RC24248 (W0704-149) April 27, 2007
Computer Science

IBM Research Report

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

Sebastian Hudert
University of Bayreuth
Germany

Heiko Ludwig
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Guido Wirtz
Otto-Friedrich University
Bamberg, Germany

Negotiating SLAs - An approach for a generic negotiation framework for WS-Agreement

Sebastian Hudert
University of Bayreuth, Department of Information Systems

Heiko Ludwig
IBM Almaden Research Center, San Jose, CA

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

April 14th, 2007

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 analyzing a couple of taxonomies from the literature in order to allow for capturing a variety of different negotiation models within a single, WS-Agreement compatible, framework.

Keywords: SLA, Negotiation Protocols, WS Agreement

1. Introduction

Managing Quality of Service (QoS) in loosely-coupled distributed systems such as computational Grids cannot rely on traditional, centralized 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 standardization effort conducted by the Open Grid Forum (OGF) in order to facilitate creation and monitoring of SLAs (Andrieux et al., 2005). This standard defines an XML-based structural definition of SLA documents, a simple request-response protocol for agreeement 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 in 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 negotiations 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 negotition protocol definitions could be specified. During the actual negotiation the corresponding protocol description is simply referenced. However, this limits the set of available negotiation protocols to a predefined, finite set.

In this paper, we propose an 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 with protocol has been chosen. Furthermore, we propose a negotiation meta-protocol to distribute the negotiation definitions to all prospective negotiators and choose a negotiation protocol. Finally we propose, as an example, a generic negotiation protocol that is able to support all specific negotiation protocols that can be described with the presented negotiation attributes as extension to basic WS-Agreement offers. This generic protocol notwithstanding, other negotiation protocols can be used.

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 philosphy as well as the interfaces and building blocks offered for the negotiation meta-protocol and the negotiation protocol itself are introduced, respectively. We conclude with 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.

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 in multi-agent systems 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 meta-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.

The first taxonomy used has been presented by Martin Bichler and Jayant R. Kalgnanam (Bichler and Kalagnanam, 2007). Their work puts its focus on one particular class of negotiation protocols: auctions. Although this approach does not cover as many negotiation protocols as would be desirable it nevertheless presents some interesting aspects that can be applied either to auctions and negotiations in general.

The next paper proposing a taxonomy for electronic negotiations was published by Alessio R. Lomuscio, Michael Wooldridge and Nicholas R. Jennings (Lomuscio et al., 2003). They emphasize automated negotiations in electronic commerce (e-commerce) settings. Even though these scenarios and software agents do not exactly fit the SLA settings, 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 between software agents. Such problems are formalisation of negotiations or implementation of negotiation strategies by agents themselves. The authors want "‘to define (...) the negotiation space for electronic commerce"’ (Lomuscio et al., 2003). 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 the approach of creating a data structure for multiple negotiation protocols.

Peter Wurman, Michael Wellman and William Walsh presented a set of auction parameters while developing an internet-based "‘platform for price-based negotiation - the Michigan Internet AuctionBot"’ (Wurman et al., 1998). 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 this taxonomy to also cover multidimensional auctions in a follow-up, much more comprehensive paper (Wurman et al., 2001). 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 this thesis 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 one attribute (most commonly 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) (Bartolini et al., 2005). Thus, the authors focus on executable specifications for MAS. Thus 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 last taxonomy used was published by Michael Ströbel and Christof Weinhardt (Stroebel and Weinhardt, 2003). It represents the most comprehensive categorization and description of electronic negotiations of all taxonomies. 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 (Wurman et al., 2001). They regard auctions as a particular 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"’ (Stroebel and Weinhardt, 2003). 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 techonology-related issues.

For our framework the taxonomies discussed so far were integrated and consolidated in order to derive a set of attributes and corresponding domains suitable for definition of automated SLA negotiations among software agents. The complete list of identified attributes and corresponding domains is outside the scope of this paper; for a more detailed description of our WS-Agreement negotiation taxonomy see (Hudert, 2007). The high-level attribute classes identified in our work are the following:

- General Negotiation Process: Basic negotiation parameters like start, termination or negotiation rounds.
- Negotiation Context: Negotiation configuration in terms of involved roles and agents.
- Negotiated Issues: SLA terms to be negotiated in the corresponding negotiation.
- Offer Submission: Rules concerning the bidding process, like when an offer can be posed or what constraint it has to satisfy.
- Offer Allocation: Matchmaking rules for the neogtiation.
- Information Processing: Rules defining which information about the current negotiation and bidding history is available to which agent(s).

For each of these classes a set of attributes was defined and, if feasible, also a corresponding value domain. Because preventing the generic approach of this framework from unduly restrictions was an important design issue, flexibility has been increased by allowing the use of some (external) rule-based language, like Jess (Friedman-Hill, 2006), to express some of the attributes adequately. Such attributes do not exhibit a defined value domain respectively.

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. For a more detailed description and two example protocol descriptions for an auction and a one-on-one bargaining protocol, see (Hudert, 2007).

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 of some type. 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 auctions. One instance of this negotiation type therefore represents one particular auction process.

Regarding their content, negotiation types and instantiated negotiations differ slightly. The main difference is that a negotiation type does not contain some sort of identifier that can be used to refer to a given negotiation instance. The other attributes identified in the previous section, however, have to be initialized when defining a negotiation type. For example, the offer submission rules and involved roles have to be set before instantiating because they represent a fundamental part of a whole class of negotiation instances, and therefore of a negotiation type.

There are only three exceptions; only three attributes do not necessarily have to be defined in a negotiation type document: the start and the termination as well as 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. If a negotiation start or termination is specified in terms of time points, this has to be done in the negotiation instance document; when defined as a constraint expression over other parameters it has to be set in the negotiation type document.

The involved agents are not specified in a negotiation type document for two reasons: Foremost, not every agent participating in the negotiation is known when a negotiation type is created. Typically, only one agent will be known mostly: the agent that provides the negotiation type document. During the exchange protocol described in the next sections other agents subsequently join this negotiation without being known to the agent that created the negotiation type. Hence, not all agents involved can be defined in the negotiation type document, but have to be incorporated into the datastructure representing the actual negotiation instance subsequently. Another reason for not already setting the involved agents in the type document is that if incorporated in the negotiation type every instance of this type would have to be conducted by exactly the same agents which is not feasible. Determining the participating agents in the negotiation instance document allows for a much more flexible application of the approach.

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) for each of these concepts were defined. These documents will be used within the actual negotiation protocols as described in the following sections. The detailed XML-Schema documents can be found in (Hudert, 2007).

2.3. Abstract Architecture of Negotiation Documents

The main negotiation object is a WS-Agreement template (Ludwig et al., 2003) with its corresponding creation constraints as defined in the current WS-Agreement specification. 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 initialized 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 as already hinted. This document refers to the negotiation's type, its participants and specifies a unique identifier. 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 referenced in the negotiation type document.

2.4. Involved Roles

In order to fix the different parties involved in a negotation process as well as their expected behaviour, 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. In such a negotiation process the Negotiation Participants represent regular agents participating in the initial negotiation meta-protocol (used to distribute the negotiation documents to the prospective negotiators) and the negotiation process itself. 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 administrated by the Information Service. This service offers

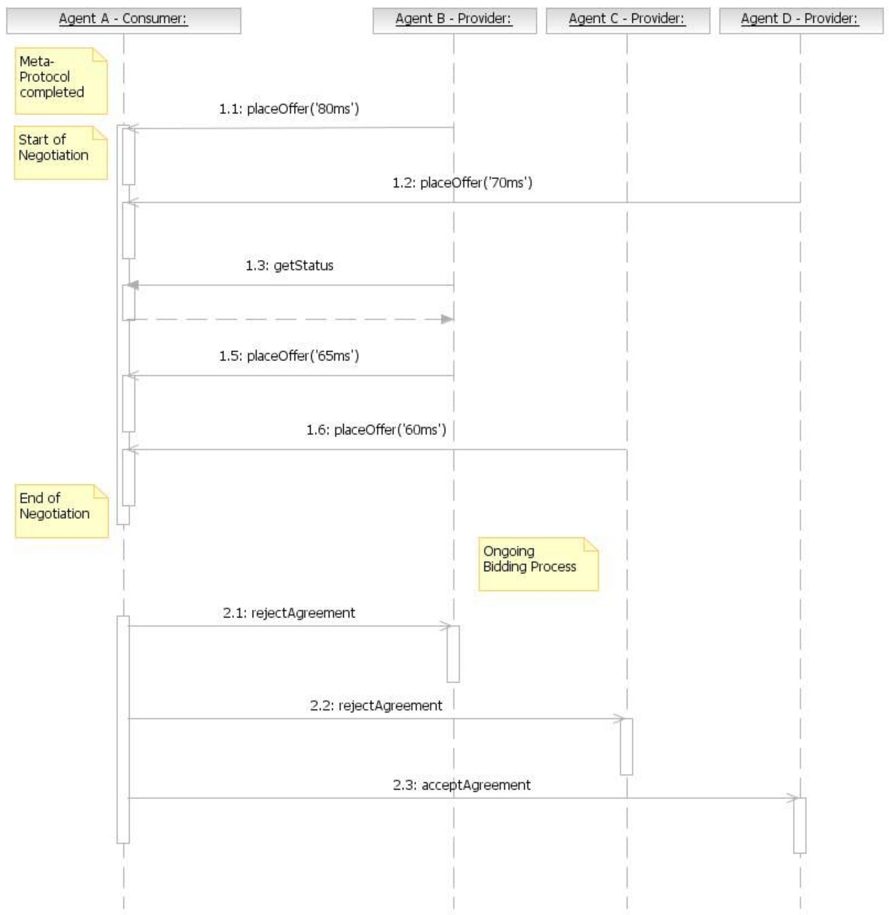

Figure 1. Scenario - Example Auction

information about the current status of a negotiation (for example the currently highest bid) or about the offer history to the negotiators.

The basic guidelines, data structures and roles presented here, will be made more concrete in the next sections by using an example scenario to introduce the negotiation meta-protocol for negotiation data as well as the negotiation protocol more preciesely.

3. Example Scenario

In order to make our concept clear, the following scenario (see Figure 1) provides a typical application for the framework presented here: 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 Service Level Agreements (SLAs), using the WS-Agreement standard. In order to allow for flexible negotiations of such SLAs, each Grid node is associated with a software agent or some similar intelligent service used for the negotiation process, e. g. offering the needed negotiation interfaces as proposed by our approach. In our scenario Grid service A (represented by software agent A) requests some service 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, response time is an important service quality parameter of the offered service. Since agent A potentially submits a big batch of currency conversions during a given timespan 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 of at a response time of 100ms which is the maximum agent A can accept. A typical interaction stemming from this setting is shown in Figure 1

For this purpose agent A creates a WS-Agreement template document according to the current specification draft (Andrieux et al., 2005). 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 100ms is set for the `guaranteedResponseTime` 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 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 I shows some excerpts of the type definition for a negotiation as described in section 2.3: besides referencing an WS-Agreement template, two roles, i.e. `serviceProvider` and `serviceConsumer` are introduced as well as the `guaranteedResponseTime` issue to be negotiated with a restrictive upper bound. In contrast to the negotiation type definition, Table II, presents some aspects of an negotiation instance that references this auction type: con-

Table I. Auction type description for 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>
```

Table II. Auction type 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>
```

crete start and end dates are given as well as the involved agents according to the roles to be defined in the next two sections in more detail.

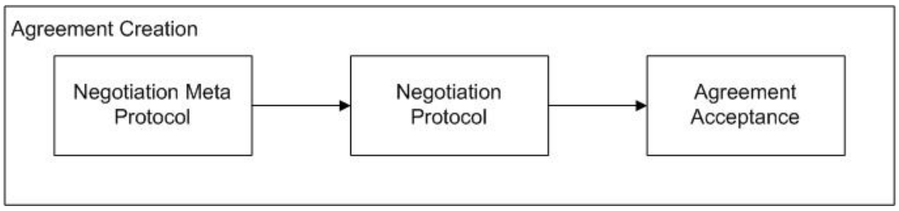

Figure 2. Agreement Creation Processs

4. Negotiation Meta-Protocol

The framework supports the complete process of agreement creation. As depicted in Figure 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 negotiation meta-protocol as defined in this section. Subsequently the actual negotiation process takes, 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 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 meta- 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 works in a Web Services setting, the meta-protocol will not focus on exchanged messages primarily, but on the provided services and respective methods to be invoked subsequently. This approach was taken due to the service oriented environment for WS-Agreement negotiations. Such service oriented environments focus inherently on provided services and define protocols in terms of method invocation sequences. 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 of negotiation data can be realized.

4.1. Interfaces for the Roles Involved

In order to conduct a negotiation, the involved agents are assumed to take part in one particular negotiation instance of one particular type as defined above. The data structure describing the negotiation instance contains 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 the previous section. 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 metadata 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 hints at which of the roles already introduced are involved in the exchange process of the negotation meta-protocol: a centralized Negotiation Coordinator interacts with one 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 as well as the XML schema for all data structures used can be found in (Hudert, 2007).

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

– *getAllNegotiationTypes()/getAllNegotiationTypesForTemplate(...)*

– *getCurrentNegotiations()/getCurrentNegotiationsForTemplate(...)*

Thus, very general queries are possible as well as queries concerning negotiations currently active. The second dimension, indicated by 'template', allows for using WS-Agreement templates when performing a query. Based on this information, participants should also be able to act in different ways in order to initiate and/or become involved in negotiations. Besides simply asking to join an already running process, an agent may actively propose a negotiation instance to a coordinating agent. This agent may then either act as coordinator only in the respective negotiation or also join the negotiation as a participant.

– *joinNegotiation(negotiationID, agentEPR, 'credentials')*

– *proposeNegotiation(NegotiationInstance-document)*

– *publishNegotiation(NegotiationInstance-document)*

– *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, propably 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 or just be offered pull-style as a result of the query-methods.

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 meta-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 meta-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.

- *updateNegotiation(NegotiationInstance-document)*
- *proposeNegotiation(NegotiationInstance-document)*
- *acceptNegotiation(negotiationID)*

On the other hand, a participant should be able to actively propose a negotiation instance to another (potential) Negotiation Participant. 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 participant this agent can decide whether to join or not. If it joins, it invokes the *acceptNegotiation()*-method, otherwise nothing more happens.

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 that are implemented using the roles and methods presented: request for and proposal of negotiation data as well as their mediated exchange.

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. Within the exchange protocol two different types of negotiation documents can be queried: negotiation type or negotiation instance information. 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.

Accordingly, 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. As a result the coordinator returns a list of negotiation type documents. 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 *getCurrentNegotiations()*. As described earlier a negotiation instance is created from a certain type by specifying the involved agents and optionally the start and termination rules of the negotiation. Hence a negotiation instance can be already running when queried. 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. This template represents a generic WS-Agreement, with references to (possibly already existing) services, elements yet to be filled in and corresponding restrictions. 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 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 result of the proposal. Asynchronous communication was considered useful here because this concept allows a (potential) coordinating agent to check its current circumstances before accepting a request. In the process visualized in Figure 3, the coordinating agent would be set as Negotiation Coordinator within the instance document. After such a negotiation is proposed, the coordinator checks it's resource situation for deciding whether to accept such a negotiation or not. The diagram depicts the situation where the Negotiation Coordinator accepts the negotiation instance by invoking the *acceptNegotiation()*-method on the proposing agent. 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 Figure 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 component regularly follows a request for negotiation types in order to propose the newly created instance to the coordinating agent. The details

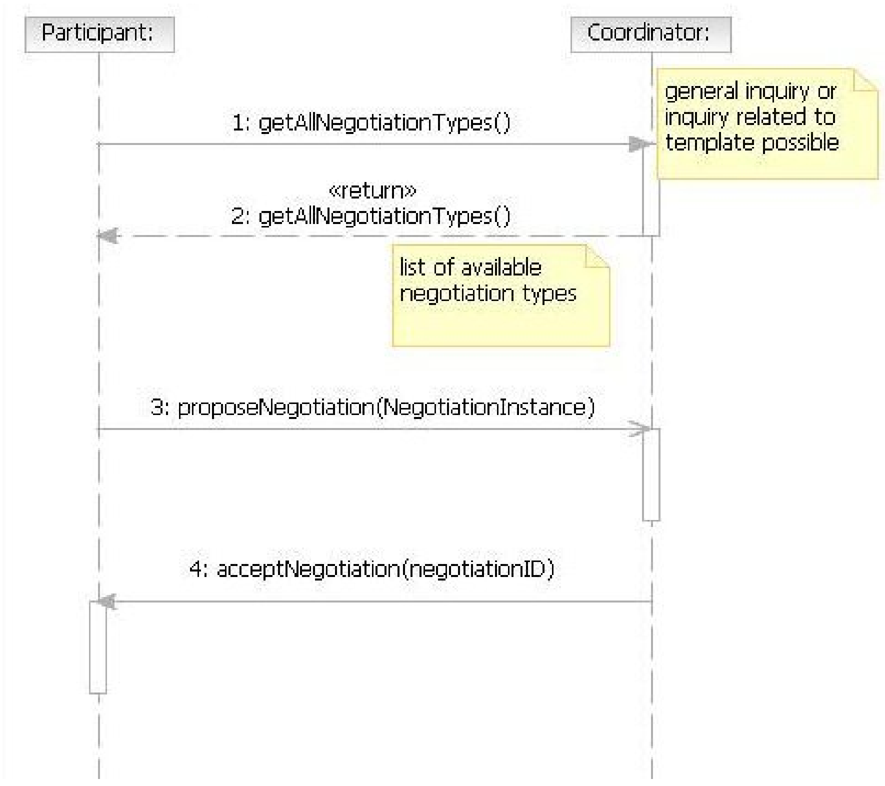
Participant:
Coordinator:
general inquiry or inquiry related to template possible
1: getAllNegotiationTypes()
«return»
2: getAllNegotiationTypes()
list of available negotiation types
3: proposeNegotiation(NegotiationInstance)
4: acceptNegotiation(negotiationID)

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

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. In order to implement such architectures, the Negotiation Coordinator

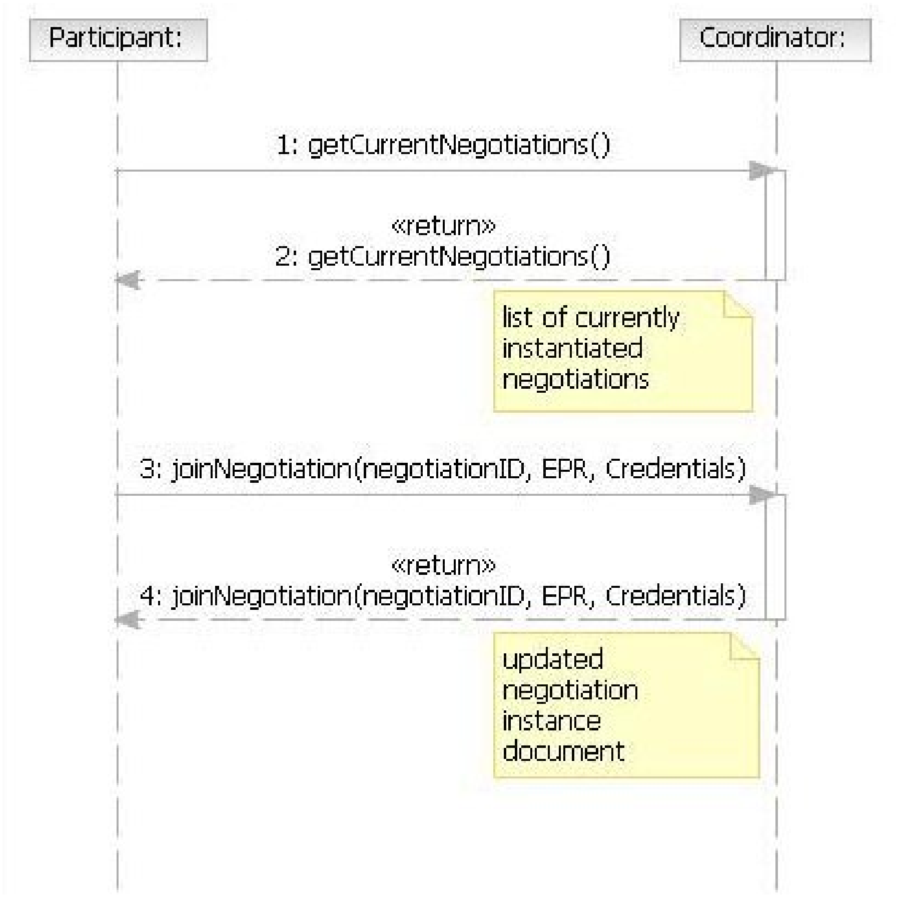
Participant:
Coordinator:
1: getCurrentNegotiations()
«return»
2: getCurrentNegotiations()
list of currently instantiated negotiations
3: joinNegotiation(negotiationID, EPR, Credentials)
«return»
4: joinNegotiation(negotiationID, EPR, Credentials)
updated negotiation instance document

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

offers the *publishNegotiation()* interface 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, which is therefore also present in the Negotiation Participant interface.

The diagram in Figure 5 shows an exchange process where an agent A proposes a negotiation instance to the coordinator that proposes 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 requesting this document by using request-methods could join, respectively.

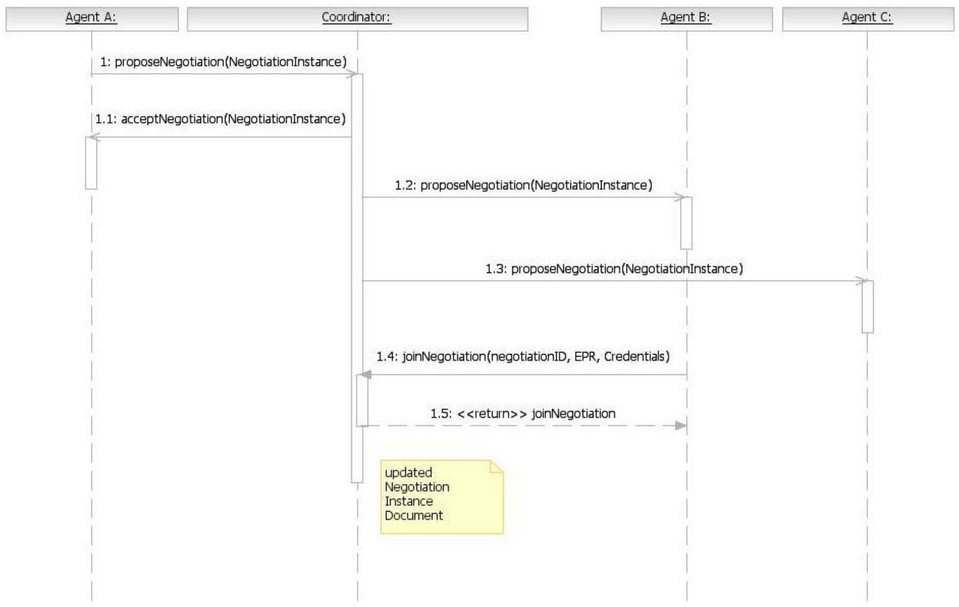
Agent A:
Coordinator:
Agent B:
Agent C:
1: proposeNegotiation(NegotiationInstance)
1.1: acceptNegotiation(NegotiationInstance)
1.2: proposeNegotiation(NegotiationInstance)
1.3: proposeNegotiation(NegotiationInstance)
1.4: joinNegotiation(negotiationID, EPR, Credentials)
1.5: <<return>> joinNegotiation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

Figure 5. Mediated Exchange Process

By combining these three basic protocol components as bulding 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 of 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. Since SLA scenarios only exhibit two logical positions actively involved in a negotiation, the service providers and consumers, only such two-sided negotiation protocols are considered here.

Although only two sides of a negotiation are present, a multitude of different protocols can be defined. In One-on-One Bargaining, for example, each side consists of only one agent. These two agents take turns in posting offers and counter-offers until one of these posted bids is accepted by the other agent. On the other hand, negotiation protocols such as auctions let all agents involved on one side post offers to the one agent representing the other side. Each agent can offer more than just one bid, but every agent can have only one currently valid bid. Hence, when posting a new offer this offer replaces the last one posted by the same agent. When the negotiation is terminated the currently best valid bid will be transfered into the resulting agreement. The data structures used for this purpose allow for the definition of a multitude of auction- and bargaining-like negotiation types. In order to support such processes 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.

NEGOTIATION PARTICIPANT INTERFACE

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 meta-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 EPR of the posting agent and a complete WS-Agreement document representing the offered SLA.

– *placeOffer(agentEPR, WS-Agreement-document)*

– *acceptAgreement(negotiationID, agreementID)*

– *rejectAgreement(negotiationID)*

In order to promote a negotiation's outcome to all participants, the *accept/reject* methods are used. In the positive case, the winner(s) of the negotiation and therefore the agent(s) involved in the resulting agreement are 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 after the termination of the negotiation. Thus, all agents not being involved in the resulting agreement are informed of the negative outcome.

Usually, the protocol involves much more negotiation steps than a simple offer-agree/reject pair. For this purpose, the

– *newRound(negotiationID, 'Information')*

method enables centralised coordination of multi-round negotiations. Each round is started when the Negotiation Coordinator invokes this method on all particpants (except the first one whose start is defined in the Start-attribute of the Negotiatoin Type/Instance). Multi-rounded negotiations consist of several phases, which are all conducted according to the same protocol. The only difference between the individual rounds is the knowledge of the involved agents. At the beginning of each round, some negotiation information is revealed to the participants that was not available in the previous round. This way the involved agents can alter their offers based on their increased knowledge about the negotiation. Negotiation designers could, for example, define a negotiation protocol, that allows only for one sealed bid from each participant in each round. After each round the bids from all other agents are promoted to all participants and every agent can post another offer in round two, and so on. In order to promote the newly accessible information the push-concept is applied. When invoking the *newRound()*-method the coordinator service posts this information to all agents along with the identifier of the negotiation the new round is started for. The parameter(s) needed here are the ID of the negotiation instance for which a new round should be started, the datastructure containing some information about the bidding process, that was not available in the last round for push-distribution, for example the current negotiation status.

INFORMATION SERVICE INTERFACE

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

– *getStatus(negotiationID)*

– *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.

Using these roles and interfaces a rather generic negotiation protocol is defined, capable of conducting any negotiation protocol describable with the attributes identified before.

Example Negotiation

In the rest of this section, the approach is illustrated by describing an auction process conducted using the above presented interfaces. The diagram in Figure 6 describes such an auction process: In this negotiation process 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.

Initially, the necessary negotiation data has to be distributed among the participants according to the exchange protocol defined in the last section. Under the assumption that this phase of negotiation 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 diagramm 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 diagram.

After the negotiation is terminated (the termination condition again stated in the negotiation type or instance document), 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 shown in the diagram.

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

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 Figure 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 realizes that agent

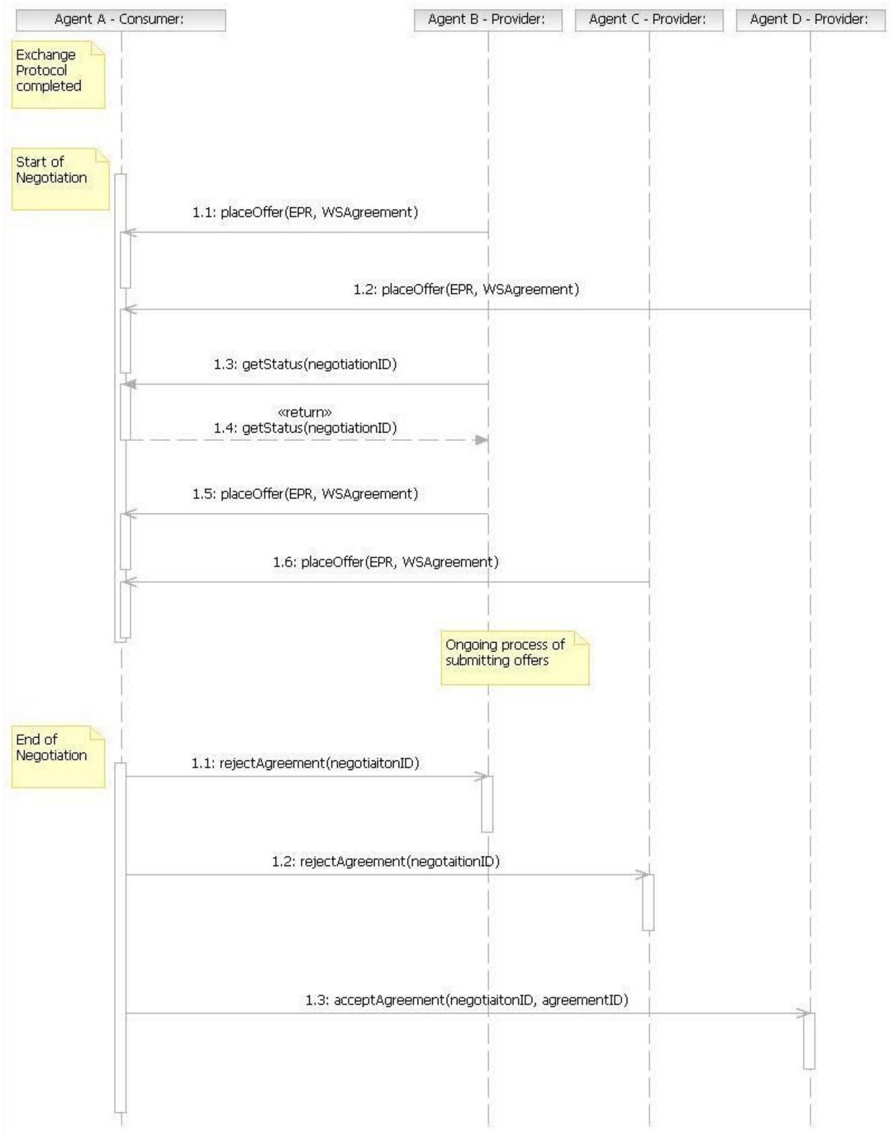
Agent A - Consumer:
Agent B - Provider:
Agent C - Provider:
Agent D - Provider:
Exchange Protocol completed
Start of Negotiation
1.1: placeOffer(EPR, WSAgreement)
1.2: placeOffer(EPR, WSAgreement)
1.3: getStatus(negotiationID)
«return» 1.4: getStatus(negotiationID)
1.5: placeOffer(EPR, WSAgreement)
1.6: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

Figure 6. Sample Auction Process

D posted an offer exceeding it's 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 loose the negotiation. As a reaction to this negotiation status agent B creates another, better offer and posts it to agent A to succeed the formerly winning offer. However, after an ongoing process of offer submission agent D still wins the negotiation in the process described in this example.

6. Conclusion

This paper proposes a negotiation framework for WS-Agreement, enabling the integration of a variety of negotiation protocols suitable for different application domains based on a negotiation meta-protocol determining the actual negotiation protocol used. Negotiation protocols can be specified in a description language and made available to parties interested in negotiations. Parties interested in negotiating an agreement first run the negotiation meta-protocol to establish which negotiation protocol is used. Subsequently, the protocol is executed to determine the resulting, negotiated WS-Agreement document. Finally, after winner determination, acceptance and rejection is performed again 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 (SOR, 2007), and thus verifying the expressiveness of the negotiation description language and the capabilities of the negotiation meta-protocol.

References

: 2007, 'EU Information Society Technologies project SORMA - Self-Organizing ICT Resource Management'.

Andrieux, A., K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu: 2005, 'Web Services Agreement Specification Draft, Version 09/2005'.

Bartolini, C., C. Preist, and N. R. Jennings: 2005, 'A Software Framework for Automated Negotiation'. *Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications (eds. R. Choren, A. Garcia, C. Lucena, and A. Ramonovsky)* pp. 213–235.

Bichler, M. and J. R. Kalagnanam: 2007, 'Software Frameworks for Advanced Procurement Auction Markets'. *Communications of the ACM (CACM).*

Friedman-Hill, E.: 2006, 'Jess: The Java Expert System Shell'. *Sandia Laboratories.*

Hudert, S.: February 2007, 'A Proposal for a Web Services Agreement Protocol Framework'. Bamberger Beiträge zur Wirtschaftsinformatik 70, Bamberg University. ISSN 0937-3349.

Lomuscio, A. R., M. Wooldridge, and N. R. Jennings: 2003, 'A Classification Scheme for Negotiation in Electronic Commerce'. *Int Journal of Group Decision and Negotiation* **12**(1), 31–56.

Ludwig, H., A. Keller, A. Dan, R. P. King, and R. Franck: 2003, 'Web Service Level Agreement (WSLA): Language Specification Version 1.0'. (1.0).

Stroebel, M. and C. Weinhardt: 2003, 'The Montreal Taxonomy for Electronic Negotiations'. *Journal of Group Decision and Negotiation* **12**, 143–164.

Wurman, P. R., M. P. Wellman, and W. E. Walsh: 1998, 'The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents'. *Second International Conference on Autonomous Agents.*

Wurman, P. R., M. P. Wellman, and W. E. Walsh: 2001, 'A Parametrization of the Auction Design Space'. *Games and Economic Behavior* **35**(1-2), 304–338.