specified by any set of rules. This is far beyond current state-of-the-art in negotiation technology. However, there are negotiation algorithms able to participate in several different negotiation mechanisms, and able to adjust their behaviour depending on the details of the mechanism. For example, [3] present an agent algorithm able to simultaneously participate in multiple English, Dutch and Sealed Bid auctions, requiring details of bid increments, closing times and sealed bid winner announcement times to determine its exact behaviour. An agent using such an algorithm could identify auctions of different types by checking the mechanism rules against templates, and could identify parameter values in the rules to determine the mechanism details.

To demonstrate the validity of this approach, in this paper we also describe a software framework for automated negotiation, implemented in JADE [2] that allows agents to effectively participate in negotiations defined using our rule taxonomy and protocol. The software framework can form a highly modular and reusable component in a multi-agent system. It advances the state of the art beyond the negotiation protocol approach because (i) it can be used to implement a wide variety of negotiation mechanisms simply by instantiating it with appropriate sets of rules. (ii) It is easy to maintain and update. If a software engineer determines that a particular negotiation must change its mechanism, all they need do is adjust the rules appropriately. (iii) Agents involved in that negotiation can access the new rules, so at worst can identify that their current behaviour is inappropriate and issue a warning. A more advanced agent would be able to automatically modify their behaviour as necessary, provided the changes to the mechanism were not too great.

2 The Generic Negotiation Framework

In this section, we present an abstraction of the negotiation process, developed from the analysis of many different negotiations, both automated and human. From this, we develop a general protocol for negotiation.

2.1 An Abstract Negotiation Process

The roles involved in the negotiation process are *negotiation participant* and *negotiation host*. In some market mechanisms participants address one another, whereas in others (e.g. auctions), participants send messages to a negotiation host that forwards them to other participants that have the right and interest in seeing them. Our abstraction is that participants always publish their proposals on a common multicast space, the *negotiation locale*, which is managed by the negotiation host. The negotiation locale can be considered as a form of blackboard, with access to write and visibility of information on it mediated by the negotiation host. Visibility rules are associated to proposals so that only the participants that have right to see them can see them. This allows us to see one-to-one and one-to-many negotiation as a particular case of many-to-many.[2]

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 first action to be taken is for a participant to require *admission* to the negotiation. Admission consists of a simple conversation between the participant and the host where the participant requests admission to a particular negotiation and presents its credentials. Based on the credentials that the participant presents, the negotiation host decides whether to admit the participant to negotiation and informs the participant of the decision. If the participant is admitted, then we move onto the negotiation itself. The admission step is very important because it is when participants are informed of the rules of negotiation.

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 (e.g. price). 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. The constraints expressed in the negotiation template remain static as the negotiation proceeds.

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.

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

[2] This model always requires the participants to trust the negotiation host. Trust consideration are beyond the scope of this paper.

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. For example, in an English auction the termination rule would state that the auction finishes when no participant has placed a bid for a certain time, and the agreement formation rule would state that an agreement is formed between the highest bidder and the seller, at the price the bidder has bid.

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 Definition of the Generic Negotiation Protocol

We begin by describing the admission phase. The protocol requires the participant requesting admission to send an `ACL.PROPOSE`[3] message to the negotiation host. The payload of the message may contain credentials of the participant. The negotiation host replies either with an `ACL.ACCEPT_PROPOSAL` or an `ACL.REJECT_PROPOSAL` message, signifying admission (resp. rejection) of the participant to the negotiation.

After admission, the participants submit proposals by posting them to the negotiation locale. Participants do so by sending an `ACL.PROPOSE` message to the negotiation host. Proposal submission 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 the negotiation host checks that it is a constrained form of the negotiation template and is syntactically well formed.

[3] We use FIPA ACL messages to describe the protocol. Other ACLs could equally be used.

If the proposal is not valid, it is rejected. The submitter is notified with an `ACL.REJECT_PROPOSAL` message. If the proposal passes this first stage of validation, the negotiation host 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 the submitter and other participants are notified. Who is notified, and the structure of the notification, is defined by the visibility rules and display rules. The submitter is notified through an `ACL.ACCEPT_PROPOSAL` message. Others are notified through `ACL.INFORM` messages.

An agreement formation process can be triggered at any time during negotiation, according to the agreement formation rules. The negotiation host 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.

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 notifies the participants with `ACL.INFORM messages`.

Having defined the general protocol for negotiation (for a more complete specification and graphical representation, see [1]), 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.

A Taxonomy of Rules for Negotiation. Our analysis [1] has identified the following categories of negotiation rules:

Rules for admission of participants

Admission rules: Govern admission to negotiation.

Rules for proposal validity

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

Rules for protocol enforcement

Posting rule: Determines when 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
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 agreement formation
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.
Rules for lifecycle of negotiation
Termination rule: Specifies when no more proposals may be posted (e.g. a given time, period of quiescence).

3 Implementation of the Software Framework

In our software framework, the negotiation host functionality is implemented as a responsible agent and a set of subsidiary agents (cmp. [6]). Each sub-agent is responsible for the enforcement of one of the categories of rules described in section 2.2: *Gatekeeper* (admission), *Proposal Validator*, *Protocol Enforcer*, *Information Updater* (updating status and informing participants), *Negotiation Terminator* (lifecycle of negotiation) and *Agreement Maker*. Each sub-agent interacts with other agents, both via direct messaging and by sharing data using a blackboard system. Any agent can join as a negotiation participant, provided it conforms to the generic negotiation protocol described in section 2.

The main task of the negotiation host agents is to evaluate negotiation rules and take actions as a consequence. To do so, they use the blackboard which contains information about the negotiation as a whole (e.g. valid proposals, participants, status of the negotiation). Each of the agents is initialised with the 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. Full details of the abstract architecture are given in [1].

We have implemented the negotiation framework using the JADE multi-agent platform. JADE [2] is compliant with the FIPA abstract architecture. Agents communicate using messages in the FIPA Agent Communication Language (ACL) [4]. 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 negotiation host sub-agents are implemented as rule engines using the Java Expert System Shell (JESS). Following [5], 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. Proposals are also stored on the blackboard, provided they satisfy the negotiation template. Facts are asserted on the blackboard as JESS assertions. For example, parameters associated with an English auction can be specified in the following way:

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

Facts are also asserted about participants and proposal status (valid, active).

3.1 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)) ; submitter is a participant
   (role BuyerSeller) ; role defined as either buyer OR seller
   (automobile ?A)     ; must declare an auto
   (price ?P))         ; and a price
```

Negotiation participant agents can send proposals as `ACL.PROPOSE` messages containing a collection of facts and predicate constraints. The Proposal Validator determines whether the proposal is valid with respect to (i.e. *is subsumed by*) the negotiation template by checking. An example of a proposal that is valid with respect to the template presented above is:

```
(proposal
   (proposal-id Alice-37)  ;ID is generated by the Negotiation Host
   (submitter Alice)
   (role Seller)             ; Alice wishes to sell...
   (automobile
         (make FIAT)         ;.. a FIAT Punto....
         (model Punto))
   (price ?P\&:(>= 3000 ?P))) ;... for at least 3000.
```

In the next section we give guidelines on how to write negotiation rules for various negotiation mechanisms.

3.2 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. For example, the *display rule* in the Information Updater has the format:

```
(defrule display-rule ; declare the rule name
   (negotiation
      (...)          ; extract and process relevant parameters
   => (assert
         (information-digest (...)))
   ; assert processed parameters to be published in the info digest
```

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 and process relevant parameters
      (test (...))      ; test the required condition
   => (assert (visible-proposal (...)))
   ; if valid, assert that the proposal is visible
```

The *termination rule* in the Negotiation Terminator has the format:

```
(defrule termination-rule
   (...)           ; extract and process relevant parameters
   (test (...))       ; test the termination condition
=> (assert (terminate <negotiation-id>)))
; if termination condition is met, assert negotiation is terminated
```

Rules in the Protocol Enforcer (both *posting* and *withdrawal*) 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 JESSś cumbersome mechanism to support backward chaining, we implement these rules in the format:

```
(defrule <rule-name>}
   (proposal (proposal-id ?Proposal-id)
      (...))              ; extract any other relevant parameters
   (test not(...))        ; REQUIRED CONDITION IN A NEGATED FORM!!!
   =>  (assert (failed <rule-name> ?proposal-id)))
   ; if the condition is NOT met, assert the proposal is NOT valid
```

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.

Single Item English Auction. Assume a Negotiation Host has advertised an agreement template as per section 3.1, 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 3.1, 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 stated in the end of section 3.1.

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 body of the withdrawal rule (in format specified earlier in this section - *posting* and *withdrawal* rules) 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, 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 ?B-PID) (submitter ?BUYER)
      (role buyer) (price ?PRICE))
   (active-proposal
      (proposal-id ?S-PID) (submitter ?SELLER)
      (role seller) (price ?RES-PRICE))
   (test
      (> PRICE RES-PRICE))
   => (assert
         (agreement
            (buyer ?BUYER) (seller ?SELLER) (price ?PRICE))))
```

The single item English auction mechanism is presented here as an example to give a flavour of how to write rules. A more extensive collection of mechanisms is presented in [1].

4 Conclusions

In this paper, we have discussed the shortcomings of the state-of-the-art in representing negotiation mechanisms in agent oriented software engineering. Specifically, we have shown that the protocol approach (adopted by FIPA and many others) results in only part of a mechanism being explicitly formalised and standardised, which can result in significant drawbacks from a software engineering perspective. Alternatively, we propose a modular approach to negotiation mechanisms; a generalised interaction protocol which can be specialised with declarative rules. We provide a taxonomy of such rules, a general framework which implements this approach and give examples of rules for an English auction.

References

1. Claudio Bartolini, Chris Preist, and N. R. Jennings. A generic software framwork for automated negotiation. *HP Technical Report*, TR 2002-2, 2002.
2. Fabio Bellifemmine, Agostino Poggi, and Giovanni Rimassa. Jade - a fipa compliant agent framework. In *4th International Conference on Practical Applications of Intelligent Agents and Multi-Agent Systems*, 1999.
3. Andrew Byde, Chris Preist, and Nicholas R. Jennings. Decision procedures for multiple auctions. In *Proceedings of the 1st Joint International Conference on Autonomous Agents and Multi-Agent Systems, to appear*, 2002.
4. Foundation for Physical Agents. Fipa abstract architecture specification, 2000.
5. O. Hoffmann, M. Stumptner, and T Chalabi. A perspective based approach to design. In *Workshop on Planning, Scheduling and Configuration, KI2001*, 2001.
6. N. R. Jennings, T. J. Norman, and P. Faratin. ADEPT: An agent-based approach to business process management. *ACM SIGMOD Record*, 27(4):32–39, 1998.
7. Simon Parsons, Carles Sierra, and Nick Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
8. Peter R. Wurman, Michael P. Wellman, and William E. Walsh. The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 301–308, New York, 9–13, 1998. ACM Press.