J Grid Computing (2009) 7:225–246
DOI 10.1007/s10723-009-9118-3

Negotiating SLAs-An Approach for a Generic Negotiation Framework for WS-Agreement

Sebastian Hudert · Heiko Ludwig · Guido Wirtz

Received: 15 April 2007 / Accepted: 2 April 2009 / Published online: 22 April 2009

Abstract The current Web Services Agreement specification draft proposes a simple request-response protocol for agreement creation only addressing bilateral offer exchanges. This paper proposes a framework augmenting this WS-Agreement to enable negotiations according to a variety of bilateral and multilateral negotiation protocols. The framework design is based on a thorough analysis of taxonomies for negotiations from the literature in order to allow for capturing a variety of different negotiation models within a single, WS-Agreement compatible, framework. In order to provide for the intended flexibility, the proposed protocol takes a two-stage approach: a meta-protocol is conducted among interested parties to agree on a common negotiation protocol first before the real negotiation is carried out in the second step due to the protocol established in the first step.

Keywords SLA · Negotiation protocols · WS Agreement

S. Hudert (✉)
Department of Information Systems Management, University of Bayreuth, Bayreuth, Germany
e-mail: sebastian.hudert@uni-bayreuth.de

H. Ludwig
IBM Research, T. J. Watson Research Center, Hawthorne, NY, USA

G. Wirtz
Distributed and Mobile Systems Group, Otto-Friedrich University Bamberg, Bamberg, Germany

1 Introduction

Managing quality of service (QoS) in loosely-coupled distributed systems such as computational Grids cannot rely on traditional, centralised management. Since parameters of systems in other domains cannot be manipulated, QoS guarantees must be obtained in the form of service level agreements (SLAs). SLAs represent qualitative guarantees placed on service invocations within a service oriented environment. Service consumers benefit from guarantees because they make non-functional properties of service predictable, often secured by a penalty. On the other hand, SLAs enable service providers to manage their capacity, knowing the expected requirements. By employing SLAs, a robust service oriented architecture can be realised, even across company boundaries. To support broad application, standards for the structure of agreement documents as well as a a standard process to establish and monitor them are required. Such protocols are particularly important if the agreement creation is to be executed automatically.

The Web Services Agreement (WS-Agreement) specification is a standardisation effort conducted by the Open Grid Forum (OGF) in order to facilitate creation and monitoring of SLAs [1]. This standard defines an XML-based structural definition of SLA documents, a simple request-response protocol for agreement creation as well as corresponding interfaces for agreement creation and monitoring. A WS-Agreement specifies functional properties and qualitative service level guarantees in a detailed way as a set of terms.

However, the proposed agreement creation process is restricted to a simple request-response protocol: one party (agreement initiator) creates an agreement document, possibly based on an agreement template, and proposes it to the other party (agreement responder). The agreement responder evaluates the offered agreement and assesses its resource situation before accepting or rejecting the offer. This protocol does not enable advanced negotiation formats involving numerous parties in different roles such as auctions. Enabling a variety of negotiation protocols would result in wider applicability of WS-Agreement for more demanding allocation problems.

The incorporation of different negotiation protocols into the agreement creation process of WS-Agreement poses several problems: First, such protocols must be integrated seamlessly in the overall WS-Agreement protocol to enable subsequent agreement monitoring, as defined in the WS-Agreement specification. Furthermore, in an automated negotiation, all participating components—here referred to as agents—must be aware of all rules and constraints concerning the negotiation protocol. Finally, a corresponding infrastructure of role definitions, interfaces and methods has to be presented to facilitate the actual negotiations.

To supply the negotiating agents with the necessary information to participate in the actual negotiation protocol a fixed, well known set of negotiation protocol definitions could be specified. Afterwards, during the actual negotiation the corresponding protocol description is simply referenced by all participants. However, this limits the set of available negotiation protocols to a predefined, finite set. As this would be a first step to make the negotiation process more flexible, it limits the range of protocols permitted to those foreseen at the time, the standard itself is fixed.

In this paper, we propose an even more generic approach in which parties in a distributed system agree on a negotiation mechanism first, then conduct the SLA negotiation and then fulfill the SLA. To this end, we define a meta-language for negotiation protocols. Using such a meta-language a multitude of specific negotiation protocols can be defined using a well-defined set of attributes and parameters. These protocol definitions are made available to all prospective negotiators before the actual negotiation to inform them which protocol has been chosen. Furthermore, we propose an exchange protocol to distribute the negotiation definitions to all prospective negotiators and choose a negotiation protocol. Finally we propose a generic negotiation protocol that is able to support all specific negotiation protocols that can be described with the presented negotiation attributes by simply carrying out the formerly established protocol according to its definition in the document the participants formerly have agreed on. This can be done as close to standard as possible, i.e., as an extension to basic WS-Agreement offers.

The rest of this paper is organized as follows: Section 2 discusses the negotiation models analyzed in order to define a flexible framework and introduces the basic concepts and data structures used in our framework. These are illustrated by a simple scenario in Section 3. Afterwards, in Sections 4 and 5, the underlying philosophy as well as the interfaces and building blocks offered for the exchange protocol and the negotiation protocol itself are introduced, respectively. Section 7 discusses some related work. We conclude the paper in Section 8 with a summary as well as some remarks on future work.

2 Basic Definitions and Data Structures

Before describing the exchange and negotiation protocols this section will give a short overview of the basic concepts and data structures used in the negotiation framework subsequently. In order to provide a sufficient background to render the decisions made more transparent, characteristical attributes found in negotiation taxonomies from the literature are discussed here, too.

2.1 Negotiation Protocol Definition

This framework supports a multitude of different negotiation protocols, like various auction types or one-on-one bargaining protocols. Each negotiation protocol that is to be conducted fully automated has to be exhaustively described. Only by providing a complete and machine-processable process description its correct application in automated distributed systems can be guaranteed. In order to enable such a protocol description a set of negotiation attributes has been identified as the basis for this framework. Employing these attributes a variety of different negotiation protocols can be specified in terms of a structured protocol description document. Such documents are subsequently distributed to all agents involved in a particular negotiation according to the exchange protocol described in the next section.

In order to specify a comprehensive set of negotiation attributes this framework employs negotiation taxonomies originating in e-commerce research and economics. These taxonomies present a set of parameters that allow for detailed description of specific negotiation protocols. The most important taxonomies for this paper will shortly be described in the following.

Alessio R. Lomuscio, Michael Wooldridge and Nicholas R. Jennings presented a quite comprehensive definition of negotiation parameters in [17]. They emphasize automated negotiations in electronic commerce (e-commerce) settings. Even though the scenario definitions and agent concepts used do not exactly fit the SLA settings our work is based on, because of the focus on pre-negotiation phases and high human interaction rates, they already show some of the issues that arise when automated negotiations take place. Such problems are formalisation of negotiations or implementation of negotiation strategies by agents themselves. One aspect, that makes their work very suitable as a basis for our framework is that the authors do not only concentrate on auctions, but on negotiations in general which fits our approach of creating a data structure for multiple negotiation protocols.

Peter Wurman, Michael Wellman and William Walsh defined a set of auction parameters while developing an internet-based "platform for price-based negotiation—the Michigan Internet AuctionBot" [36]. This system was designed to serve as an auction server for humans as well as software agents and only focused on one negotiated issue: the price. Therefore unfortunately only one-dimensional auctions were supported. The same authors extended their taxonomy to also cover multidimensional auctions in a follow-up, much more comprehensive paper [37]. Here, they do not only cover auction protocols and their parameters as a necessary byproduct of the development of an internet auction server as before, but focus especially on the different possible auction protocols and respective parameters. This approach allows for a much more comprehensive examination of the auction design space. The second improvement regarding our work is the already mentioned incorporation of multidimensional auctions. Since in a SLA environment the involved parties will mostly negotiate more than one desired aspect of a service, multidimensional auctions will be much more suitable than traditional auction protocols which allow to negotiate only the price.

The next taxonomy used in this work was proposed by Claudio Bartolini, Chris Preist and Nicholas R. Jennings. It has been developed in the context of their specification of a framework for automated negotiation in multi-agents systems (MAS) [4]. Thus, the authors focus on executable specifications for MAS. They concentrate on message formats used and activities conducted by software agents which represents a very technical and practical approach to negotiation research. Their work contributes to this framework as it assumes similar conditions as in automated SLA negotiations between software agents like those supported with our work.

The most detailed taxonomy used was published by Michael Ströbel and Christof Weinhardt [34]. It represents the most comprehensive categorisation and description of electronic negotiations found so far. In contrast to Wurman et al., Ströbel and Weinhardt aim for a more generic understanding of negotiations than just claiming negotiation design is essentially equivalent to auction design [37]. They regard auctions as one particular (sub)class of negotiations; this point of view was adopted in our work. The authors also do not stress software agent characteristics such as agent strategy or computational complexity like Lomuscio et al. do, because they consider "the degree of automation (...) as orthogonal to the classification criteria in [their] taxonomy" [34]. Thus the negotiation taxonomy presented by Ströbel and Weinhardt aims to describe and classify a multitude of negotiation protocols in a very generic, yet comprehensive way without stressing technology-related issues.

To ensure a solid foundation for our framework, the taxonomies discussed so far were integrated and consolidated in order to derive a set of attributes and corresponding domains that are especially suitable for SLA negotiation settings. Such scenarios inherently exhibit distinct characteristics, leading to distinct requirements for our framework:

- SLAs normally comprise more than just one attribute, therefore an SLA negotiation framework must focus on combinatorial negotiations.
- Each service referenced individually and therefore defines an individual item. Hence multi-unit negotiations focussing on homogenous goods to be traded are not appropriate in SLA negotiation settings.
- An SLA always governs one or more service invocations done by a service consumer. The service is offered by the service provider. Therefore SLA negotiations primarily focus on these two roles in a negotiation.
- In order to guarantee integrity of the negotiation a common concept in negotiation theory is a trusted third party governing the negotiation process. This is also appropriate for SLA environments in which service consumers and providers can utilise such a central service for discovery of potential negotiation partners as well as a infra-structural node used in the negotiation process itself.
- A special requirement posed on this framework is the need for fully automated negotiations. Negotiation protocol descriptions used in such fully automated scenarios must conform to a very strict structure and must be syntactically processable by software agents.

We consolidated the negotiation taxonomies found in the literature with respect to the above requirements; the result of which will be shortly sketched in the following. Due to its complexity a comprehensive description of our taxonomy along with the derived data model is outside of the scope of this paper. However, such a detailed definition can be found in [14], comprising the taxonomy of negotiation parameters, a respective data model (formalised as an Entity Relationship Diagram [7], as well as an Extensible Markup Language based representation of this data model for seamless integration with the WS-Agreement specification. We identified the following high level attribute categories:

1. General Negotiation Process
2. Negotiation Context
3. Negotiated Issues
4. Offer Submission
5. Offer Allocation
6. Information Processing

The General Negotiation Process attributes abstractly define the overall negotiation process. This includes for example the starting and termination rules for a negotiation, the number of rounds or whether or not the negotiation protocol is rewarding protocol compliance or punishing protocol violation (employing reputation concepts).

The Negotiation Context groups attributes concerning the agent-related aspects of a negotiation. This includes for example the definition of roles and the admission rules for each role. In terms of negotiation theory this category defines the negotiation's configuration.

The attributes making up the Negotiated Issues category define the values of the SLA to be negotiated. This category therefore defines for example. which attributes of a service are subject to the negotiation, or if the negotiators are allowed to extend the set of negotiated attributes of a service, for example to request an additional quality guarantee.

Offer Submission parameters govern the bidding process. Rules concerning the submission of bids or the relation between bids are specified with these attributes defining which roles are allowed to post bids, under which conditions a bid can be posted or whether a valid bid has to fulfill some constraint. This is used in English Auctions for example to restrict bids to be superior to the last valid bid.

The Offer Allocation category attributes govern the matching process of a negotiation, more precisely the agreement formation in SLA scenarios.

The accessible information concerning the current status of the negotiation, past offers from participating agents and the permission to access this data is handled with Information Processing attributes. This category therefore defines whether agents can react on other agents' actions by defining what information can be accessed by which agent, allowing to define sealed-bid or open-cry negotiations respectively.

We defined all attributes to exhibit one of three possible value domains:

1. A given data type, such as Integer or Boolean.
2. An explicitly defined value domain, such as an enumeration of possible values.
3. Unrestricted, meaning that any given rule-based language can be used for expressing the respective attribute value. This way complex attributes, such as offer submission restrictions, can benefit from the flexibility and power of external languages, such as e. g. Jess [12].

Based on these attributes a multitude of 1:1 and 1:n negotiation protocols can be defined as detailed as is necessary for automated execution.

2.2 Negotiation Types and Instances

For understanding our approach, the distinction between negotiation types and instances is essential. Analogously to types and instances in object-oriented programming languages, negotiation types describe general classes of negotiations and define their common attributes and elements. A negotiation instance, in contrast, stands for one particular negotiation process of some type that can be unambiguously referenced. For example, a negotiation type may define, that there is one agent involved not allowed to post offers himself, whereas on the other side n agents can participate by posting offers, in which every offer has to beat the last posted offer by some amount and so on. This roughly describes some type of auction. One instance of this negotiation type therefore represents one particular auction process.

Regarding their content, negotiation type and negotiation instance documents differ slightly. The main difference is that a negotiation type does not contain an identifier that is used to refer to a given negotiation process; such an identifier identifies a particular negotiation instance and is therefore only present in the respective instance description. The majority of the attributes identified in the previous section, however, have to be initialized when defining a negotiation type for they describe types of negotiations and not individual negotiation processes.

There are only three exceptions; three attributes do not necessarily have to be defined in a negotiation type document: the start and the termination and the agents involved in a negotiation instance.

The start and the termination of a negotiation can be set in the negotiation type or in the negotiation instance document according to the same rule. Both attributes can be defined as assertions over time or as arbitrary constraint expressions not concerning some time values. A negotiation's starting rule can for example define the negotiation to start at August 28th 2008 at 11:00pm (Central European Time). This would be a time-based assertion. On the other hand a negotiation can be defined to start whenever the trusted third party governing the negotiation sends a respective message to / invokes a corresponding method on the participants; this being a constraint expression not concerning some time values. If a negotiation start or termination is specified in terms of time points, this has to be done in the negotiation instance document since such time points are inherently different for each instance. When defined as a constraint expression over other parameters it has the starting and termination rules are defined in the negotiation type since those rules apply to all negotiations of this type equally.

The involved agents are not specified in a negotiation type document because the creation of a negotiation type should be independent from the actual negotiation process, within which the negotiators take part. Not having to know all agents when defining a negotiation type highly increases flexibility as it allows for definition of generic negotiation type documents that can be stored a some repository server. Such a document would already define all the negotiation protocol's attributes, except for the involved participants.

These negotiating agents regularly change over time and therefore would almost never be exactly the same for two instances of a given negotiation type. Hence having to state the set of potential negotiators in the negotiation type would prevent our framework from being able to define such independent type documents that are flexibly instantiated when needed.

During the exchange protocol described in the next sections other agents subsequently join this negotiation and have to be incorporated into the data structure representing the actual negotiation instance subsequently.

In order to supply the negotiating agents with the required information about negotiation types and instances, two Extensible Markup Language (XML) document descriptions (formalised as XML-Schema documents) are used within the exchange and negotiation protocols as described in the next sections.

A detailed description of the XML-Schema documents as well as two extensive examples of these schema documents (an auction and a one-on-one bargaining protocol) can be found at [14].

2.3 Abstract Architecture of Negotiation Documents

The main negotiation object is a WS-Agreement template with its corresponding creation constraints as defined in the current WS-Agreement specification [3]. Since this framework augments the current specification with possibilities to negotiate over a WS-Agreement this fundamental data structure is adopted for the (partial) definition of some service(s) to be negotiated. The creation constraints as part of this template are also used in this approach to give syntactical restrictions on the elements still to be initialised or to be altered during the negotiation. A WS-Agreement template, some negotiation is defined upon, can uniquely be referenced by the templateID, unique for a given template at a distinct endpoint, and optionally the EPR for the service offering this template.

The negotiation type document refers to the WS-Agreement template the negotiation is defined upon and specifies the negotiation attributes as given by our taxonomy. Given its content the negotiation type document defines which terms of a WS-Agreement can be negotiated and how to do so.

A concrete negotiation is represented by a negotiation instance document. This document refers to the negotiation's type, its participants, specifies a unique identifier and optionally starting and/or termination rules as already stated above.

Finally, the result of the complete negotiation protocol is a valid WS-Agreement document satisfying the creation constraints as defined in the initial WS-Agreement template referenced in the negotiation type document.

2.4 Involved Roles

In order to define the different parties involved in a negotiation process as well as their expected behavior, three distinct roles are used in our framework, namely Negotiation Participant, Negotiation Coordinator and Information Service. Since this framework is employed in service oriented environments each of these roles offers some functionality as a service to the other agents involved in the negotiation.

A Negotiation Participant represents an agent participating in the initial exchange protocol (used to distribute the negotiation documents to the prospective negotiators) and the negotiation process itself. In terms of service oriented environments the service consumers and providers make up such Negotiation Participants.

The Negotiation Coordinator is a logically centralised instance which handles admission of agents to a given negotiation as well as (re-) distribution of the negotiation documents to the prospective negotiators.

The information distribution during the actual negotiation is governed by the Information Service. This service offers information about the current status of a negotiation (for example the currently highest bid) or about the offer history to the requesting agents.

The basic guidelines, data structures and roles presented here, will be made more concrete in the next sections by giving an example scenario for the exchange and negotiation protocols.

3 Example Scenario

The following scenario provides a typical application of our framework (see Fig. 1): In a given Grid system the individual nodes offer services to each other. Each of these services can be offered at different quality levels which have to be defined for each service invocation. Such service quality assertions are expressed as SLAs, using the WS-Agreement standard. In order to allow for flexible

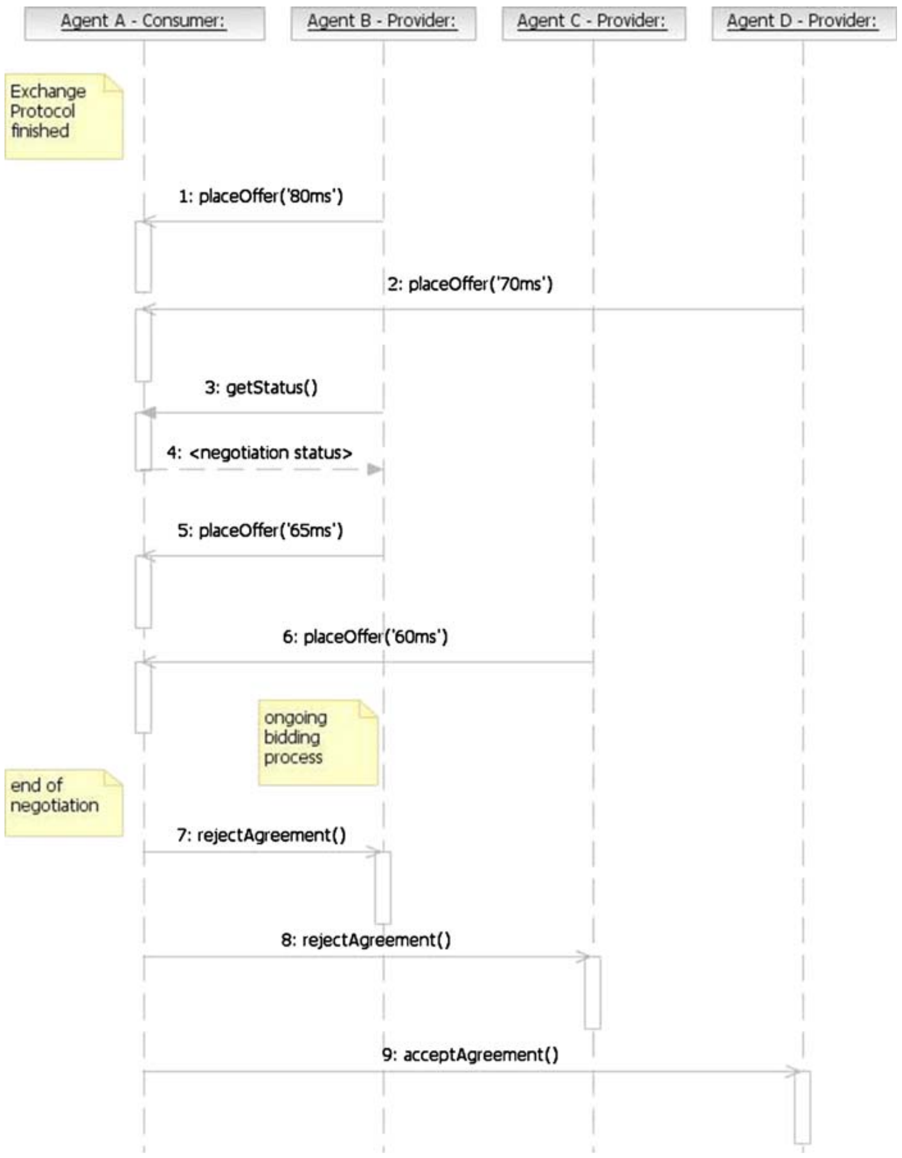

Fig. 1 Scenario—example auction

negotiations of SLAs, each Grid node is associated with a software agent or some similar intelligent service used for the negotiation process, that is offering the needed negotiation interfaces as proposed by our approach. In our scenario Grid service A (represented by software agent A) requests some service which is offered from several nodes in the Grid, for example a currency service offering methods to convert currency exchange rates and to query historical exchange rates analogously to a stock exchange chart. Agent A furthermore knows that this service is offered by several other nodes in the Grid, although at different service quality levels.

In our scenario, service A uses the currency service within more complex Grid based work flows ultimately generating financial reports to its customers, service A potentially submits a big batch of currency conversions during a given time span. Hence response time is considered to be the most important quality parameter in our scenario; a preferably small response time of the currency service is desired. For this purpose agent A decides to conduct an auction-like protocol to negotiate the final SLA. In this auction the only parameter to be negotiated is the response time of the currency service. The agent bidding the lowest guaranteedResponseTime wins the auction, starting off at a response time of 100 ms which is the maximum agent A can accept. A typical interaction stemming from this setting is shown in Fig. 1.

Table 1 Auction type description for usage scenario

```
<negotiation ...
    xsi:schemaLocation='... NegotiationType.xsd'>
    <negotiationTypeID>
        currencyServiceAuctionType
    </negotiationTypeID>
    <wsAgreementTemplate>
        <endpoint>
            http://www.serviceA.com/currency
        </endpoint>
        <templateID>
            currencyServiceTemplate
        </templateID>
    </wsAgreementTemplate>
    <!-- start and termination is set in the negotiation instance document -->
    ...
    <role roleName="serviceProvider" permissionToPostOffers="true">
        <admissionRestriction admissionRestrictionForm="open"/>
    </role>
    <role roleName="serviceConsumer" permissionToPostOffers="false">
        <maximumNumberOfAgents>
            1
        </maximumNumberOfAgents>
        <admissionRestriction admissionRestrictionForm="open"/>
    </role>
    ...
    <negotiatedIssues>
        <guaranteeTerms extendable="false">
            <guaranteeTermID domain="xsd:integer" values="single">
                guaranteedResponseTime
            </guaranteeTermID>
        </guaranteeTerms>
    </negotiatedIssues>
    <attributeRestriction>
        <attribute>
            guaranteedResponseTime
        </attribute>
        <restriction>
            <threshold>
                <upperBound>
                    100
                </upperBound>
            </threshold>
        </restriction>
    </attributeRestriction>
    ...
    </negotiation>
```

For this purpose agent A creates a WS-Agreement template document according to the current specification draft [3]. This document states the expiration of the resulting agreement, the currency service's functionality in the service description terms as well as the response time as a service property term to be used in the service quality assertion. In the service quality terms 100 ms is set for the `guarantee ResponseTime` parameter, stating the upper bound for this attribute in the subsequent negotiation.

After having defined the template document, agent A creates a protocol description as proposed in this framework. This protocol description references the template document just created (by stating the endpoint reference where agent A offers its services and the template's ID) to provide the prospective negotiators with the link to the template and therefore to the syntactical restrictions of the SLA to be derived. Furthermore it states that only one agent is allowed to join on the consumer side of the SLA, which will be agent A itself, and possibly n different agents can join the negotiation as providers. The only issue to be negotiated is the `guaranteedResponseTime` as already stated.

Table 1 shows some excerpts of the XML representation of the negotiation type definition as just described Section 2.3: besides referencing an WS-Agreement template, two roles, i.e. `serviceProvider` and `serviceConsumer` are introduced as well as the `guaranteed ResponseTime` issue to be negotiated with a restrictive upper bound.

In contrast to the negotiation type definition, Table 2 presents some aspects of an negotiation instance that references this auction type: concrete start and end dates are given as well as the agents involved.

Table 2 Auction instance description for scenario

```
<NegotiationInstance ...
    xsi:schemaLocation="... NegotiationInstance.
    xsd">
    <negotiationID>
        currencyServiceAuction
    </negotiationID>
    <negotiationType>
        <referencedNegotiationType>
            <endpoint>
                http://www.serviceA.com/currency
            </endpoint>
            <negotiationTypeID>
                currencyServiceAuctionType
            </negotiationTypeID>
        </referencedNegotiationType>
    </negotiationType>
    <start>
        2006-09-30T13:30:00
    </start>
    <termination>
        2006-09-30T23:30:00
    </termination>
    <agent>
        <role>
            serviceConsumer
        </role>
        <agentEPR>
            http://www.serviceA.com/currency
        </agentEPR>
    </agent>
    <agent>
        <role>
            coordinator
        </role>
        <agentEPR>
            http://www.serviceA.com/currency
        </agentEPR>
    </agent>
    <agent>
        <role>
            informationService
        </role>
        <agentEPR>
            http://www.serviceA.com/currency
        </agentEPR>
    </agent>
</NegotiationInstance>
```

4 Exchange Protocol

Our framework supports the complete process of agreement creation. As depicted in Fig. 2, this creation process is divided into three distinct phases: first the initially created negotiation protocol definition has to be distributed to all prospective negotiators. This process is described with the exchange protocol as defined in this section. Subsequently the actual negotiation process takes place, according to the rules defined and distributed in the previous phase. Such a generic negotiation protocol is presented in the next section. Finally, in the agreement acceptance phase one offered agreement is accepted by one of the participants

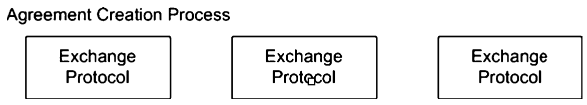

Fig. 2 Agreement creation process

to conclude the negotiation. Alternatively, there may be no acceptable offer and the negotiation is terminated by rejecting all offers.

Although not actually part of the exchange or the negotiation protocol the overall process consists of one additional phase: the creation of the negotiation protocol description, which takes place before the other three stages of agreement creation. During this phase a negotiation protocol instance is created which defines the rules for the following WS-Agreement negotiation.

Since the approach for conduction negotiations is intended for a Web Services setting, the exchange protocol will not focus on exchanged messages primarily, but on the provided services and respective methods and method sequences to be invoked subsequently.

The methods needed for the different negotiation steps within the protocol are structured in interfaces according to the different roles as already outlined in the previous section. Using these roles and respective interface methods, a range of different scenarios within the exchange process can be realised.

4.1 Interfaces for the Roles Involved

In order to conduct a negotiation, the involved agents are assumed to have agreed upon a particular negotiation instance. Such an instance is uniquely described with an negotiation instance document, a data structure containing a unique identifier for this instance, a reference to the negotiation type, the list of participating agents and optionally time-based start and termination parameters. This follows the instantiation concept presented in Section 2. Every time an agent joins an already instantiated negotiation, the data structure is updated, by adding this agent to the role it adopts, and redistributed to all participants of the negotiation that are already known. At the end of the meta data exchange process thus every involved agent is aware of the start and termination of the negotiation, its type and the agents currently involved. This very general protocol description already shows which of the roles already introduced are involved in the exchange process: a centralised Negotiation Coordinator interacting with two or more Negotiation Participants.

In the following, we discuss the interface functionality of both roles involved in a bit more detail. A complete description of all interfaces involving the corresponding WSDL documents can be found in [14].

Since the Negotiation Coordinator provides negotiation protocol descriptions and handles the admission of participating agents to a given negotiation, the corresponding interface offers a set of query methods for participants that are used for requesting available negotiation type and instance documents.

1. *getAllNegotiationTypes()/getAllNegotiationTypesForTemplate(...)*
2. *getCurrentNegotiations()/getCurrentNegotiationsForTemplate(...)*

Thus, very general queries (requesting available negotiation types) are possible as well as queries concerning negotiations currently active. The second dimension, indicated by '...ForTemplate', allows for querying negotiation documents for a particular WS Agreement template the negotiation is defined upon.

Besides simply asking to join an already running process, an agent may actively propose a negotiation instance to a coordinating agent.

1. *joinNegotiation(negotiationID, agentEPR, 'credentials')*
2. *proposeNegotiation(NegotiationInstance-document)*
3. *publishNegotiation(NegotiationInstance-document)*
4. *publishNegotiationToReceipients(..., [receipients])*

Publishing a negotiation differs from the *proposeNegotiation()*-method in that it is not assumed that the coordinator used for publishing also is to act as Negotiation Coordinator of the respective negotiation. It only offers this negotiation instance for look-up purposes while the actual admission and information (re)distribution tasks are conducted by the actual coordinating agent, probably the one publishing the negotiation instance. This method can be used to implement systems of distributed look-up servers.

The *publishNegotiationToReceipients()*-method is more specific as the agents that should be actively notified of this negotiation are explicitly named. In the more generic method the publishing agent cannot specify to which agents the negotiation should be published or whether this negotiation should be published push-style with the *proposeNegotiation()*-method at all.

Processing admission of agents at one logical centralised coordinator service eases the integration of reputation or security related external systems involved in the admission process. This way most of the consistency problems arising when operating distributed systems can be solved in a centralised way. All agents joining a negotiation do so by invoking the corresponding method on the central coordinator service which handles the whole admission process. Another coordinator task is therefore to notify the participating agents when others have joined by posting the updated negotiation instance document to them as described above.

The second role needed in the exchange protocol is the one of a regular participant. This role is adopted by all agents actively participating in a negotiation, i.e., by service providers as well as consumers. All these agents have to offer some methods to enable negotiations.

The Negotiation Participant role, however, is present in both, the exchange and the negotiation protocol. In order to describe the offered methods in a consistent way, the methods used for the exchange protocol are described here while the ones used in the actual negotiation will be sketched in the context of the negotiation protocol (see next section).

As described in the context of the coordinator already, the coordinator needs a handle to provide participants with up-to-date information about a negotiation. This is used when new agents have joined the negotiation and the updated instance document has to be promoted to all Negotiation Participants.

1. *updateNegotiation(NegotiationInstance-document)*
2. *proposeNegotiation(NegotiationInstance-document)*
3. *acceptNegotiation(negotiationID)*

On the other hand, it should be possible for a Negotiation Coordinator to propose a negotiation instance to a (potential) Negotiation Participant, as already described. As opposed to the corresponding method of the Negotiation Coordinator interface, this method proposes a negotiation to agents to act as regular participants in the resulting negotiation.

The *acceptNegotiation()*-method is offered as a counterpart for the *proposeNegotiation*-method to support asynchronous communication. When a negotiation is proposed to a Negotiation Coordinator this agent can decide whether to coordinate this negotiation or not. If it decides to do so, it invokes the *acceptNegotiation()*-method, if not a timeout in the proposing agent occurs showing this agent that the desired Negotiation Coordinator will not govern the proposed negotiation. The proposing agent can subsequently propose this negotiation to another known coordinating service.

4.2 Protocol Components

Although only providing two distinct roles, the exchange protocol provides a broad support for different exchange processes. A multitude of possible protocols can be constructed from only three basic logical protocol components: request for and proposal of negotiation data as well as their mediated exchange. During the first two of which one agent acts as a coordinator and a participant and the last of which marks the situation where an independent Negotiation Coordinator is present.

4.2.1 Request for Negotiation Documents

This step describes the process of one agent requesting negotiation type or instance documents from the respective Negotiation Coordinator. The corresponding request-methods are defined in the Negotiation Coordinator interface (see Section 4.1) for the coordinating agent stores and (re)distributes this information.

The distinction between types and instances allows agents to request actually instantiated negotiations, that are already running or that are about to start, as well as supported negotiation types in general. After requesting general types an agent can propose a concrete instance of a specific type to the coordinator agent in order to trigger the instantiation of a new negotiation.

There are several possible ways to request negotiation data. An agent may, for example, query all negotiation types supported by the respective Negotiation Coordinator using the *getAllNegotiationTypes()*-method. This allows the Negotiation Coordinator to generally define the supported negotiation protocols without instantiating one particular negotiation. The coordinator agent can thus wait until other agents have requested the types that are available and propose a particular type to be instantiated.

On the other hand, agents may query already instantiated negotiations with the *getCurrent Negotiations()*. As a result to such a query for negotiation instances, the Negotiation Coordinator returns a list of negotiation instance documents describing the currently available negotiation instances.

Analogously to requesting all available negotiation types or instances agents may also query only types and instances defined for a given WS-Agreement template, identified with an endpoint reference (EPR) referencing the service offering the template and a templateID identifying the particular template within the set of available ones at this endpoint. Hence this method provides agents with a means to inspect all possible negotiation types available for some service they already know.

If only the possible negotiation types or current negotiations are to be queried, the exchange protocol only consists of one method invocation and respective return parameters. If one of the returned negotiation types or instances is appealing for the requesting agent and it wishes to take part in the respective negotiation, the involved agents have to conduct an additional step:

In case of the request for negotiation types an agent can identify a negotiation type it wishes to instantiate and propose the created instance to the coordinator by invoking the *proposeNegotiation()*-method.

If a negotiation is proposed to the Negotiation Coordinator, the proposing agent is supposed to

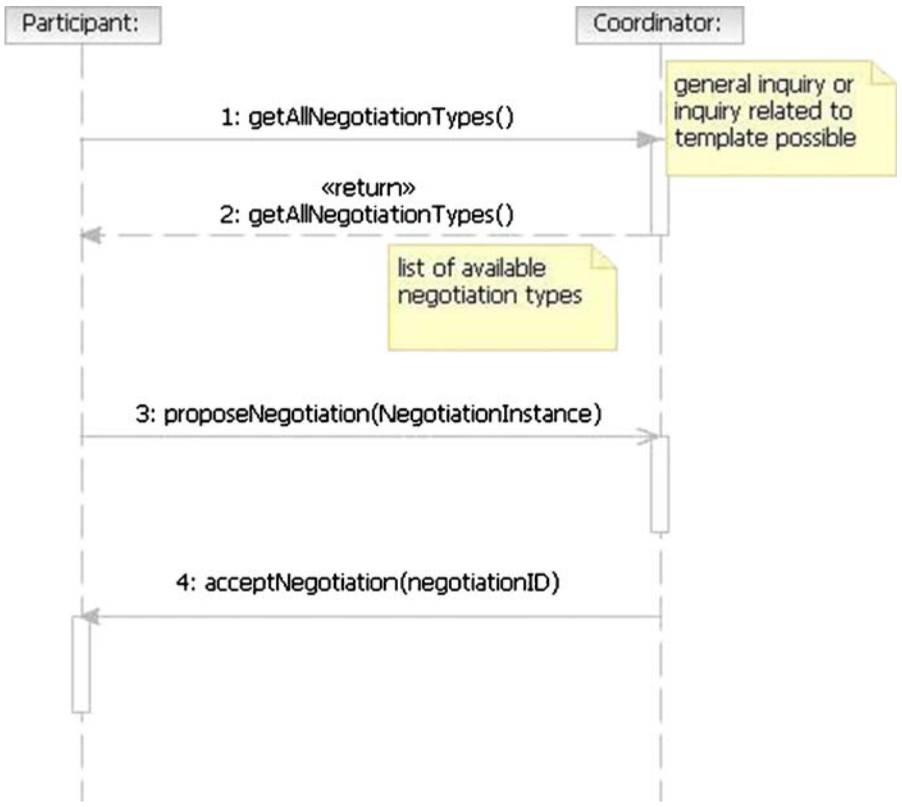
Participant:
Coordinator:
1: getAllNegotiationTypes()
general inquiry or inquiry related to template possible
«return»
2: getAllNegotiationTypes()
list of available negotiation types
3: proposeNegotiation(NegotiationInstance)
4: acceptNegotiation(negotiationID)

Fig. 3 Process of requesting available negotiation types and instantiating a negotiation

act as coordinator during the respective negotiation. Because the participant proposing this negotiation instance has to know whether it was accepted or not, the *acceptNegotiation()*-method is used to inform asynchronously (much like a callback) about the coordinator's decision. Asynchronous communication was considered useful here because this concept allows a (potential) coordinating agent to check its current resource situation before accepting a request.

In the process visualized in Fig. 3, the Negotiation Coordinator is proposed a negotiation instance document. It subsequently checks its resource situation for deciding whether to accept such a negotiation or not. The diagram depicts the situation where the Negotiation Coordinator accepts. The proposing agent can join this negotiation instance by invoking the *joinNegotiation()*-method afterwards.

If an agent wants to join an already instantiated negotiation (queried before), the *proposeNegotiation()*-step is omitted. The agent requests the currently available negotiation instances first, chooses an appropriate one and invokes the *joinNegotiation()*-method on the coordinator afterwards. This situation and the resulting steps are shown in Fig. 4.

4.2.2 Proposal of Negotiation Documents

This step represents the process of actively proposing some instance document to a prospective participant or coordinator. When proposed to a coordinator this agent is set as Negotiation Coordinator in the instance document. This way negotiations can either be proposed to agents simply taking part in or to some agent coordinating the subsequent bidding process. The protocol

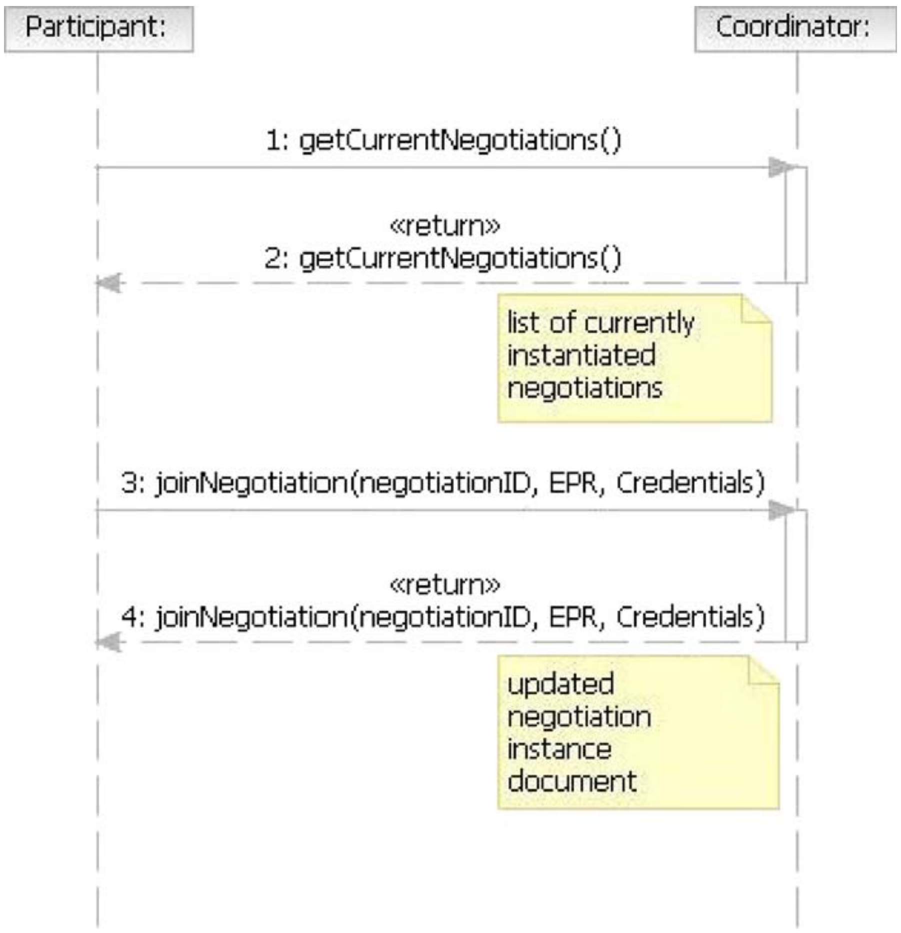
Participant:
Coordinator:
1: getCurrentNegotiations()
«return»
2: getCurrentNegotiations()
list of currently instantiated negotiations
3: joinNegotiation(negotiationID, EPR, Credentials)
«return»
4: joinNegotiation(negotiationID, EPR, Credentials)
updated negotiation instance document

Fig. 4 Requesting available negotiation instances and joining of the requesting agent

component regularly follows a request for negotiation types in order to propose the newly created instance to the coordinating agent. The details complement the process of querying information and are carried out by the interface methods already discussed in Section 4.1.

4.2.3 Mediated Exchange Processes

This third building block offers publish/subscribe functionality to the participants. Agents can publish negotiation instances at some Negotiation Coordinator to make it available to a larger community of prospective negotiation participants.

As already discussed, a Negotiation Coordinator does not have to join the negotiation as a service provider or consumer, but may act as a third party only responsible for administrative tasks. This concept enables the specification of centralised look-up servers only distributing negotiation data without taking part in any of these negotiations. These coordinators hence act as mediating third parties within the exchange protocol.

To enable publish/subscribe-like functionality agents should be able to publish negotiation instances to such look-up servers to promote their desired negotiation protocol. On the other hand agents requesting available protocols should be able to query these submitted protocols and search for appropriate ones.

For this purpose the Negotiation Coordinator offers the *publishNegotiation()* method. This method allows for publication of instantiated negotiations at some coordinating service. The other agents requesting the available protocols again query these by invoking the already introduced request-methods. As described before, the Negotiation Coordinator can also actively suggest negotiation instances to other agents using the *proposeNegotiation()*-method.

The diagram in Fig. 5 shows an exchange process where an agent A proposes a negotiation instance to the coordinator that proposes

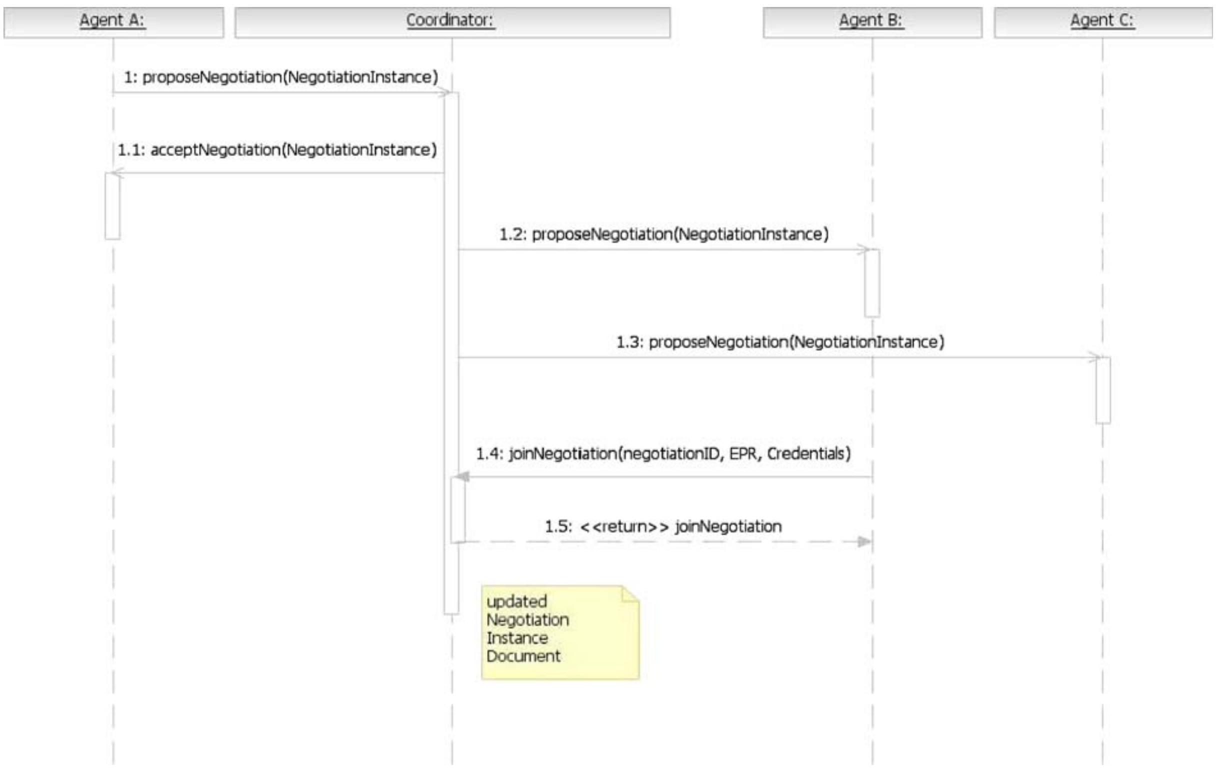
Agent A:
Coordinator:
Agent B:
Agent C:
1: proposeNegotiation(NegotiationInstance)
1.1: acceptNegotiation(NegotiationInstance)
1.2: proposeNegotiation(NegotiationInstance)
1.3: proposeNegotiation(NegotiationInstance)
1.4: joinNegotiation(negotiationID, EPR, Credentials)
1.5: <<return>> joinNegotiation
updated Negotiation Instance Document

Fig. 5 Mediated exchange process

this negotiation to two different agents B and C of which only agent B joins. Of course, this is just a fragment of the complete exchange process because certainly the coordinator would propose this negotiation to much more other agents and also other agents would request this document respectively.

By combining these three basic protocol components as building blocks, a multitude of different exchange processes can be specified, all resulting in distributing the information, needed to participate in a particular negotiation, to all prospective participants.

5 Negotiation Protocol

After supplying all negotiation participants with the negotiation type and instance documents the actual negotiation can start. The protocol governing this process is described in this section. Using this protocol the different negotiation types that can be specified with the presented data structure can be executed.

In general, we describe every negotiation as a bidding process. Each party involved in a negotiation offers an agreement to the other party concerning the issues subject to the negotiation that is currently acceptable for them. Then the other party assesses the offered agreement and generates a counter-offer, accepts the offer or rejects it and terminates the negotiation. This way the two parties involved move from a conflict situation concerning some (logical) resource(s) to a consensus represented by the resulting agreement.

Although only two sides (service provider and consumer) of a negotiation are present, a multitude of different protocols can be defined. These range from One-on-One Bargaining to various auction settings, all of which can be defined using the presented negotiation data structures and are provided by the documents introduced in Section 2.2.

Based on these documents the generic negotiation protocol introduced now has to provide the agents with means to post offers and to promote the decision made about a concrete offer.

The two roles present within the actual negotiation protocol are the Negotiation Participant and the Information Service. Analogously to the exchange protocol presented in Section 4, the negotiation protocol is defined in terms of roles and their respective interface methods to be compatible to the target service-oriented environments.

5.1 Protocol Sketch

As the Negotiation Participant represents a regular participant of a given negotiation process, it offers the following methods in the context of the negotiation protocol in addition to the methods needed for the exchange protocol (see Section 4.1). First of all, a participant has to be able to place offers in the context of a negotiation. Such an offer consists of an endpoint reference (EPR) of the posting agent and a complete WS-Agreement document representing the offered SLA.

1. placeOffer(agentEPR, WS-Agreement-document)
2. acceptAgreement(negotiationID, agreementID)
3. rejectAgreement(negotiationID)
4. startNegotiation(negotiationID)

In order to promote a negotiation's outcome to all participants, the *accept* and *reject*-methods are used. In the positive case, the winner of the negotiation and therefore the agent involved in the resulting agreement is notified by invoking this method. Since each agent could be involved in multiple negotiations, the id of the negotiation instance is given as a parameter along with the id of the accepted agreement offer. In case of disagreement, the *rejectAgreement* is invoked on all agents that did not win in the negotiation. As already sketched a method for starting the actual negotiation is also defined.

The Information Service role provides access to information on the current negotiation status or past offers. Hence the corresponding interface provides the following methods:

1. getStatus(negotiationID)
2. getPastOffers(negotiationID)/ getPastOffers(negotiationID, agentID)

The *getStatus*-method is used by all negotiation participants to access the current negotiation status. This allows, for example, to assess which offer is currently winning the negotiation and if necessary to adopt the own offer. It results in a data structure containing the current negotiation status, which is denoted by all current offers of all parties allowed to post offers. Note, however, that the currently winning bid may of course only be identified if the offer matching rules of this negotiation are given in the negotiation type document, otherwise the requesting agent can not anticipate the current winner.

The remaining methods let participating agents access all past offers of a negotiation. This information can be used for internal decision making of the negotiating agents. Such a request may be restricted to offers posted by as specific agent denoted by its ID as an additional parameter.

Note: Currently only a polling mechanism is available for accessing negotiation related information. We are currently exploring other concepts of information distribution, such as publish/ subscribe and notification functionality for a more

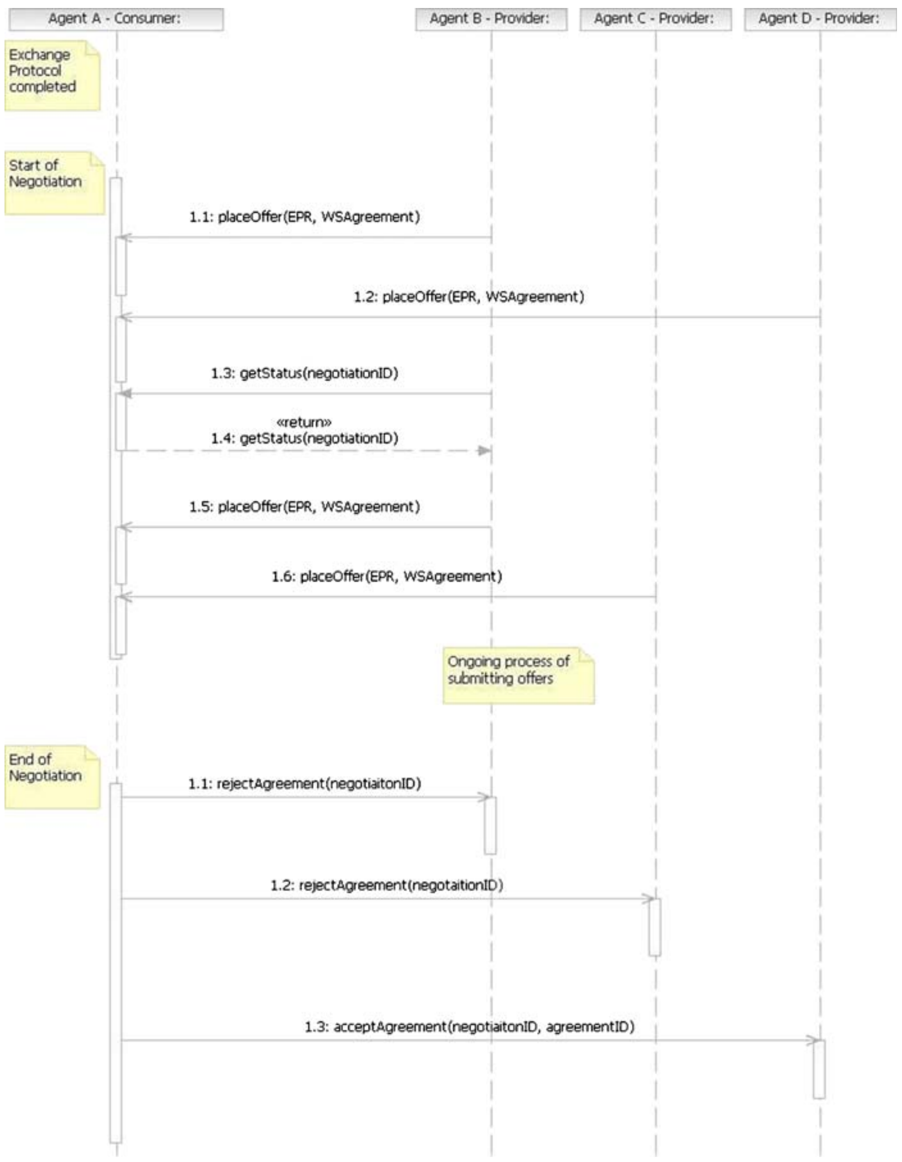
Agent A - Consumer:
Agent B - Provider:
Agent C - Provider:
Agent D - Provider:
Exchange Protocol completed
Start of Negotiation
1.1: placeOffer(EPR, WSAgreement)
1.2: placeOffer(EPR, WSAgreement)
1.3: getStatus(negotiationID)
«return» 1.4: getStatus(negotiationID)
1.5: placeOffer(EPR, WSAgreement)
1.6: placeOffer(EPR, WSAgreement)
Ongoing process of submitting offers
End of Negotiation
1.1: rejectAgreement(negotiaitonID)
1.2: rejectAgreement(negotaitionID)
1.3: acceptAgreement(negotiaitonID, agreementID)

Fig. 6 Sample auction process

flexible information processing mechanism. The InformationService would have to be extended by several methods to allow such a functionality.

5.2 Example Negotiation

In the rest of this section, we will illustrate our approach by describing an auction combined from the interfaces just presented Fig. 6:

In this setting agent A acts as service consumer requesting offers from different service providers. Agents B, C and D represent those service providers posting offers to agent A.

Under the assumption that the exchange process is already completed the actual negotiation starts as defined in the corresponding start-rule. After the negotiation started the bidding process takes place. Agents B, C and D subsequently post offers to agent A. This is depicted in the diagram by explicitly showing the submission of offers by each of these agents. As also hinted in the diagram this bidding process will go on for some amount of time resulting in much more offer postings than actually shown in the picture.

After the negotiation is terminated its result is communicated to all participants. In this case agent D offered the best agreement of all bidding agents and therefore wins this auction. Agent A subsequently promotes the result to all participants by invoking the *acceptAgreement()*- or *rejectAgreement()*-methods, respectively.

As a result of this negotiation agents A and D engage in an agreement with each other, whereas agents B and C do not reach an agreement.

Even though this negotiation process only shows a very simplified auction because of scope reasons it already sketches how even more complex negotiations can be conducted using the roles and respective methods defined above.

The diagram in Fig. 6 also depicts the information processing component of a negotiation. By retrieving the negotiation status from the Information Service (in this case also offered by agent A) agent B realises that agent D posted an offer exceeding its initial offer. If the negotiation would end at that point agent D would engage in an agreement with agent A and the other participants would not reach an agreement. As a reaction to this agent B creates another, better offer and posts

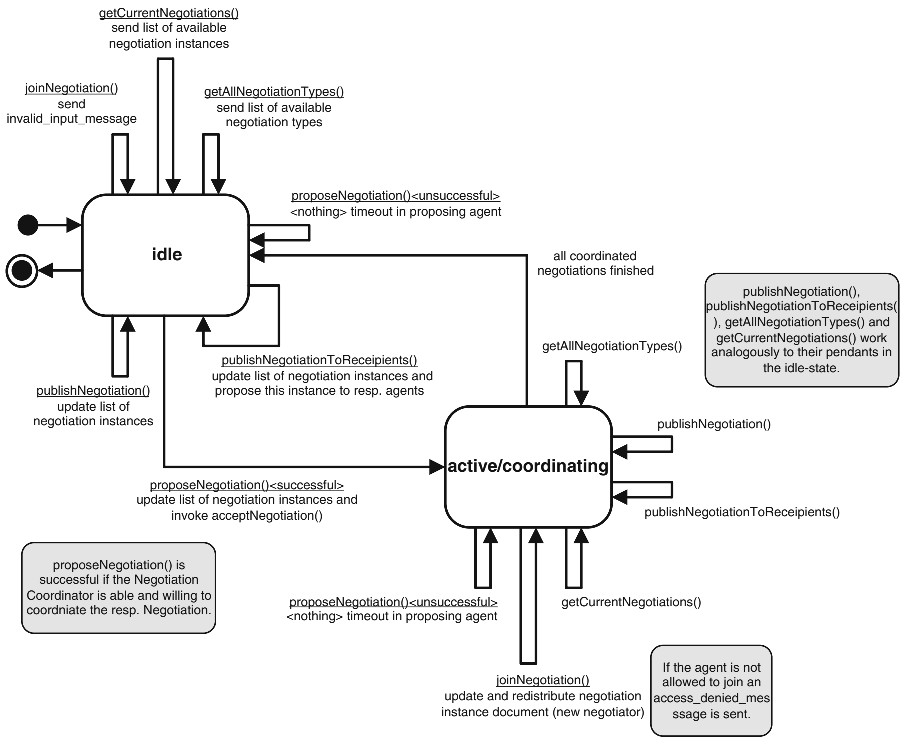
getCurrentNegotiations() send list of available negotiation instances
joinNegotiation() send invalid_input_message
getAllNegotiationTypes() send list of available negotiation types
proposeNegotiation()<unsuccessful> <nothing> timeout in proposing agent
idle
all coordinated negotiations finished
publishNegotiation(), publishNegotiationToReceipients(), getAllNegotiationTypes() and getCurrentNegotiations() work analogously to their pendants in the idle-state.
getAllNegotiationTypes()
publishNegotiationToReceipients() update list of negotiation instances and propose this instance to resp. agents
publishNegotiation() update list of negotiation instances
publishNegotiation()
active/coordinating
proposeNegotiation()<successful> update list of negotiation instances and invoke acceptNegotiation()
publishNegotiationToReceipients()
proposeNegotiation() is successful if the Negotiation Coordinator is able and willing to coordniate the resp. Negotiation.
proposeNegotiation()<unsuccessful> <nothing> timeout in proposing agent
getCurrentNegotiations()
joinNegotiation() update and redistribute negotiation instance document (new negotiator)
If the agent is not allowed to join an access_denied_messsage is sent.

Fig. 7 Negotiation coordinator—internal states

it to agent A to succeed the formerly winning offer. However, after an ongoing process of offer submissions agent D still wins the negotiation in the process described in this example.

6 State Diagrams of the Involved Roles

In the following the permitted internal states and state transitions for each of the three roles (Negotiation Coordinator, Negotiation Participant and InformationService) will be described. This acts as a theoretical validation of the consistency of our overall design and provides the first step in implementing this framework.

6.1 Negotiation Coordinator

Figure 7 depicts the internal states present for the Negotiation Coordinator role. There do exist only two states: *idle* and *active/coordinating*. A Negotiation Coordinator is only active if it is assigned to coordinate at least one particular negotiation instance. Thus a coordinating agent changes its state from *idle* to *active* if a negotiation stating this agent to be Negotiation Coordinator is proposed to it and the agent accepts this proposal.

An agent can coordinate more than just one negotiation. That is why the internal state changes from *active* to *idle* only when all coordinated negotiation instances have been finished.

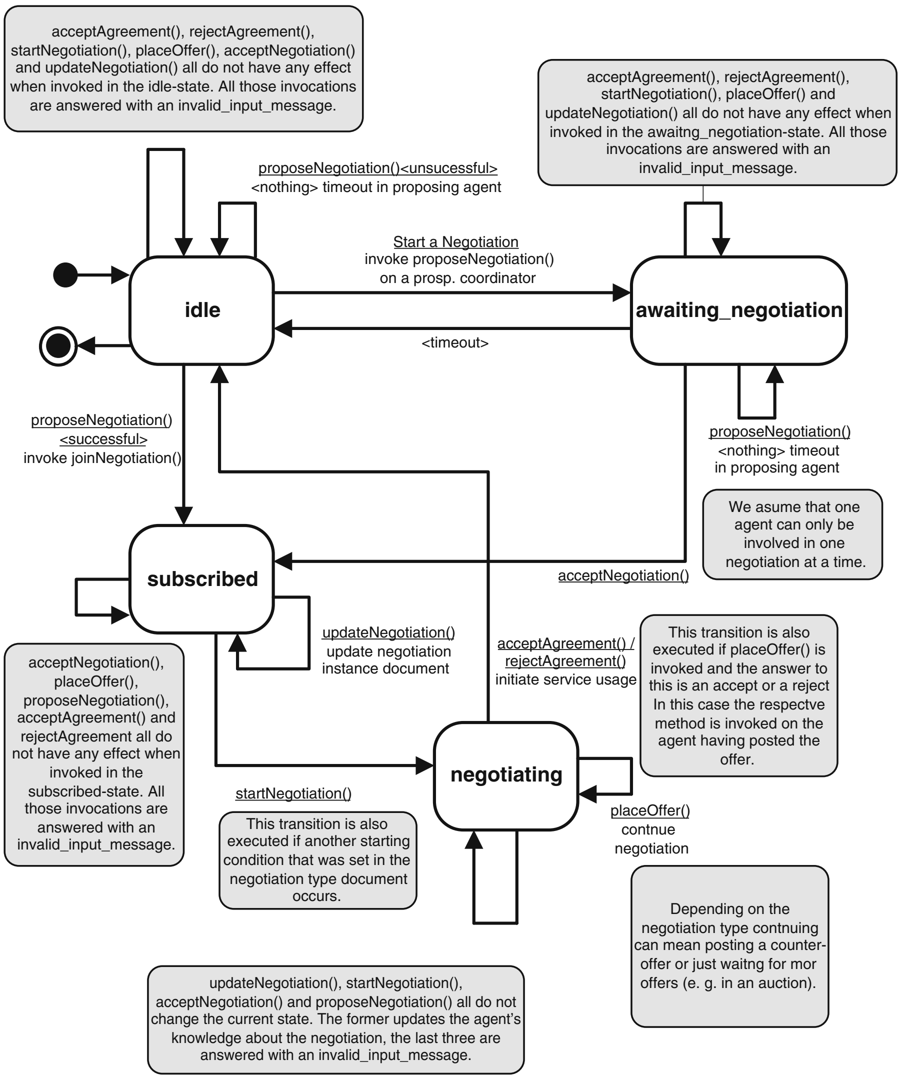
acceptAgreement(), rejectAgreement(), startNegotiation(), placeOffer(), acceptNegotiation() and updateNegotiation() all do not have any effect when invoked in the idle-state. All those invocations are answered with an invalid_input_message.
acceptAgreement(), rejectAgreement(), startNegotiation(), placeOffer() and updateNegotiation() all do not have any effect when invoked in the awaitng_negotiation-state. All those invocations are answered with an invalid_input_message.
proposeNegotiation()<unsucessful>
<nothing> timeout in proposing agent
Start a Negotiation
invoke proposeNegotiation() on a prosp. coordinator
idle
awaiting_negotiation
<timeout>
proposeNegotiation() <successful> invoke joinNegotiation()
proposeNegotiation() <nothing> timeout in proposing agent
We asume that one agent can only be involved in one negotiation at a time.
subscribed
acceptNegotiation()
updateNegotiation() update negotiation instance document
acceptAgreement() / rejectAgreement() initiate service usage
This transition is also executed if placeOffer() is invoked and the answer to this is an accept or a reject In this case the respectve method is invoked on the agent having posted the offer.
acceptNegotiation(), placeOffer(), proposeNegotiation(), acceptAgreement() and rejectAgreement all do not have any effect when invoked in the subscribed-state. All those invocations are answered with an invalid_input_message.
negotiating
startNegotiation()
This transition is also executed if another starting condition that was set in the negotiation type document occurs.
placeOffer() contnue negotiation
Depending on the negotiation type contnuing can mean posting a counter-offer or just waitng for mor offers (e. g. in an auction).
updateNegotiation(), startNegotiation(), acceptNegotiation() and proposeNegotiation() all do not change the current state. The former updates the agent's knowledge about the negotiation, the last three are answered with an invalid_input_message.

Fig. 8 Negotiation participant—internal states

For all possible method invocations the respective behavior of the agent is described in Fig. 7. A small detail to be especially mentioned is the *getCurrentNegotiations()*-method. This method returns all negotiation instances currently available. In the *idle* state these instances can only be those that are being coordinated by some other agent. This depicts the aforementioned possibility for distributed lookup servers for negotiation documents.

6.2 Negotiation Participant

The Negotiation Participant role exposes four different internal states: *idle*, *awaiting_negotiation*, *subscribed* and *negotiating*. An agent moves from the *idle* state to the *awaiting_negotiation* state if it proposes a negotiation instance to a prospective coordinator. In this state the agent awaits the decision of the coordinator whether it is willing to govern the respective negotiation or not.

If the Negotiation Coordinator does not join the negotiation, a timeout occurs leading the proposing agent to move to the *idle* state again. On the other hand, the Negotiation Participant moves to the *subscribed* state if the negotiation is accepted by the Negotiation Coordinator, i.e., the *acceptNegotiation()*-method is invoked. In the *subscribed* state an agent has already joined a negotiation instance. However this negotiation has not started yet.

Whenever a starting-condition as stated in the negotiation type/instance document occurs the negotiation is started. This leads to changing the agent's state to *negotiating*. After finishing a negotiation (no matter if reaching or failing to reach an agreement) the agent moves back to the *idle* state.

Note that for reasons of simplicity, it is assumed in Fig. 8 that an agent may only be involved in one negotiation at a time.

6.3 Information Service

Figure 9 shows the internal states of the InformationService role. After starting the negotiation the agent moves from the *idle* to the *active* state. After terminating the negotiation it returns to the *idle* state. Analogously to the Negotiation Participant

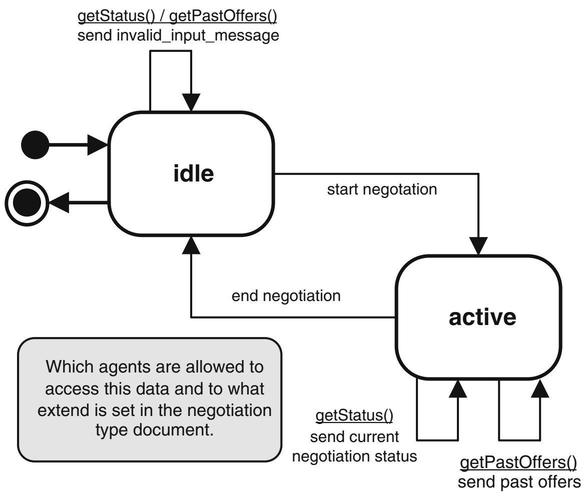
getStatus() / getPastOffers()
send invalid_input_message
idle
start negotiation
end negotiation
active
Which agents are allowed to access this data and to what extend is set in the negotiation type document.
getStatus()
send current negotiation status
getPastOffers()
send past offers

Fig. 9 Information service—internal states

the transition from *idle* to *active* can be triggered by the occurrence of any starting condition as stated in the negotiation type/instance document.

7 Related Work

Negotiation protocols and strategies have been studied for quite an amount of time now. However such issues are mainly investigated within social sciences or economics (see for example [13, 20, 27]).

Ubiquitous access to global communication networks, such as the Internet, and advances in information technology recently have led to negotiations that are electronically supported and even fully electronic negotiations. Electronic negotiations are characterised as negotiation processes employing electronic devices within at least one of the above mentioned phases [6]. Thus electronic negotiations and negotiation systems range from communication structures, only providing electronic means for transmitting messages, to negotiation support systems (NSS) assisting the negotiator in assessing the problem space, formalising his preferences and evaluating bids [16].

Current research is focussing on implementing fully automated negotiations, conducted by software agents [23]. Electronic negotiations based on interacting software agents are seen as the next step for automating the negotiation problem as present in everyday business transactions [24].

SLAs have been discussed in different research areas, such as Grid Computing, Service Oriented Architectures (SOA) or Information Systems Management [19, 21, 35]. They represent a powerful concept for the definition of mutually agreed upon QoS guarantees and serve as the main input for electronic service monitoring. SLAs have become both, a tool for electronic representation of legal business contracts and therefore acting as an input for policy definition, as well as a powerful concept for efficient resource management in distributed software systems [29]. Our aim is to fully automate the SLA Negotiation Phase for electronic SLAs in order to achieve such efficient resource management infrastructures for distributed computing.

Negotiation researchers argue that "[t]here is no perfect mechanism" (Peter Wurman, 2001) to negotiation problems, i. e., there is no negotiation protocol suitable for all possibly useful negotiation scenarios [5, 31, 32]. However, most of the SLA negotiation frameworks available today, like, e.g., the approaches described by Chhetri et al. [9], Pichot et al. [26], Chhetri et al. [8], Czajkowski et al. [10], Parkin et al. [25] or Mobach et al. [22], support a single negotiation protocol only or do not offer a mechanism for distributed automated decision on what protocol to use [18], therefore lacking efficiency of the outcomes.

On the other hand, electronic negotiation research already developed some negotiation systems capable of different protocols; see, for example, the work of Wurman et al. [36, 37] and Stroebel [33]. Those systems mainly focus on human negotiations and therefore lack a structured discovery and execution process for software agents. Such an approach is not applicable for electronic SLAs due to the complexity of the negotiations and the sheer volume of SLAs to be negotiated.

We currently found two approaches allowing for automated, protocol-generic approaches. Paprzycki et al. [24] only provide a distinct set of negotiation protocols to the agents, one of which has to be chosen at design-time. Our framework allows for very generic and a-priori not restricted negotiation protocols to be chosen at runtime. Rolli et al. [28] rely on database queries for defining their auction rules. This, however, creates a bottleneck, the database, with all problems a centralised instance exhibits in distributed systems. Our framework relies on a peer-based distributed architecture that is much more robust than a centralised system.

Regarding other standards, there has been some work done in recent years. The Foundation for Intelligent Physical Agents (FIPA) proposed a set of negotiation protocol descriptions for software agents as a standard way for agent interaction [11]. Our generic negotiation protocol language enables the definition of but is not limited to these standard protocols.

The approach of Hung et al. [15] tries to formulate the overall negotiation process including the negotiation messages, the protocol as well as the decision making procedures. However, the authors currently only sketch how their approach is going to look like, up to now there is no comprehensive specification of their language.

Additionally, the OGF currently prepares a proposal for a negotiation standard for WS Agreement: WSAgreementNegotiation [2]. The authors are involved in this standardisation process.

Building on the above mentioned research efforts we designed a framework for multi-protocol, fully automated SLA negotiations based on software agents. These agents accompany the service requestors and providers introducing an abstract SLA management layer in distributed systems.

8 Conclusion

This paper proposes a negotiation framework for WS-Agreement, enabling the integration of a variety of negotiation protocols suitable for different application domains. Negotiation protocols can be specified in a description language and made available to parties interested in respective negotiations through a defined exchange process. Executing this exchange protocol establishes among the involved agents, which negotiation protocol is used.

Subsequently, the protocol is executed to determine the resulting, negotiated WS-Agreement document. After winner determination, acceptance and rejection is performed according to the standard WS-Agreement protocol. With these two protocols fully automated WS-Agreement negotiations according to a variety of different negotiation protocols can be conducted in Web Service environments.

Future work focuses on testing a variety of negotiation protocols, e.g., in the context of other research projects dealing with service level agreements in Grid environments, such as SORMA [30], and thus verifying the expressiveness of the negotiation description language and the capabilities of the exchange protocol.

References

1. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web Services Agreement Specification Draft, Version 09/2005' (2005)
2. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement negotiation specification (WSAgreementNegotiation) (draft). Technical Report, Open Grid Forum, GRAAP WG (2008)
3. Andrieux, A., Dan, K.C.A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: Web services agreement specification (WS-Agreement). Published at the Open Grid Forum (OGF) Website (2007)
4. Bartolini, C., Preist, C., Jennings, N.R.: A software framework for automated negotiation. In: Choren, R., Garcia, A., Lucena, C., Ramonovsky, A. (eds.) Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications, pp. 213–235. Springer, New York (2005)
5. Benyoucef, M. Rinderle, S.: Modeling e-negotiation processes for a service oriented architecture. Group Decis. Negot. **15**(5), 449–467 (2006)
6. Bichler, M., Kersten, G., Strecker, S.: Towards a structured design of electronic negotiations. Group Decis. Negot. **12**(4), 311–335 (2003)
7. Chen, P.P.-S.: The entity-relationship model—toward a unified view of data. ACM Trans. Database Syst. Arch. **1**(1), 9–36 (1976)
8. Chhetri, M.B., Goh, S., Lin, J., Brzotowski, J., Kowalczyk, R.: Agent-based negotiation of service level agreements for web service compositions. In: Proceedings of the Joint Conference of the INFORMS Section on Group Decision and Negotiation, the EURO Working Group on Decision and Negotiation Support, and the EURO Working Group on Decision Support Systems, GDNŠ07, Montreal, 14–17 May 2007
9. Chhetri, M.B., Lin, J., Goh, S., Zhang, J.Y., Kowalczyk, R., Yan, J.: A coordinated architecture for the agent-based service level agreement negotiation ofweb service composition. In: ASWEC '06: Proceedings of the Australian Software Engineering Conference (ASWEC'06), pp. 90–99. IEEE Computer Society, Washington, DC (2006)
10. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: a protocol for negotiating service level agreements and coordinating resource management in distributed systems. In: 8th Workshop on Job Scheduling Strategies for Parallel Processing, Edinburgh, July 2002
11. FIPA: FIPA Communicative Act Library Specification. FIPA TC Communication (SC00037J) (2002)
12. Friedman-Hill, E.: Jess: The Java Expert System Shell. Sandia, Albuquerque (2006)
13. Gulliver, P.H.: Disputes and Negotiation: A Cross-Cultural Perspective. Academic, New York (1979)
14. Hudert, S.: A Proposal for a Web Services Agreement Protocol Framework. Bamberger Beiträge zur Wirtschaftsinformatik69, Bamberg University. ISSN 0937-3349 (2007)
15. Hung, P.C.K., Li, H., Jeng, J.-J.: WS-Negotiation: an overview of research issues. In: HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS' 04) - Track 1, p. 10033.2. IEEE Computer Society, Washington, DC (2004)
16. Kersten, G.E. Noronha, S.J.: WWW-based negotiation support: design, implementation, and use. Decis. Support Syst. **25**(2), 135–154 (1999)
17. Lomuscio, A.R., Wooldridge, M., Jennings, N.R.: A classification scheme for negotiation in electronic commerce. Int. J. Group Decis. Negot. **12**(1), 31–56 (2003)
18. Ludwig, A., Braun, P., Kowalczyk, R., Franczyk, B.: A framework for automated negotiation of service level agreements in services grids. In: Lecture Notes in Computer Science, Proceedings of the Workshop on Web Service Choreography and Orchestration for Business Process Management, 2006, vol. 3812/2006. Springer, New York (2006)
19. Ludwig, H., Keller, A., Dan, A., King, R., Franck, R.: A service level agreement language for dynamic electronic services. J. Electron. Commer. Res. **3**, 43–59 (2003)
20. McAfee, P., McMillan, J.: Auctions and bidding. J. Econ. Lit. **25**, 699–738 (1987)
21. Mitchell, B., Mckee, P.: SLAs a key commercial tool. In: Proceedings of eChallenges e-2005, Ljubljana, 19–21 October 2005
22. Mobach, D.G.A., Overeinder, B.J., Brazier, F.M.T., Dignum, F.P.M.: A two-tiered model of negotiation based on web service agreements. In: Gleizes, M.P., Kaminka, G.A., Nowé, A., Ossowski, S., Tuyls, K. Verbeeck, K. (eds.) EUMAS, pp. 202–213. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, Lelystad (2005)
23. Nassif, L., Nogueira, J.M., Ahmed, M., Impey, R., Karmouch, A.: Agent-based negotiation for resource

allocation in Grid. In: Proceedings of the 3rd Workshop on computational Grids and Applications, Summer Program LNCC (2005)
24. Paprzycki, M., Abraham, A., Prvanescu, A., Badica, C.: Implementing agents capable of dynamic negotiations. In: Petcu, D., Negru, V. (eds.) SYNASC, pp. 369–380. Mirton, Timisoara (2004)
25. Parkin, M., Kuo, D., Brooke, J.: A framework & negotiation protocol for service contracts. In: IEEE SCC, pp. 253–256. IEEE Computer Society, Washington, DC (2006)
26. Pichot, A., Wieder, P., Waeldrich, O., Ziegler, W.: Dynamic SLA-negotiatioan based on WS-agreement. Technical Report TR-0082, CoreGRID (2007)
27. Pruitt, D.G.: Negotiation Behavior. Academic, New York (1981)
28. Rolli, D., Luckner, S., Gimpel, H., Weinhardt, C.: A descriptive auction language. J. Electron. Mater. **16**(1), 51–62 (2006)
29. Seidel, J., Waeldrich, O., Ziegler, W., Wieder, P., Yahyapour, R.: Using SLA for resource management and scheduling—a survey. Technical Report TR-0096, CoreGRID (2007)
30. SORMA: EU Information society technologies project SORMA—Self-Organizing ICT Resource Management (2007)
31. Stroebel, M.: Effects of electronic markets on negotiation processes. In: Proceedings of the 8th European Conference on Information Systems, vol. 1, pp. 445–452 (2000a)
32. Stroebel, M.: On auctions as the negotiation paradigm of electronic markets. EM J. Electron. Mater. **10**(1), 39–44 (2000b)
33. Stroebel, M.: Design of roles and protocols for electronic negotiations. Electronic Commerce Research **1**, 335–353 (2001)
34. Stroebel, M. Weinhardt, C.: The Montreal taxonomy for electronic negotiations. J. Group Decis. Negot. **12**, 143–164 (2003)
35. Tosic, V., Pagurek, B., Esfandiari, B., Patel, K., Ma, W.: Web Service Offerings Language (WSOL) and Web Service Composition Management (WSCM). In: Workshop on Object-Oriented Web Services—OOWS (at OOPSLA 2002), Seattle, November 2002
36. Wurman, P.R., Wellman, M.P., Walsh, W.E.: The Michigan internet AuctionBot: a configurable auction server for human and software agents. In: Second international conference on autonomous agents, Minneapolis, 9–13 May 1998
37. Wurman, P.R., Wellman, M.P., Walsh, W.E.: A parametrization of the auction design space. Games Econom. Behav. **35**(1–2), 304–338 (2001)