A Software Framework for Automated Negotiation♦ (Revised and Updated)

Claudio Bartolini, Chris Preist, Nicholas R. Jennings
HP Laboratories Palo Alto
HPL-2006-33
February 17, 2006*

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 is the structure of the resulting agreements. Current standardization activities such as FIPA [2] and WS-Agreement [3] represent 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 incompatibility, maintenance and re-usability problems. In this chapter, 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 that allows agents to effectively participate in negotiations defined using our rule taxonomy and protocol and (iv) a language for expressing aspects of the negotiation based on OWL-Lite [4]. We provide examples of some of the mechanisms that the framework can support.

* Internal Accession Date Only
♦ This chapter is an updated and extended version of [1] C. Bartolini, C. Preist, N.R. Jennings – *Architecting for Reuse: A Software Framework for Automated Negotiation* in F. Giunchiglia, J. Odell, G. Weiß (Eds.) Agent-Oriented Software Engineering III, Springer-Verlag LNCS 2585/2003

Approved for External Publication

A Software Framework for Automated Negotiation*

Claudio Bartolini[1], Chris Preist[2], and Nicholas R. Jennings[3]

[1] HP Laboratories. Page Mill Rd., Palo Alto, CA 94304, USA
claudio.bartolini@hp.com
[2] HP Laboratories. Filton Road Store Gifford Bristol BS34 8QZ, UK
chris.priest@hp.com
[3] University of Southampton. Southampton SO17 1BJ, 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 is the structure of the resulting agreements. Current standardization activities such as FIPA [2] and WS-Agreement [3] represent 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 incompatibility, maintenance and re-usability problems. In this chapter, 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 that allows agents to effectively participate in negotiations defined using our rule taxonomy and protocol and (iv) a language for expressing aspects of the negotiation based on OWL-Lite [4]. We provide examples of some of the mechanisms that the framework can support.

1 Introduction

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 (individual or collective) 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) ter-

* This chapter is an updated and extended version of [1] C. Bartolini, C. Preist, N.R. Jennings *Architecting for Reuse: A Software Framework for Automated Negotiation*, in F. Giunchiglia, J. Odell, G. Weiß (Eds.) Agent-Oriented Software Engineering III, Springer-Verlag LNCS 2585/2003.

R. Choren et al. (Eds.): SELMAS 2004, LNCS 3390, pp. 213–235, 2005.

mination occurs when no buyer has posted a bid in the last five minutes; (iv) after termination, 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 chapter we consider mechanisms not strategies. In particular, we are concerned with the definition of interaction protocols underpinning a mechanism, rather than the emerging properties of the mechanism itself (for an example of the latter, compare [5]). 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 [6]), one-to-many or many-to-many (such as auctions [7]). 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 [2] and WS-Agreement [3]. FIPA provides formal definitions of several standard negotiation protocols. The FIPA protocol for an English auction, described informally above, is shown in [8].

However, these negotiation protocols only formalise the interactions between the agents involved. They specify the permissible 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 [9].

All the considerations made above also apply WS-Agreement [3], a standard proposed by The Global Grid Forum (GGF). WS-Agreement includes the definition of a simple interaction protocol to support one-to-one negotiation, with the likely aim to support different mechanisms in the future through definition of multiple interaction protocols.

We propose an alternative to that currently adopted by FIPA and GGF. 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.

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 and (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 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. Negotiation algorithms have been developed that are able to participate in several different negotiation mechanisms, and to adjust their behaviour depending on the details of the mechanism. For example, [10] 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. Using the negotiation framework that we present, 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 our approach, in this chapter we also describe a software framework for automated negotiation 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 (see [11]), 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.

The remainder of this chapter is organized as follows: section 2 describes the generic negotiation framework, built upon the definition of an abstract negotiation process and a taxonomy for the rules of negotiation. Section 3 describes a prototype implementation of the negotiation framework. Section 4 presents a number of sample negotiation mechanisms that can be embodied by the framework. We discuss related work in section 5 and move to the conclusions in section 6.

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[1].

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. Much like in [13], 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.

---

[1] This model always requires the participants to trust the negotiation host. Trust issues between participants and the negotiation host are addressed through the use of convertible undeniable signatures [12]. The imposition that proposals have to be signed with convertible undeniable signatures gives the protocol the following desirable properties. Even though proposals are invisible to the negotiation host, when an agreement is formed (i) participants cannot falsely claim ownership of the proposals and (ii) participants cannot repudiate the proposals that they have submitted, unless by refusing to collaborate in a revelation process.

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 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 Taxonomy of Rules for Negotiation

So far we have been talking about negotiation rules in a very generic fashion. It is useful at this point to divide the negotiation rules into categories. By examining the

flexibility points of the abstract negotiation process described in the previous section, – for a more complete analysis see [14] – we 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.

Rules for Lifecycle of Negotiation
Termination Rule: Specifies when no more proposals may be posted (e.g. a given time, period of quiescence).

2.3 Definition of the Generic Negotiation Protocol

The three main phases of the generic negotiation protocol are: admission, proposal submission and agreement formation.

Admission Phase
We begin by describing the admission phase. The protocol requires the participant requesting admission to send an ACL.PROPOSE[2] 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 (respectively rejection) of the participant to the negotiation. It has to be noted that this is a straightforward rendition of the

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

FIPA propose interaction protocol, represented in figure 1 [8]. (Notice that in the FIPA protocol, our participant plays the role of the initiator, and the negotiation host plays the participant.)

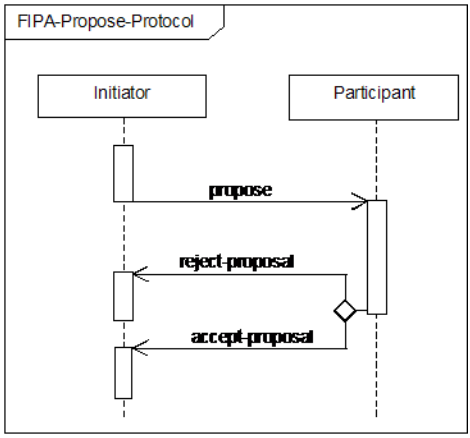

Fig. 1. The FIPA propose interaction protocol

Proposal Submission Phase

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, whose payload contains the proposal. 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 syntactically well formed and it is a more constrained version of the negotiation template.

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. Once again, the protocol here described is compliant with the FIPA propose interaction protocol. Following the rules for updating negotiation status and informing participants, other participants may be notified through ACL.INFORM messages.

Agents submitting proposals may also withdraw proposals if the rules of negotiation allow them to. This is done through sending an ACL.CANCEL message where the communicative act that is being canceled is the previous instance of the proposal (all this is done according to the FIPA cancel meta-protocol [8].

Agreement Formation Phase
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 [14]), 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.

3 Implementation of the Software Framework

In our software framework, the negotiation host functionality is implemented by a responsible agent with a set of subsidiary agents. 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 initialized 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 [14].

We have implemented the negotiation framework using the Jade multi-agent platform [15]. Jade is compliant with the FIPA abstract architecture [2]. The main abstractions in Jade are agents and behaviours (section 3.1) Agents communicate using messages in the FIPA Agent Communication Language (ACL) [16]. 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 [17], we associate a Jess rule engine with a Jade agent. We implement our negotiation rules in the Jess rule language. The agent's behavior 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 Behaviors

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 (e.g. a new message or a change on the blackboard) they evaluate their rules and take the associated actions. This overall process is represented in Fig 2 (negotiate activity diagram).

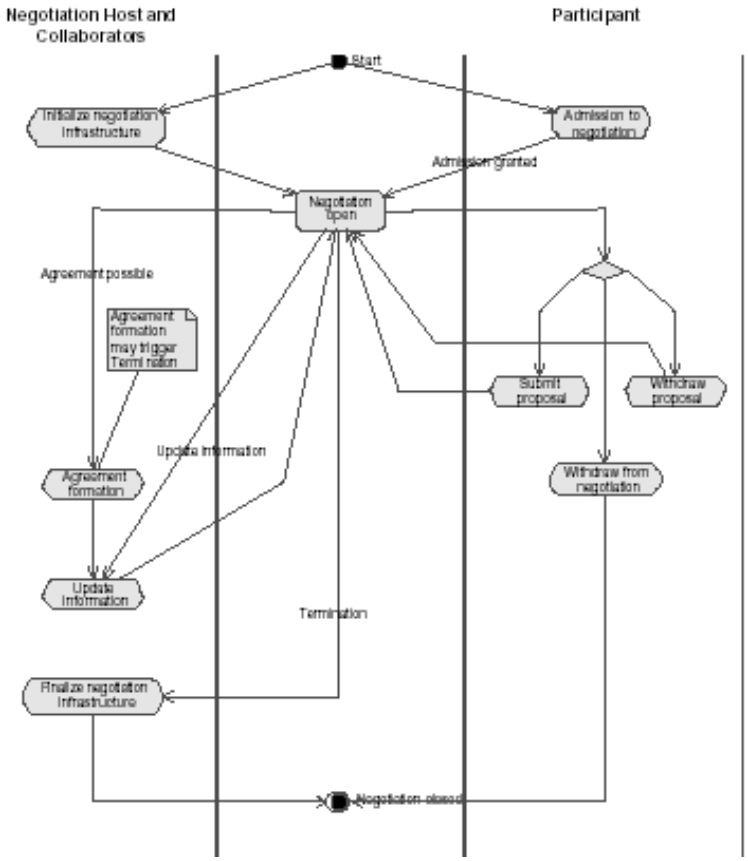

Fig. 2. Negotiate Activity Diagram

The Gatekeeper implements an agent-based version of a credentials-based access control system [18]. 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 (Fig. 3) 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).

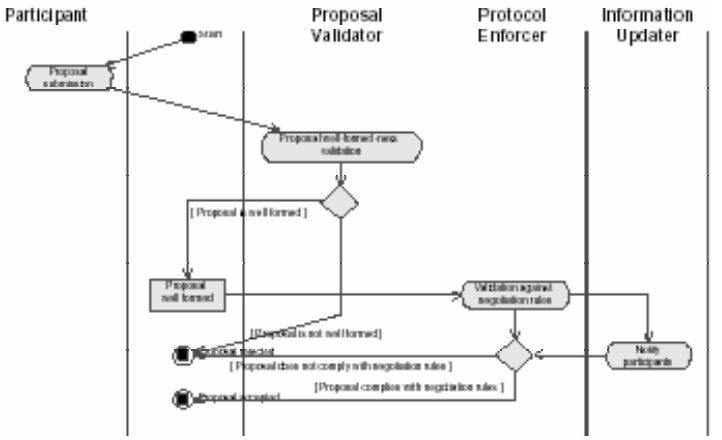

Fig. 3. Proposal Submission Activity Diagram

The Agreement Maker (Fig. 4) 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 the arrival of a new proposal or the termination of the negotiation.

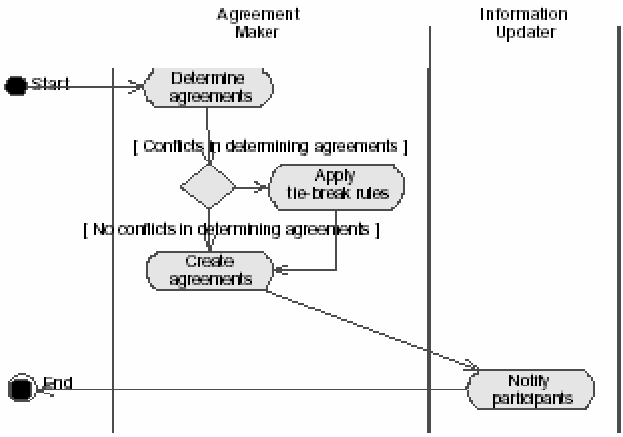

Fig. 4. Agreement Formation Activity Diagram

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.

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.

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 Bob)
    (negotiation-id auction37)
    (creditLimit 10000))
```

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 proposals are currently active (and retract this if the proposal is superseded).

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

3.3 Negotiation Proposals and Templates

The negotiation template and proposals are expressed as OWL-Lite descriptions [4]. We chose OWL-Lite because of its flexibility and expressiveness; the support that it offers for the creation and maintenance of ontologies and finally because it lends itself quite naturally to supporting the subsumption operation [19] that as we will see later is central to the functioning of our framework.

For a more in-depth discussion on why OWL-Lite and its precursor DAML+OIL [20] satisfy the requirements for a language for negotiation proposals and templates, see [19]. However, the choice of a description logic based language such as OWL-Lite is not to be intended as fundamental to the approach but is broadly indicative of what basic principles could be applied in designing the language.

For simplicity of exposition, here and in the following examples we will adopt a modified description logics notation [21] to express the proposals and the templates which is equivalent to the RDF OWL-Lite syntax [4]. XML Schema classes are not described but it should be clear by their names what they actually mean.

Before presenting the template and negotiation proposals, here are some descriptions of the concepts used. For brevity reasons, we will not exhaustively state all the description, but it should be quite intuitive to the reader what those concepts mean. For a more comprehensive description of the terms not defined here – such as Sale, Product and Participant descriptions for example – see [19].

The Car class is a subclass of Product and must have at most one Model and Make.

$$\text{Car} \subseteq \text{Product} \cap (=\exists 1\ \text{hasModel.Model}) \cap (=\exists 1\ \text{hasMakeMake})$$

```
Model = {Punto, TT, S80}
Make = {Ford, Audi, Volvo}
```

A negotiation host wishing to conduct auctions of cars could define the template as:

```
Template1 = Template ∩ Sale ∩
    ∀item.Car ∩
        ∀unitPrice.above2000 ∩
        ∀quantity.1 ∩
    ∀isComposedOf.(Delivery ∩ ∀date.before20041231)
```

A negotiation proposal must be a specialization of the negotiation template associated with the ongoing negotiation. According to the general protocol, negotiation participant agents can send proposals as ACL.PROPOSE messages containing a negotiation proposal specified as above. 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:

```
Proposal1 = Proposal ∩
    ∀seller.Alice ∩
    ∀item.(Car ∩ ∀hasMake.Fiat ∩ ∀hasModel.Punto) ∩
    ∀unitPrice.above3000 ∩ ∀quantity.1 ∩
∀isComposedOf.(Delivery ∩
∀date.between20041201and200412131))
```

This states that Alice – who is described as a participant in the participant ontology (see [19]) – wishes to sell a Fiat Punto for at least £3000 with delivery date after Dec, 1st 2004. The template requests that it also be specified that the delivery date be before the end of 2004.

When a negotiation terminates with an agreement acceptable to both parties, this agreement must specify the service that is going to be exchanged in an exact and non-ambiguous manner.

The main benefit of the choice of a description logic based language for expressing templates, proposals and agreements comes from the fact that the operations to be carried out over these descriptions by the subsidiary agents during the proposal validation and agreement formation phase can be reduced to the basic operations of checking for *satisfiability* and *subsumption* between descriptions that description logic reasoners can carry out effectively and efficiently [22].

Validation: The proposal validator, on receiving a proposal P, must initially check that it is valid. It is valid if it is a more constrained version of the negotiation template T for this negotiation. In description logic, this means that the negotiation host must check that T subsumes P. Formally, this can be specified as:

$$valid_T(P) \Leftrightarrow P \subseteq T$$

Agreement Formation: The agreement former come into action to identify all pairs of proposals which are compatible. Protocol specific rules are then used to determine exactly which of these pairs are used to form an agreement, and how exactly to generate the final agreement. A set of descriptions are compatible if their intersection is satisfiable:

$$compatible(D1,\ldots,Dn) \Leftrightarrow \neg(D1 \cap \ldots \cap Dn \subseteq \bot)$$

Hence, the first stage of agreement formation can be specified as follows:
Let Φ be the set of all valid proposals currently active on the negotiation locale.

$$potentialAgreements(\Phi) = \{(P_i, P_j) | compatible(P_i, P_j) \wedge i \neq j\}$$

When an agreement is formed, it can be verified a posteriori that the agreement subsumes the proposals that were used to form it and therefore the original negotiation template. Note that only two atomic operations are required to define the operations specified above:

- $satisfiability\ (\neg(X \subseteq \bot))$
- $subsumption\ (X \subseteq Y)$.

As noted above, a standard description logics reasoner is able to carry out both of these. Satisfiability lies at the core of such a reasoner, as all other reasoning or inference techniques are transformed into satisfiability checks. The subsumption operator is already defined by the OWL-Lite subClassOf, because our service descriptions are expressed as OWL-Lite classes (i.e. description logics concepts). A description logics reasoner can check whether two concepts subsume each other [22].

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

3.4 Negotiation Rules

Subsidiary 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 parame-
ters
```

```
        from the DL description in the payload[3]
    => (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 parame-
ters
    (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 nego-
tiation 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 JESS´s 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
```

[3] In this example and in some of the following, we omit the adaptation code for extracting relevant parameters from the payload of the message that is sent from the participant to the message. As an example of how the parameters are processed, the proposal exemplified in section 3.3 would be asserted on the blackboard as:

```
(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))) ;... reservation price: 3000.
```

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 Sample Mechanisms

In this section, we present a few examples of market mechanisms that the negotiation framework can support. For each of the mechanisms we give a flavor of the rules that need to be specified and the negotiation template and the negotiation proposals that participants may exchange.

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. 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, the expected delivery date, and specifies her reservation price of 3000. Facts about the auction are updated, and now appear as stated in the footnote[5] of section 3.4.

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 (section 3.4). 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 ?Termi-
nation-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))))
```

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.

4.3 Simple Shop Front

The framework can also model one-to-one negotiation such as a simple shop front. In this example the shop is a car dealership. The actors involved in the simple car dealership scenario are the car dealer and one or more buyers. A prospective buyer plays the participant role, whereas the shopkeeper plays both the participant and the negotiation host roles at the same time. The car dealership is modeled following the negotiation locale abstraction.

Before negotiation begins, the shopkeeper decides the admission policy, negotiation template, and negotiation and agreement formation rules.

Once again the template is identical to the one in the example given in section 3.3, expressing the cars that the dealer is willing to sell, minimum price and earliest delivery date.

The car dealer adopts standard 'shop front take it or leave it' negotiation rules. These state that[4]:

[Posting rule] A buyer may post a proposal at any time, irrespective of posted proposals by other buyers. A seller may post proposals at any time.

[Termination rule] Termination occurs when there are no seller proposals posted in the shop front

[Withdrawal rule] A seller may withdraw proposals at any time so long as they have not been matched yet with buyer's proposals. Proposals from the buyers are committing, so buyers cannot ever withdraw proposals.

The car dealer adopts standard shop front agreement formation rule:

[Agreement formation rule] Agreements are formed whenever a buyer posts a proposal identical to the seller's proposal.

After rules have been specified, negotiation can begin. The car dealer in its seller role (Alice) submits proposals for all goods it sells. The seller's proposals take a similar form as the example given in section 3.3.

If it expects high demand, it can place several identical proposals on the table for the same good. If all proposals for a given good are accepted, and the car dealer still has more in stock, it resubmits identical proposals. A buyer submits a proposal, an identical copy of the car dealer's proposal, when it wishes to purchase a given good. Agreement formation occurs as the car dealer– in the referee role – identifies valid buyer proposals and sends agreements to the buyers.

4.4 Multi-party Contracts

The examples given so far addressed the formation of two-party contracts, whatever the number of participants. The negotiation framework though extends quite naturally to the case of agreements among multiple parties playing different roles, noting a couple of observations.

To begin with, admission can be conditioned to being able to bid for one or more roles. Participants submit proposals specifying the role they want to play, selected from the role (or roles) for which they have been admitted. The proposals may also constrain who should (or should not) play the other roles.

Secondly, visibility rules enforce that participants that have been admitted to play a certain role have a restricted view over other participant's proposals. Each participant will only be able to see the part of the other proposals that are directly relevant to the role they want to fulfill. This enables entities to propose modifications to relevant parts of the contract without having access to other non-relevant parts. When all parties have agreed, each will have proposed a partly-instantiated contract that is consistent with all the others and hence the negotiation host will be able to produce the final contract according to the agreement formation rules.

As an example, imagine that a multi-party agreement is sought between a building contractor and other participants to fulfill the role of a carpenter, builder, and electri-

[4] For this example we do not present the rules in Jess language for reasons of space. However, they are similar enough to the ones in the two previous examples that the attentive reader will not have problems deriving them.

cian. Participants are admitted to the negotiation bidding to undertake the roles that they specialize for. The agreement template will include general information accessible to all parties, such as general recital information, boiler plate terms etc. Other parts of the agreement template might be restricted to fewer roles. The rest of the negotiation process is carried out exactly as described in section 2.1. The only difference is that the resulting agreement will concern more than two roles which therefore will be assigned to more than two participants.

5 Related Work

Research on agent negotiation protocols has primarily focused on the specification of specific protocols, often using conversations [23] specified as finite state machines. For example, Parsons et. al. define a flexible protocol for one-to-one bargaining using this approach [6]. The FIPA agent standardization effort has defined various interaction protocols, including English and Dutch auctions, as interchanges of messages in FIPA ACL [8]. These are effectively a set of one-to-one conversations which must be coordinated. Pitt et. al.[24] 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. WS-Agreement [3] defines a simple interaction protocol aimed at supporting one-to-one negotiation. 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.

Research in negotiation in the semantic web domain spun from the concern of demonstrating that semantic web languages can provide useful semantic support to the processes of matchmaking and negotiation [19] therefore only marginally touching on the problem of defining interaction protocols.

Naftaly Minsky's Law Governed Interactions (LGI) [25] is a paradigm for agent co-ordination that can presents similarities to our approach. However, the scope of LGI is much wider than just negotiation and applies to a much wider variety of coordination mechanisms. It's true that LGI has been applied to peer-to-peer auctions [26], but the focus of that work was mainly on the peer-to-peer aspect, aiming to dispense with a centralized service for auctions. In contrast, our framework is especially designed for providing a protocol that can embody multiple negotiation mechanisms. In this chapter, we describe a reference implementation for the framework based on Jess, but one could envisage populating the taxomomy of negotiation rules that we propose through LGI laws. Similar considerations apply to comparing the framework here described with the work of Artikis et al. [27].

Esteva et.al. [13] 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.

Reeves et. al. [28] 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. Other architectures for negotiating agents have been proposed [29] that present a neater separation of concerns between the definition of the protocols according to the principles described in this chapter and the construction of the negotiating agents.

Wurman et. al. [30] carried out a thorough analysis of the auction design space, classifying auction mechanisms according to different parameters. This work, focusing primarily on auction rules, provided valuable input to our analysis.

Mechanism design has recently had a surge in popularity [15, 5] as a foundation for building multi-agent software systems. We envisage that the generic negotiation framework described in this chapter could provide a useful platform for experimenting with it, given the flexibility that it provides for the declarative definition of interaction protocols underpinning different the negotiation mechanisms.

6 Conclusions

In this chapter, we have discussed the shortcomings of the representation of negotiation mechanisms in standardization activities such as FIPA [2] and the Global Grid Forum's (GGF) WS-Agreement [3]. Specifically, we have shown that the protocol approach adopted by them 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 generalized interaction protocol which can be specialised with declarative rules. We provide a taxonomy of such rules and a software framework that implements this approach and give examples of rules for various negotiation mechanisms. The aim of our framework is to go beyond what is currently offered by the existing standards, to provide a flexible approach to defining negotiation protocols enforcing the rules of the negotiation without having to adopt a fully-fledged coordination mechanism à la LGI [25]. We believe that our framework covers a wide variety of negotiation mechanisms – of which we give a flavor in section 4 - and gives a mechanism designer the possibility of easily creating new combination of negotiation rules.

References

1. Bartolini, C., Preist, C., Jennings N.R.: Architecting for Reuse: A Software Framework for Automated Negotiation, in Giunchiglia, F., Odell, J., Weiss G. (Eds.): Agent-Oriented Software Engineering III (2002), Springer-Verlag LNCS 2585/2003.
2. Foundation for Physical Agents. Fipa abstract architecture specification, 2000. Available at www.fipa.org.
3. Andrieux A.et. Al.: Web-Services Agreement Specification (WS-Agreement) Global Grid Forum Recommendation. Available at www.ggf.org
4. Dean, M., Schreiber G.: OWL Web Ontology Language Reference W3C Recommendation. Available at www.w3c.org
5. Dash, R. K, Jennings, N. R., Parks, D. C.: Computational Mechanism Design: A Call to Arms. IEEE Intelligent Systems (2003), vol. 18 (6), 40-47.
6. Parsons, S., Sierra, C., Jennings, N.R.: Agents that reason and negotiate by arguing. Journal of Logic and Computation (1998), 8(3), 261–292.

7. Wurman, P.R., Wellman, M. P., Walsh, W.E.: The Michigan Internet AuctionBot: A configurable auction server for human and software agents. In Sycara, K. P., Wooldridge M. (eds), Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98) (1998), 301–308, New York, 9–13, 1998. ACM Press.
8. Foundation for Physical Agents. FIPA Interaction Protocol Library Specification, 2000. Available at www.fipa.org
9. Jennings, N. R., Norman, T. J., Faratin, P.: ADEPT: An agent-based approach to business process management. ACM SIGMOD Record (1998), 27(4), 32–39.
10. Byde, A., Preist, C., Jennings, N.R.: Decision procedures for multiple auctions. In Proceedings of the 1st Joint International Conference on Autonomous Agents and Multi-Agent Systems (2002), 613-620.
11. Sandholm, T.: Automated Mechanism Design: A New Application Area for Search Algorithms. In proceedings International Conference on Principles and Practice of Constraint Programming (CP-03), 2003.
12. Boyar. J., Chaum, D., Damgard, I. Pedersen, T.: Convertible Undeniable Signatures; Crypto '90, LNCS 537, Springer-Verlag, Berlin (1991), 189-205.
13. Esteva, M., Rodriguez, J. A., Sierra, C., Garcia, P., Arcos, J. L.,: On the formal specifications of electronic institutions, In Dignum F. Sierra, C. (eds.) Agent-mediated Electronic commerce (The European AgentLink Perspective), Springer LNAI. (2000)
14. Bartolini, C. Preist, C., Jennings N.R.: A Generic Software Framework for Automated Negotiation, HP Laboratories Technical Report HPL-2002-2, (2002)
15. 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)
16. Foundation for Physical Agents. FIPA ACL Message Structure Specification, 2000. Available at www.fipa.org
17. Hoffmann, O., Stumptner, M., Chalabi, T.: A perspective based approach to design. In Workshop on Planning, Scheduling and Configuration, KI2001 (2001)
18. 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).
19. Trastour, D., Bartolini, C., Preist C.: Semantic Web Support for the Business-to-business E-Commerce Lifecycle In Computer Networks: The International Journal of Computer and Telecommunications Networking, Vol. 42, Issue 5 (2003) Special Issue on The Semantic Web: an Evolution for a Revolution - North Holland / Elsevier
20. van Harmelen, F. and Horrocks, I. Reference Description of the DAML+OIL Markup Language. Available from www.daml.org, (2000)
21. Baaders, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press (2002)
22. Horrocks I, Patel-Schneider, P.F.: Comparing subsumption optimizations. In Franconi, E. De Giacomo, G. MacGregor, R.M., Nutt, W. Welty, C.A., Sebastiani, F. (eds), Collected Papers from the International Description Logics Workshop (DL'98), pages 90–94. CEUR, (1998).
23. Barbuceanu, M. and Fox, M.S. COOL: A language for describing coordination in multiagent systems. In Proc. First International Conference on Multi-Agent Systems, MIT Press, (1995), 17-24.
24. 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), 269-276
25. Minsky, N. and Ungureanu, V. Law-Governed Interaction: A Coordination & Control Mechanism for Heterogeneous Distributed Systems. in ACM Transactions on Software Engineering and Methodology (TOSEM) Vol 9.3, 273-305 (2000)

26. Fontoura, M., Ionesu, M. and Minsky N. Law-Governed Peer-to-Peer Auctions In Proc. of the eleventh international world wide web conference (WWW2002) Honolulu, Hawaii, May (2002)
27. Artikis A., Sergot M. and Pitt J. Specifying Electronic Societies with the Causal Calculator. In the Proceedings of the Agent-Oriented Software Engineering III (AOSE) workshop, LNCS 2585, Springer, (2003)
28. Reeves, D., Wellman, M. and Grosof, B. Automated Negotiation from Declarative Contract Descriptions. In Proc. Fifth International Conference on Autonomous Agents, (2001)
29. Ashri, R., Rahwan, I. and Luck, M.: Architectures for Negotiating Agents, in Mueller, M.V. Pechoucek, J.(eds). Multi-Agent Systems and Applications III, pages pp. 136-146. Springer, (2003)
30. Wurman, P, Wellman, M. and Walsh W.: A Parameterization of the Auction Design Space, in Games and Economic Behavior, 35 Vol. 1/2 (2001), 271-303