GWD-I
GRAAP-WG

Oliver Waeldrich
(Editor)

March 07, 2010

WS-Agreement Version Negotiation 1.0

Status of This Document

This document provides information to the Grid, Distributed Systems and Cloud Computing community about WS-Agreement Negotiation (version 1.0). In describes WS-Agreement Negotiation as an extension to the WS-Agreement Specification Version 1 (GFD.107). Distribution is unlimited.

Copyright Notice

Trademark

Abstract

Oliver Waeldrich
(Editor)

March 09, 2010

Contents

1 Introduction

1.1 Requirements

- Must support bilateral negotiation of agreement offers
- Must allow negotiation of agreement offers for new and renegotiation of existing agreements
- Must provide a symmetric protocol
- Must build on top of the WS-Agreement specification
- Must provide a simple negotiation state machine

1.2 Out of Scope

- Negotiation of Agreement
- Definition of compensation methods for negotiated offers
- Definition of concrete negotiation strategies

2 Notational Conventions and Terminology

The key words ‘MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” are to be interpreted as described in RFC 2119 (Bradner, 1997).

Negotiation
The negotiation is the process between an agreement initiator and an agreement responder of getting from an initial agreement template to an acceptable agreement offer. Negotiation of agreement offers is a non-binding process that allows two parties to exchange information and find a consensus for an acceptable agreement offer.

Negotiation Offer
A negotiation offer is a non-binding proposal for a potential agreement that one negotiation party makes to another. One or more negotiation offers made by one or more negotiating parties may precede a binding agreement offer as defined in the WS-Agreement specification. The offer describes the service a SLA to be negotiated is about and the associated quality of service in terms of guarantees. Additionally offers contain negotiation constraints that identify the negotiable terms as well as their value spaces.

Negotiation Counter Offer
An offer that is created on the base of a previous offer is called counter offer. Counter offers must take into account the negotiation constraints of the offer it

relates to. In general each offer in a negotiation process is a counter offer. In the context of this specification the term counter offer refers to the relationship of the originating offer and the offer that was created on the base of it.

Negotiated Offer
In the context of this specification a negotiated offer is an offer that has reached the accepted state. Negotiated offers can be used as valid agreement offers in order to create new agreements or to replace existing agreements.

Agreement Initiator
The agreement initiator is the entity in the negotiation process that creates an agreement based on a negotiated offer. This role corresponds to the *agreement initiator* role as defined in the WS-Agreement specification.

Agreement Responder
The agreement responder is the entity in the negotiation process that responds to an agreement creation request based on a negotiated offer. This role corresponds to the agreement responder role as defined in the WS-Agreement specification.

Negotiation Initiator
A negotiation initiator is a party in a negotiation process. The negotiation initiator acts behalf of either the agreement initiator or the agreement responder. It initiates and participates in a negotiation process. The negotiation initiator invokes the initiateNegotiation method of this specification.

Negotiation Responder
A negotiation responder is a party in a negotiation process. The negotiation responder acts behalf of either the agreement initiator or the agreement responder. The negotiation responder implements at least the NegotiationFactory and Negotiation port types of this specification.

Negotiation Participator
A negotiation participator is one party that takes part in a negotiation process. The negotiation participator can either be the negotiation initiator or the negotiation responder.

Negotiation Context
The negotiation context defines the type of a negotiation process, identifies the negotiation participators and their roles, and optionally specifies additional domain specific negotiation parameters, such number of negotiation rounds or expiration time.

Negotiation Offer Context
The negotiation offer context represents metadata associated with a specific negotiation offer. It contains information such as the id of the offer that was used to create this offer and the expiration time of the offer. It may also contain domain specific extensions in order to define augmented negotiation protocols.

Negotiation Constraints
The negotiation constraints are a method to control a negotiation process. A negotiation participator uses negotiation constraints to define the structure or specific values that are applicable for counter offers that are based on a specific offer. Therefore, negotiation constraints are a means to express the requirements of a negotiating party.

Negotiation Offer State
The negotiation offer state is used to describe a specific state in the life cycle of a negotiation offer. The negotiation offer state can include domain specific data that can be used by the negotiating parties to exchange information related to the offer life cycle, and advance the negotiation process in an efficient way.

2.1 Namespaces
The following is an XML or other code example:

```
http://schemas.ogf.org/graap/2009/11/ws-agreement-negotiation (code)
```

The following namespaces are used in this document:

Prefix	Namespace
wsag-neg	http://schemas.ogf.org/graap/2009/11/ws-agreement-negotiation
wsag	http://schemas.ggf.org/graap/2007/03/ws-agreement
wsa	http://www.w3.org/2005/08/addressing
wsrf-rp	http://docs.oasis-open.org/wsrf/rp-2
wsrf-rw	http://docs.oasis-open.org/wsrf/rw-2
xs/xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
wsdl	http://schemas.xmlsoap.org/wsdl

3 Use Cases

WS-Agreement Negotiation supports a wide set of use cases for the negotiation of new agreements and the renegotiating existing agreements. Therefore, WS-Agreement Negotiation provides a protocol and a language. The example below illustrates the negotiation of a new agreement.

3.1 Advance Reservation of Compute Resources
In this scenario, a service provider offers compute resources to customers that are available in a specific time frame. The compute resource service consists of a job submission service and portal application to manage the job submission service. The job submission service is a web service that provides methods for submitting and managing compute jobs, such as SubmitJob, StartJob, QueryJob, and CancelJob. These methods are exposed via the Web Service Description Language (WSDL). The portal application provides methods to manage the job submission service, such as UpdateUserProfile, GetResourceAvailability, GetResourceUsage, DeployApplication, or ManageStorage.

Advance reservations are an abstract concept to define an ongoing relationship between the resource provider and a resource consumer for jobs that are subsequently submitted in the context of the agreement. The resource provisioning model is implementation specific; whether resources are exclusively dedicated to a user, prediction models or preemption is used is up to the resource provider.

The compute resource provider offers available compute resources via an agreement template. The template includes the description of the service and its guarantees, as well as a set of options the customer can choose from. The service description contains the available compute resources and the timeframe the resources should be available. The offered compute resources may differ in hardware; e.g. they may have different CPU architectures, CPU speed, memory, or hard disk space. The service consumer may compose these resources in the agreement in order to satisfy its needs. Moreover, the resource provider offers different service levels for the compute resource service to its customers. A costumer may choose between different levels of Quality of Service (QoS) for availability and average response times for the service. The customer can select a service availability of 95%, 98%, 99% or 99.9%. The availability defines the probability that a request is processed within 15 seconds. For the average response time, a costumer may select a value of 0.5, 1, or 2 seconds and the number of requests per minute this guarantee must hold. The QoS parameters can be specified separately for the job submission service and the portal application. The pricing of the service is dependent on type and number of the selected compute resources and the selected QoS levels.

This template provides many possibilities to parameterize the compute resource service. Moreover, it contains parameters, such as pricing that can only be provided by the resource provider and are dependent on the chosen resources and the QoS guarantees. Once a customer has filled in all its requirements it sends the offer to the resource provider. The provider then checks whether the requested service can be provisioned. In case the service can be provided it sends back a completed counter offer with the pricing information to the customer that in turn can now choose to create a negotiated agreement based on the offer. In case the resource provider is not able to fulfill all the requirements stated by the customer, it can also send back a counter offer indicating a service it is able to provide instead. E.g. a customer has requested 128 nodes with 8GB memory in a given timeframe, but the resource provider could not fulfill this request at this time. Instead the provider sends back a counter offers for 96 nodes with 8GB memory and 32 nodes with 6GB memory for a lower price. The customer may now choose to accept the counter offer or to only reserve the 96 nodes fitting to its needs and purchase the remaining capacity somewhere else. The process of filling in all required fields of a negotiation offer can take multiple rounds.

At a later point in time, the customer may recognize that it requires more or less resources to efficiently complete its computation. In that case it may start a renegotiation of the agreement in order to scale the resources up or down, according to its requirements.

4 WS-Agreement Negotiation Model

The WS-Agreement negotiation model consists of 3 layers, the negotiation layer, the agreement layer and the service layer. These layers are depicted in Figure 1.

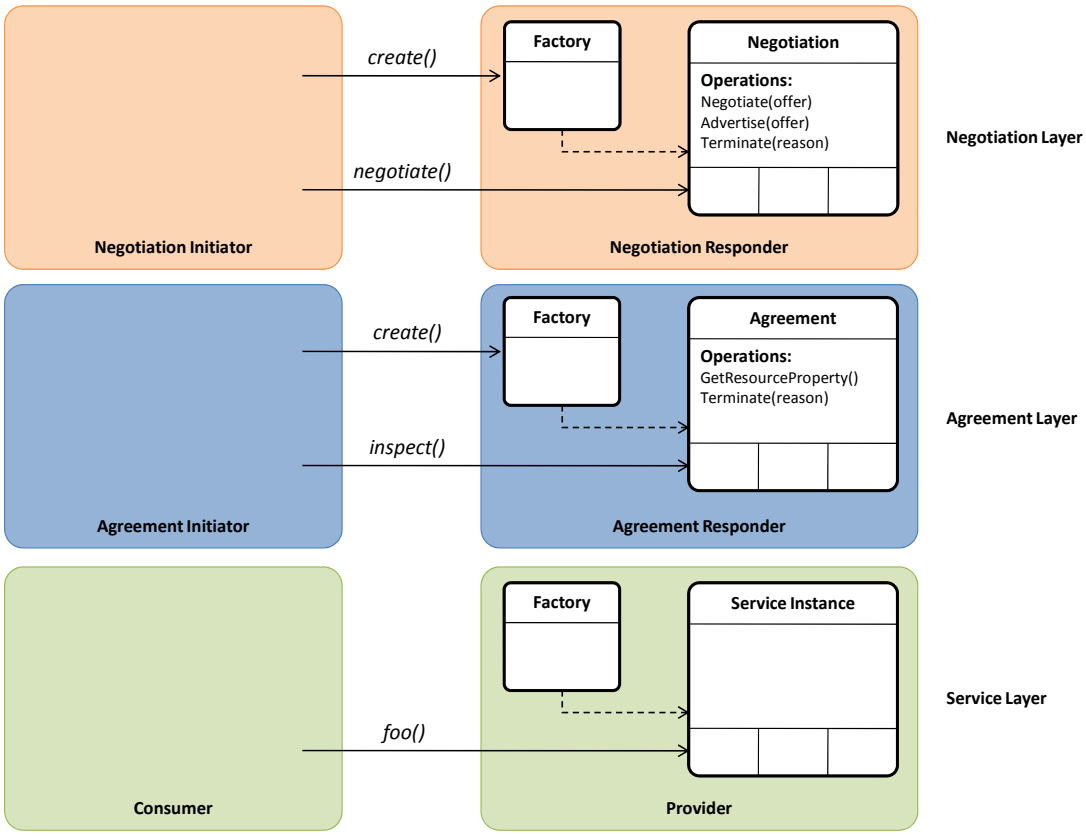

Figure 1: Conceptual overview of the layered negotiation model

There is a clear separation between these 3 layers in the negotiation model. The negotiation layer sits on top of the agreement layer. Therefore, it is decoupled from the agreement layer and the service layer. By that, the negotiation layer may change independently of the agreement layer and be replaced by another negotiation layer that may be better suited for specific negotiation scenarios.

Negotiation layer

The negotiation layer provides a protocol and a language to negotiate agreement offers and counter offers and to create agreements based on negotiated offers. The negotiation process comprises the exchange of offer and counter offers. Negotiation offers, as defined in this specification, are non-binding by nature. Therefore, negotiated offers do not make any promises that a subsequent agreement based on a negotiated offer will be created. They only indicate the willingness of two negotiating parties to accept a subsequent creation of an agreement. However, it is possible to create languages that can be used in conjunction with this specification in order to realize binding negotiation processes.

Negotiated agreements are created by either calling the *createAgreement* or *createPendingAgreement* operation on the Agreement Responders Agreement Factory instance, which is part of the agreement layer.

Agreement layer
The Agreement layer provides the basic functionality to create and monitor agreements. It provides a protocol and a language defined in the WS-Agreement specification. For further reference refer to the WS-Agreement specification.

Service layer
At the service layer the actual service defined by an agreement is provided. This service may or may not be a web service. Moreover, a service defined by an agreement may consist of multiple services, e.g. a service for resource provisioning may consist of the provisioning service and a monitoring service for the provided resources. The services on the service layer are governed by the agreement layer.

5 WS-Negotiation Protocol and Language

5.1 Negotiation

A Negotiation is a service instance that is used by two negotiating parties to exchange information in order to come to a common understanding of valid agreement offers. In the process of a negotiation the two negotiating parties exchange negotiation offers and indicate their goals and requirements. A negotiation may be limited in lifetime or rounds of negotiation. These limitations are defined in the negotiation context.

5.1.1 Negotiation Context

The negotiation context defines the roles of the negotiation participators, their obligations, and the nature of the negotiation process. Since a negotiation is a bi-lateral process, the roles of each participating party must be clearly defined.

```
<wsag-neg:NegotiationContext>
    <wsag-neg:NegotiationType>
        wsag-neg:NegotiationType
    </wsag-neg:NegotiationType>
    <wsag-neg:ExpirationTime>
        xsd:dateTime
    </wsag-neg:ExpirationTime> ?
    <wsag-neg:NegotiationInitiator>
          xsd:anyType
    </wsag-neg:NegotiationInitiator> ?
    <wsag-neg:NegotiationResponder>
          xsd:anyType
    </wsag-neg:NegotiationResponder> ?
    <wsag-neg:AgreementResponder>
          wsag-neg:NegotiationRoleType
    </wsag-neg:AgreementResponder> ?
    <wsag-neg:AgreementFactoryEPR>
          wsa:EndpointReferenceType
    </wsag-neg:AgreementFactoryEPR> ?
    <xsd:any /> *
</wsag-neg:NegotiationContext>
```

Listing 1: Content of a negotiation context

In general a negotiation process can either refer to the negotiation of new agreements or the renegotiation of an existing agreement. Therefore, the type of the negotiation must be defined in the negotiation context. Moreover, the negotiation context defines the roles of the parties participating in the negotiation process. The negotiation participators must acknowledge these parameters for the entire negotiation process.

/wsag-neg:NegotiationContext
This is the outermost document tag that defines the context of a negotiation. The negotiation context defines the type of the negotiation and the roles of the negotiation participators.

/wsag-neg:NegotiationContext/wsag-neg:NegotiationType
This REQUIRED element specifies the nature of the negotiation process.

/wsag-neg:NegotiationContext/wsag-neg:ExpirationTime
This OPTIONAL element specifies the lifetime of the negotiation instance. If specified, the negotiation instance is accessible until the specified time. After the lifetime of the negotiation instance has expired, the negotiation instance is no longer accessible.

/wsag-neg:NegotiationContext/wsag-neg:NegotiationInitiator
This OPTIONAL element identifies the initiator of the negotiation process. The negotiation initiator element can be an URI or an Endpoint Reference that can be used to contact the initiator. It can also be a distinguished name identifying the initiator in a security context.

/wsag-neg:NegotiationContext/wsag-neg:NegotiationResponder
This OPTIONAL element identifies the party that responds to the initiation of the negotiation process. This party implements the NegotiationFactory port type of this specification. This element can be an URI or an Endpoint Reference that can be used to contact the negotiation responder. It can also be a distinguished name identifying the negotiation responder in a security context.

/wsag-neg:NegotiationContext/wsag-neg:AgreementResponder
This REQUIRED element identifies the party in the negotiation process that acts on behalf of the agreement responder. This element can either take the value *NegotiationInitiator* or *NegotiationResponder*. The default value is *NegotiationResponder*, The party identified as agreement responder MUST provide a reference to the AgreementFactory (PendingAgreementFactory) it acts behalf of in the negotiation context by using the AgreementFactoryEPR element.

/wsag-neg:NegotiationContext/wsag-neg:AgreementFactoryEPR
This REQUIRED element identifies the endpoint reference of the agreement factory that can be used to create agreements based on the negotiated agreement offers. After an agreement offer was successfully negotiated, the party identified as agreement initiator MAY create a new agreement with the referenced factory.

/wsag-neg:NegotiationContext/{any}
Additional child elements MAY be specified to provide additional information but MUST NOT contradict the semantics of the parent element; if an element is not recognized, it SHOULD be ignored.

5.1.1.1 Negotiation Type

The negotiation type defines the nature of a negotiation. In general two types of negotiation exist; the negotiation of a new agreement and the re-negotiation of an existing agreement. The structure of the negotiation type is depicted in Listing 2.

```
<wsag-neg:NegotiationType>
    {
        <wsag-neg:Negotiation>
                <xsd:any /> *
        </wsag-neg:Negotiation>        |
        <wsag-neg:Renegotiation>
            <wsag-neg:ResponderAgreementEPR>
                wsa:EndpointReferenceType
            </wsag-neg:ResponderAgreementEPR>
            <wsag-neg:InitiatorAgreementEPR>
                wsa:EndpointReferenceType
            </wsag-neg:InitiatorAgreementEPR> ?
            <xsd:any /> *
        </wsag-neg:Renegotiation>
    }
</wsag-neg:NegotiationType>
```

Listing 2: Structure and content of the negotiation type

/wsag-neg:NegotiationType
This is the outermost element that encapsulates the negotiation type. It MUST either contain a *Negotiation* or *Renegotiation* element.

/wsag-neg:NegotiationType/wsag-neg:Negotiation
The existence of this element defines that the negotiation process is about the negotiation of a new agreement.

/wsag-neg:NegotiationType/wsag-neg:Negotiation/{any}
Additional elements MAY be used to carry critical extensions which control additional negotiation mechanisms. All extensions are considered mandatory, i.e. the responder MUST return a fault if any extension is not understood or the responder is unwilling to support the extension. The meaning of extensions and how to obey them is domain-specific and MUST be understood from the extension content itself.

/wsag-neg:NegotiationType/wsag-neg:Renegotiation
The existence of this element defines that the negotiation process is about the renegotiation of an existing agreement. The renegotiation of an existing agreement is a bilateral negotiation process between an agreement initiator and an agreement responder. The renegotiation element MUST include an endpoint reference to the responder agreement that is renegotiated. In a symmetric layout of the agreement port types this element MAY also contain the endpoint reference to the initiator agreement. The renegotiation element MAY contain domain specific data that can be used to control the negotiation process in a specific domain.

/wsag-neg:NegotiationType/wsag-neg:Renegotiation/wsag-neg:ResponderAgreementEPR
This REQUIRED element identifies the agreement responder copy of the original agreement that is renegotiated. The endpoint provided by the endpoint reference MUST implement the Agreement port type.

/wsag-neg:NegotiationType/wsag-neg:Renegotiation/wsag-neg:InitiatorAgreementEPR
This OPTIONAL element identifies the agreement initiator copy of the original agreement that is renegotiated. In a symmetrical layout of the agreement layer, the initiator and the responder of an agreement host an instance of the agreement. If a renegotiated agreement is created afterwards, both agreement instances must go into the state *Completed*. The endpoint provided by the endpoint reference MUST implement the Agreement port type.

/wsag-neg:NegotiationType/wsag-neg:Renegotiation/{any}
Additional elements MAY be used to carry critical extensions which control additional renegotiation mechanisms or creation mechanisms for renegotiated agreements. All extensions are considered mandatory, i.e. the responder MUST return a fault if any extension is not understood or the responder is unwilling to support the extension. The meaning of extensions and how to obey them is domain-specific and MUST be understood from the extension content itself.

5.2 Negotiation Offer

A negotiation process comprises the exchange of offers and counter offers. Counter offers are created based on existing offers. An initial offer is created on the basis of an Agreement Template. The structure of a negotiation offer is basically the same as the structure of an Agreement Template. Agreement templates are defined in the *Agreement Template and Creation Constraints* section of the WS-Agreement specification. However, a negotiation offer contains the additional elements *Negotiation Offer Context* and *Negotiation Constraints*.

5.2.1 Negotiation Offer Structure

In order to negotiate the content of an agreement, negotiation offers are exchanged between an agreement initiator and and an agreement responder. In case one of the negotiating parties receives a negotiation offer, this party

evaluates the offer and creates zero or more Counter Offers, which are then sent back to the negotiation participator. The basic structure of a negotiation offer is shown in Figure 2.

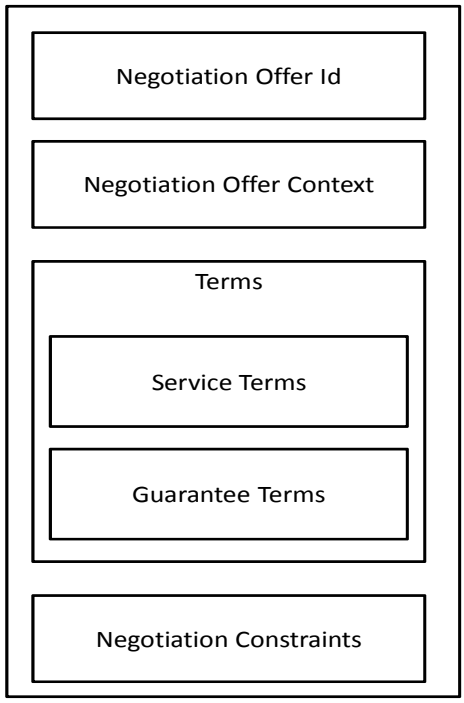

Figure 2: Structure of a negotiation offer

A negotiation offer has basically the same structure as an agreement template, but the negotiation offer also contains a Negotiation Offer Id and a Negotiation Context.

A Negotiation Offer also contains a Negotiation Constraints section. The Negotiation Constraints define the structure, valid ranges or distinct values that Service Terms may take in a counter offer. The Negotiation Constraints of a negotiation offer must hold true for every counter offer. In a negotiation process, however, the Creation Constraints MAY change during the advance of the negotiation. For example, if the negotiation initiator chooses one specific Service Term out of a set of Service Terms (ExactlyOne), a negotiation responder may adopt to this choice by changing the Creation Constrains section in a Counter Offer.

Negotiation Constraints are structurally identical to the Creation Constraints defined in an agreement template. Creation Constraints are defined in the section *Agreement Template and Creation Constraints* of the WS-Agreement specification.

The contents of a Negotiation Offer are of the form:

```
<wsag-neg:NegotiationOffer wsag-neg:OfferId="xs:string">
    <wsag-neg:NegotiationOfferContext>
            wsag-neg:NegotiationOfferContextType
    </wsag-neg:NegotiationOfferContext>
    <wsag:Name>
            xs:string
    </wsag:Name> ?
    <wsag:Context>
            wsag:AgreementContextType
    </wsag:Context>
    <wsag:Terms>
            wsag:TermCompositorType
    </wsag:Terms>
    <wsag-neg:NegotiationConstraints>
            wsag:ConstraintSectionType
    </wsag-neg:NegotiationConstraints>
</wsag-neg:NegotiationOffer>
```

Listing 3: Content of a negotiation offer

The following section describes the attributes and tags of a Negotiation Offer:

/wsag-neg:NegotiationOffer
This is the outermost document tag which encapsulates the entire negotiation offer.

/wsag-neg:NegotiationOffer/@wsag-neg:OfferId
The MANDATORY *OfferId* is the identifier of a specific Negotiation Offer. It MUST be unique for both parties in the context of a negotiation.

/wsag-neg:NegotiationOffer/wsag-neg:NegotiationOfferContext
The REQUIRED Negotiation Offer Context contains the metadata associated with this negotiation offer. This metadata comprises the id of the negotiation offer for which this offer is a counter offer, and an expiration time of this offer. Moreover, the negotiation offer context MAY include domain specific extensions in order to create advanced negotiation mechanisms or to control the negotiation process in a specific domain.

/wsag-neg:NegotiationOffer/wsag:Name
This is an OPTIONAL element is the name of the agreement to negotiate. It is described in the section "*Agreement Structure"* of the WS-Agreement specification.

/wsag-neg:NegotiationOffer/wsag:Context
This is a REQUIRED element in the Negotiation Offer. The agreement context SHOULD include parties to an agreement. Additionally, the agreement context contains various metadata about the agreement such as the duration of the agreement, and optionally, the template name from which the agreement is created. The structure of the agreement context is described in the section "*Agreement Context"* of the WS-Agreement specification.

/wsag-neg:NegotiationOffer/wsag:Terms
This element specifies the terms of an agreement that is negotiated. Both the structure of and the values of the agreement terms can be subject of the negotiation process. The agreement terms are described in the WS-Agreement specification in the section "*Agreement Structure*".

/wsag-neg:NegotiationOffer/wsag-neg:NegotiationConstraints
This REQUIRED element provides constraints on the values that the various terms may take in subsequent negotiation offers or in a concrete agreement. The Negotiation Constraints MUST hold true in any counter offer. Negotiation constraints are of the type *wsag:ConstraintSectionType*. This type is specified in the *Creation Constraints* section of the WS-Agreement specification.

5.2.2 Negotiation Offer Context

The REQUIRED element Negotiation Offer Context specifies the offer it is related to and the lifetime of an offer. Additionally, it may contain domain specific elements in order to provide negotiation extensions, e.g. to realize binding negotiation offers and compensation methods.

```
<wsag-neg:NegotiationOfferContext>

    <wsag-neg:CounterOfferTo>

            xs:string

    </wsag-neg:CounterOfferTo>

    <wsag:ExpirationTime>

            xs:dateTime

    </wsag:ExpirationTime> ?

    <wsag:Creator>

            wsag-neg:NegotiationRoleType

    </wsag:Creator>

    <wsag-neg:State>

            wsag-neg:NegotiationOfferStateType

    </wsag-neg:State>

    <xsd:any /> *

</wsag-neg:NegotiationOfferContext>
```

Listing 4: Content of a negotiation offer context

/wsag-neg:NegotiationOfferContext
This is the outermost tag that encapsulates the entire NegotiationOfferContext.

/wsag-neg:NegotiationOfferContext/wsag-neg:CounterOfferTo
The MANDATORY CounterOfferTo identifies the Negotiation Offer which was used to create a Counter Offer. When a Counter Offer is created, the CounterOfferTo specifies the OfferId of the Negotiation Offer that was used as a template. If a Negotiation Offer is created based on an Agreement Template, the CounterOfferTo refers to the TemplateId of the Agreement Template.

/wsag-neg:NegotiationOfferContext/wsag-neg:ExpirationTime
This REQUIRED element defines the lifetime of a negotiation offer. A negotiation participator MAY reference a negotiation offer during its lifetime and create counter offers to it.

/wsag-neg:NegotiationOfferContext/wsag-neg:Creator
This REQUIRED element identifies the party in the negotiation process that created this negotiation offer. Valid values for this element are *NegotiationInitiator* and *NegotiationResponder*.

/wsag-neg:NegotiationOfferContext/wsag-neg:State
This REQUIRED element contains the state of a specific negotiation offer.

The negotiation offer state indicates whether further negotiation is required for the current offer, counter offers must be in an acceptable state, or the current offer can be used to create an agreement. Additionally, each state MAY contain domain specific extensions to provide additional information to a negotiation offer. E.g. if an offer was rejected for some reason, the REJECTED state may contain information why this offer was rejected. This information can be used to optimize the negotiation process.

/wsag-neg:NegotiationOfferContext/{any}
Additional child elements MAY be specified to provide additional information, but the semantic of these elements MUST NOT contradict the semantics of the parent element; if an element is not recognized, it SHOULD be ignored.

5.2.3 Negotiation Offer States

During the negotiation process the content of Agreement Offers is negotiated before an agreement is created. A negotiated agreement is created by the party identified as Agreement Initiator in the negotiation context. A valid negotiated agreement offer MUST have the state Agreed when a new negotiated agreement is created. Figure 3 illustrates the states that negotiation offers can have and valid state transitions.

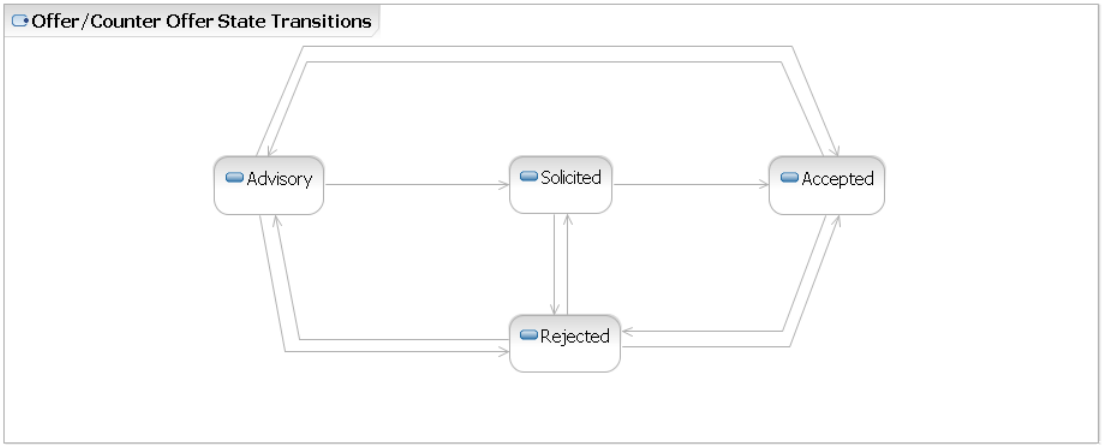

Figure 3: The state machine describes the counter offer state in relationship to the state of the offer it is related to.

Advisory State
The Advisory State identifies negotiation offers with which no further obligations associated. Offers in the Advisory State usually contain elements that are currently not specified. Therefore, these offers require further negotiation.

Solicited State
The Solicited State bears no obligations for an offer, but it requires that counter offers are either in the Accepted or the Rejected State. Solicited offers indicate that a negotiation participator wants to converge the negotiation process and requests only counter offers that can be accepted as is, e.g. where no further negotiation of the counter offers is required.

Accepted State
The Accepted State indicates that a negotiation participator accepts a negotiation offer as is. All details of a negotiation offer are specified and no further negotiation is required. However, since the negotiated offers are non-binding, there is no guarantee that a subsequent agreement is created. Augmented negotiation protocols may be created based on this specification to address binding negotiations.

Rejected State
If a negotiation offer is rejected, it is sent back to the inquiring party with the rejected state. The negotiation offer MAY contain a domain specific reason why it was rejected. Negotiation offers that are marked as rejected MUST NOT be used to create an agreement. However, they MAY be used to continue the negotiation process by taking into account the reason for rejecting the offer.

5.3 Creation of Negotiated and Renegotiated Agreements

Since Negotiation Offers extend the wsag:AgreementType, new agreement offers can easily be created based on a negotiated offer. These agreement offers can be used to create new agreements on the agreement layer. Moreover, since negotiated agreement offers bear no obligations for either of the negotiating parties, the creation of agreements based on a negotiated offer is totally independent of the negotiation process. This means that the negotiation layer and the agreement layer are totally decoupled and there is no need for additional extensions or control mechanisms to create new agreements based on negotiated offers. Nevertheless, it is still possible to design augmented negotiation protocols that tightly couple to the negotiation layer and the agreement layer by using the provided extension points.

While this is also true for renegotiated agreements, additional information is required when a renegotiated agreement is created. This information is stored in a Renegotiation Extension document and is passed to the createAgreement (createPendingAgreement) method of an Agreement Factory (PendingAgreementFactory) as Critical Extension. The Renegotiation Extension document contains the endpoint reference of the original agreement that is renegotiated and possibly domain specific extensions. The structure of a Renegotiation Extension document is shown in Listing 6. In case a renegotiated agreement is successfully created, the state of the original agreement(s) MUST change to *Complete*.

5.3.1 Negotiation Extension Document

A negotiation extension document MAY be passed to the createAgreement (createPendingAgreement) method of an AgreementFactory (PendingAgreementFactory) when an agreement is created on base of a negotiated offer. The negotiation extension document MAY be passed as critical or noncritical extension. The following describes the content of a negotiation extension document:

```
<wsag-neg:NegotiationExtension>

    <wsag-neg:ResponderNegotiationEPR>

            wsa:EndpointReferenceType

    </wsag-neg:ResponderNegotiationEPR> ?

    <wsag-neg:InitiatorNegotiationEPR>

            wsa:EndpointReferenceType

    </wsag-neg:InitiatorNegotiationEPR> ?

    <xsd:any /> *

</wsag-neg:NegotiationExtension>
```

Listing 5: Negotiation extension document to create agreements based on negotiated offers

/wsag-neg:NegotiationExtension
This is the outermost element of a negotiation extension document. This document is passed to an agreement factory (pending agreement factory) as a (non)critical extension in a *createAgreement* (*createPendingAgreement*) call.

/wsag-neg:NegotiationExtension/wsag-neg:ResponderNegotiationEPR
This OPTIONAL element specifies the endpoint reference to the negotiation instance of the negotiation responder. Implementations MAY use this reference to identify the negotiation process in which an agreement offer was negotiated.

/wsag-neg:NegotiationExtension/wsag-neg:InitiatorNegotiationEPR
This OPTIONAL element specifies the endpoint reference to the negotiation instance of the negotiation initiator. Implementations MAY use this reference to identify the negotiation process in which an agreement offer was negotiated.

/wsag-neg:NegotiationExtension/{any}
This OPTIONAL element contains domain specific extensions that can be used to realize augmented negotiation mechanisms.

5.3.2 Renegotiation Extension Document

The renegotiation extension document MUST be passed to the createAgreement (createPendingAgreement) method of an AgreementFactory (PendingAgreementFactory) as a critical extension when a renegotiated agreement is created. The following describes the content of a renegotiation extension document:

```
<wsag-neg:RenegotiationExtension>

    <wsag-neg:ResponderAgreementEPR>

            wsa:EndpointReferenceType

    </wsag-neg:ResponderAgreementEPR>

    <wsag-neg:InitiatorAgreementEPR>

            wsa:EndpointReferenceType

    </wsag-neg:InitiatorAgreementEPR> ?

    <wsag-neg:ResponderNegotiationEPR>

            wsa:EndpointReferenceType

    </wsag-neg:ResponderNegotiationEPR> ?

    <wsag-neg:InitiatorNegotiationEPR>

            wsa:EndpointReferenceType

    </wsag-neg:InitiatorNegotiationEPR> ?

    <xsd:any /> *

</wsag-neg:RenegotiationExtension>
```

Listing 6: Critical extensions to create a renegotiated agreement

/wsag-neg:RenegotiationExtension
This is the outermost element of a Renegotiation Extension document. This document is passed as a critical extension in a *createAgreement* call (*createPendingAgreement* call) to an agreement factory (pending agreement factoy). An agreement factory (pending agreement factory) MUST be able to understand all critical extensions that are contained in a *createAgreement* call (*createPendingAgreement* call). If this is not the case, the factory MUST return an error.

/wsag-neg:RenegotiationExtension/wsag-neg:ResponderAgreementEPR
This REQUIRED element specifies the endpoint of the original responder agreement instance. If an *Agreement Responder* decides to accept an offer for a renegotiated agreement, the Responder Agreement State MUST change to *Completed*.

/wsag-neg:RenegotiationExtension/wsag-neg:InitiatorAgreementEPR
This OPTIONAL element specifies the endpoint of the original initiator agreement instance. This element can is used in symmetric layouts of the agreement port type. If an *Agreement Responder* decides to accept an offer for a renegotiated agreement, the original Initiator Agreement State MUST change to *Completed*.

/wsag-neg:RenegotiationExtension/wsag-neg:ResponderNegotiationEPR
This OPTIONAL element specifies the endpoint reference to the negotiation instance of the negotiation responder. Implementations MAY use this reference to identify the negotiation process in which an agreement offer was negotiated.

/wsag-neg:RenegotiationExtension/wsag-neg:InitiatorNegotiationEPR
This OPTIONAL element specifies the endpoint reference to the negotiation instance of the negotiation initiator. Implementations MAY use this reference to identify the negotiation process in which an agreement offer was negotiated.

/wsag-neg:RenegotiationExtension/{any}
This OPTIONAL element contains domain specific extensions that can be used to realize augmented renegotiation mechanisms.

5.4 Negotiation Port Types and Operation

In this section a detailed description of the Negotiation Factory and the Negotiation port types is given. These port types can be used in different combinations in order to support a wide range of signaling scenarios. The presented signaling scenarios are not meant to cover all possible combinations of the port types. They are presented here to illustrate possible negotiation scenarios and how these scenarios are mapped to specific deployments of WS-Agreement Negotiation. Furthermore, the interaction of the negotiation layer and the agreement layer is discussed.

5.4.1 Simple client-server negotiation

The simple client-server negotiation is an asymmetric signaling scenario, where a server implements the Negotiation Factory and Negotiation port types. The negotiation process itself is driven by the client. The client initiates a negotiation by calling the server's *initiateNegotiation* operation of the Negotiation Factory. After a new negotiation is created, the client queries the available templates from the Negotiation Responder that serve as initial templates for a negotiation offer. It uses these templates to create new negotiation offers and sends these offers to the server via the *negotiate* method of the Negotiation port type. The server may create one or more counter offers for each offer that is part of the *negotiate* call. The server itself has a passive role in this negotiation process since it cannot actively influence the course of a negotiation, i.e. it can only react to negotiation requests. The process of negotiation is depicted in Figure 4.

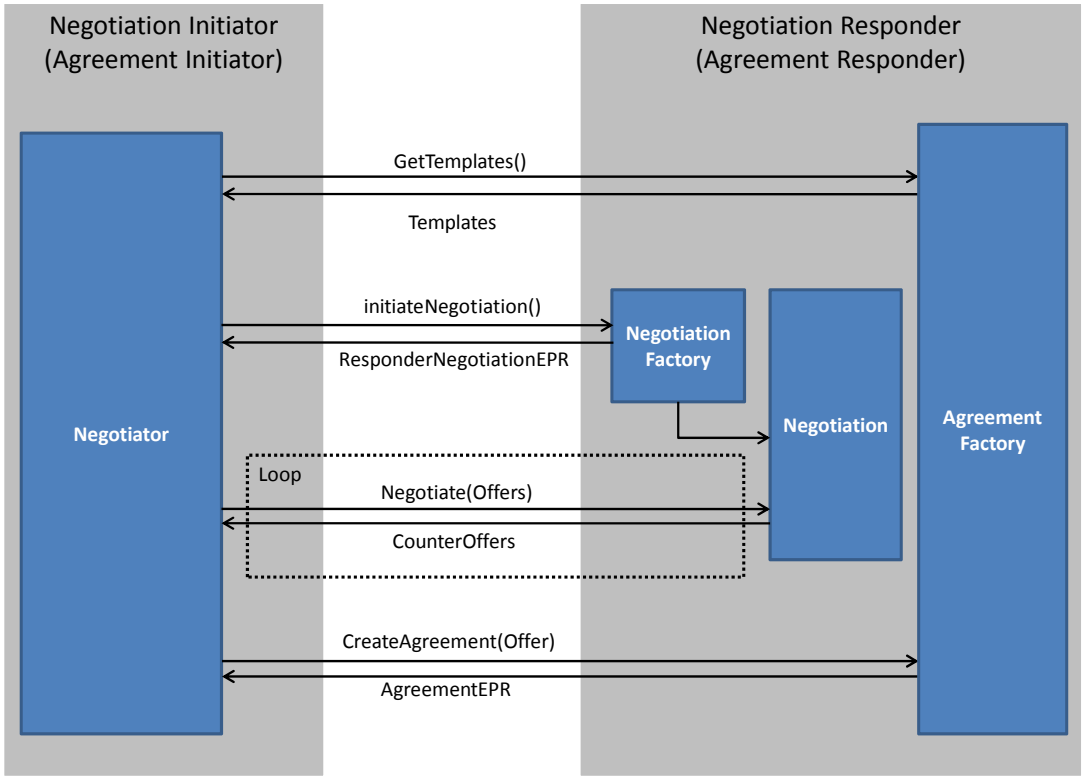

Figure 4: Asymmetric deployment of the WS-Negotiation port types

5.4.2 Bilateral negotiation with asymmetric agreement layer

In a bilateral negotiation both parties can actively participate in the negotiation process. Both parties implement the WS-Agreement *Negotiation* port types. The process of initiating a bilateral negotiation is as follows. The Negotiation Initiator creates a new negotiation instance that implements the WS-Agreement Negotiation port type. It then sends an *initiateNegotiation* request to the NegotiationFactory of the Negotiation Responder. The *initiateNegotiation* request includes an endpoint reference to the negotiation instance that was created beforehand. Moreover, it contains the negotiation context that defines the roles of each party in a negotiation, e.g. which party is the initiator and which is the responder of the agreements that are negotiated. In a bilateral negotiation process, the agreement templates that are used to create offers are provided by the agreement factory referenced in the negotiation context. The agreement initiator SHOULD query the available agreement templates from the agreement factory in order to create negotiation offers based on the provided templates. After a new Negotiation instance was created, the context of a negotiation MUST NOT change. Both parties participating in a negotiation process may actively send negotiation requests to the other party. It is not required that the initiator of a negotiation is also the initiator of the subsequent agreement. These roles may vary in different negotiation scenarios.

In the negotiation scenario depicted in Figure 5 the negotiation initiator is also the initiator of the subsequent agreements. It starts a negotiation process by retrieving the templates the responder provides. Then the initiator notifies the responder of the offers it is willing to negotiate by calling the responders *Advertise* method. Now the negotiation responder takes an active role in the negotiation process by sending offers to the initiator. After several rounds of negotiation the initiator may decide to create an agreement based on one of the negotiated offer. It therefore calls the *createAgreement* method of the negotiation responder. The input of the *createAgreement* operation includes a critical extension that is the context of the negotiated offer that the initiator uses to create an agreement.

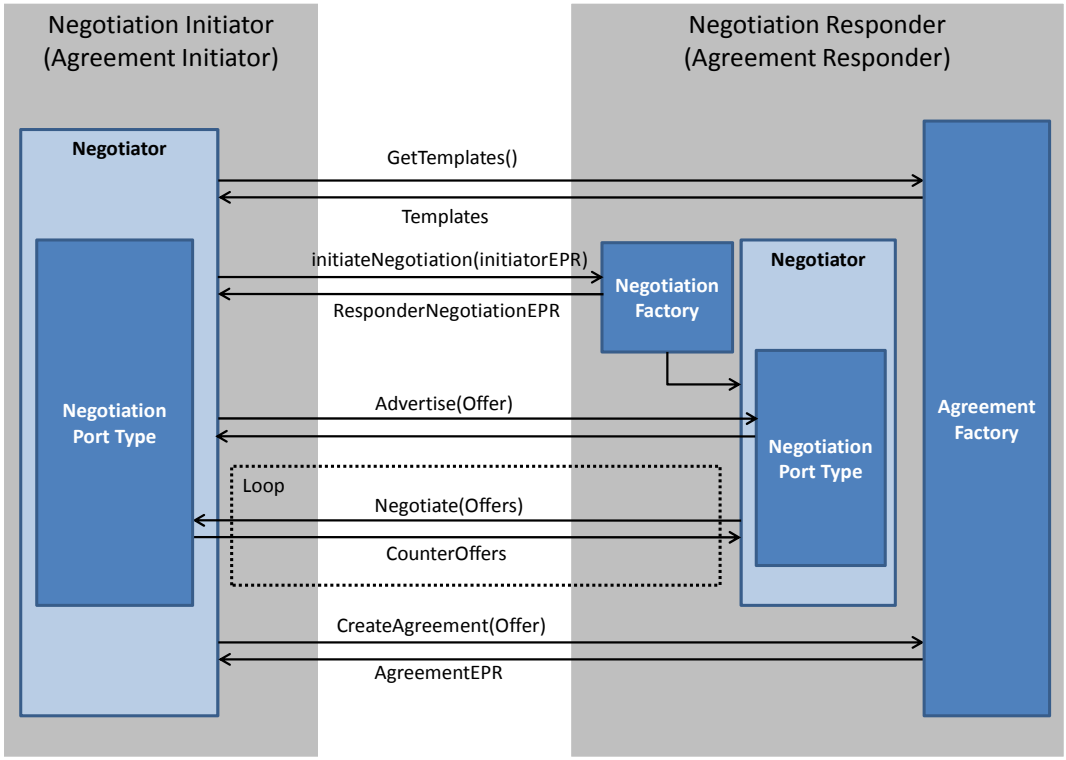

Figure 5: Symmetric deployment of WS-Agreement Negotiation, where the Negotiation Initiator is also the Agreement Initiator and the Negotiation Responder is the Agreement Responder. Both parties have an active role in the negotiation process.

5.4.3 Re-Negotiation of agreements with symmetric Agreement Layer

In general the renegotiation of an existing agreement follows the same signaling pattern as the negotiation of an agreement. If an existing agreement is renegotiated, the initiator of the original agreement SHOULD match the initiator of the renegotiated agreement, so that the roles and obligations match the original agreement. The roles and the responsibilities of the negotiating parties and are defined in the negotiation context, when a new negotiation is created. The negotiation context also includes an endpoint reference to the existing responder agreement. In a symmetric signaling scenario, the negotiation context MAY additionally include a reference to the original initiator agreement. After a new renegotiation process has been initiated, both

parties start to negotiate the contents of the agreement offer that can be used to create a renegotiated agreement. When they succeeded to negotiate a suitable offer, the initiator of the negotiated agreement creates a new the agreement by invoking the *createAgreement* (*createPendingAgreement*) method of the responder's Agreement Factory (Pending Agreement Factory) instance. When a renegotiated agreement is created, the original agreement must transition into the *Completed* state.

The layout of the agreement layer may be either asymmetric or symmetric. In case of a symmetric layout of the agreement layer, the renegotiated agreement initiator creates an instance of the renegotiated agreement before the *createAgreement* method (*createPendingAgreement* method) of the responder's agreement factory instance is invoked. This agreement MUST be in *Pending* state until the responder has either accepted or rejected the creation of the renegotiated agreement. After the initiator received the agreement responder's decision, the state of the Pending agreement is updated accordingly. When a renegotiated agreement is accepted, both parties MUST update the state of their original agreement instance to *COMPLETED*. Differences in the state of the original and renegotiated agreements are handled in domain specific manor, e.g. by applying state replication, different levels of escalation or dispute handling.

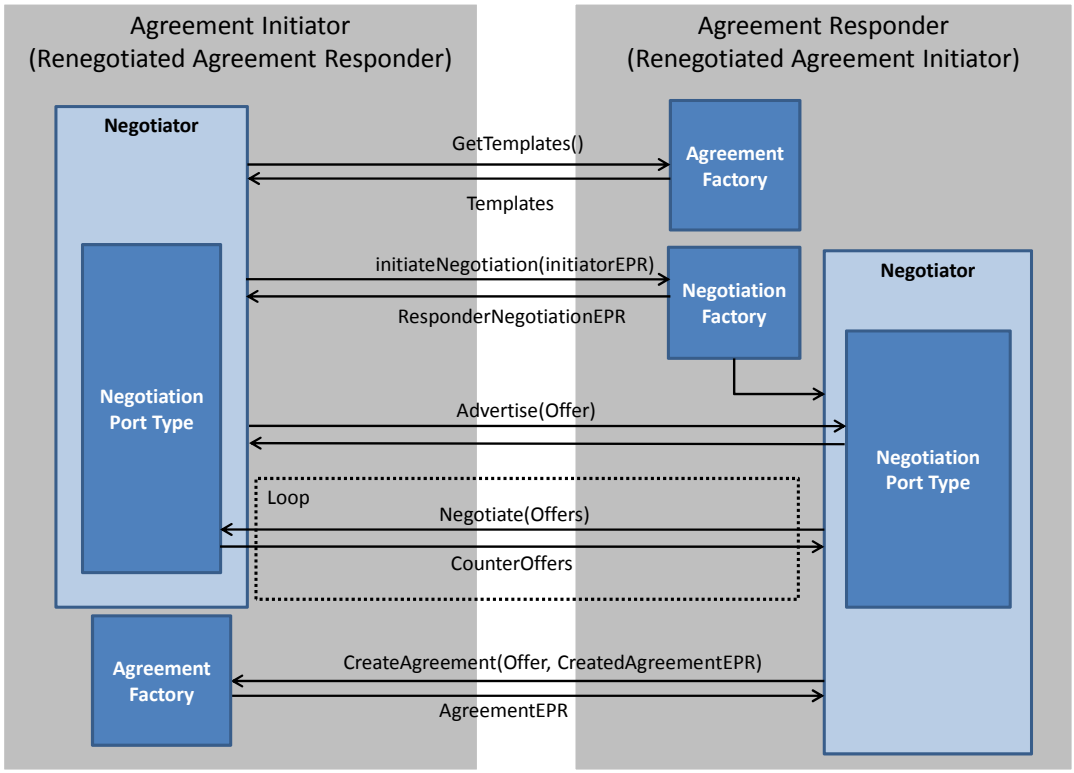

Figure 6: Symmetric signaling on the Negotiation and Agreement Layer. Both parties implement the WS-Agreement Negotiation and WS-Agreement port types. Moreover, both parties have their own instance of the original agreement. After the negotiation process, the responder of the original agreement creates the renegotiated agreement.

5.4.4 Negotiation Factory Port Type

5.4.4.1 Operation wsag-neg:InitiateNegotiation

The wsag-neg:InitiateNegotiation operation is used to create a new negotiation.

5.4.4.1.1 Input

```
<wsag-neg:InitiateNegotiationInput>

      <wsag-neg:InitiatorNegotiationEPR>

            <wsa:EndpointReference>

                  wsa:EndpointReferenceType

            </wsa:EndpointReference>

      </wsag-neg:InitiatorNegotiationEPR> ?

      <wsag-neg:NegotiationContext>

      …

      </wsag-neg:NegotiationContext>

      <wsag-neg:NoncriticalExtension>

            <xs:any> … </xs:any>

      </wsag-neg:NoncriticalExtension> *

      <xs:any> … </xs:any> *

</wsag-neg:InitiateNegotiationInput>
```

/wsag-neg:InitiateNegotiationInput
This is the outermost tag that encapsulates the input of an *initiateNegotiation* request.

/wsag-neg:InitiateNegotiationInput/wsag-neg:InitiatorNegotiationEPR
This OPTIONAL element identifies the endpoint of a *Negotiation* instance provided by the initiator of the negotiation. This endpoint MAY be used in symmetric deployment scenarios of the Negotiation port type in order to initiate a bilateral negotiation.

/wsag-neg:InitiateNegotiationInput/wsag-neg:NegotiationContext
This REQUIRED element defines the context of the negotiation that is initiated. All definitions supplied in the negotiation context apply to the whole negotiation that is initiated.

/wsag-neg:InitiateNegotiationInput/wsag-neg:NoncriticalExtension
Additional elements MAY carry non-critical extensions, which control augmented negotiation and agreement creation mechanisms. The responder MAY ignore non-critical extensions and behave as if they are not present. A responder SHOULD obey non-critical extensions if it is able and willing. The meaning of extensions and how to obey them is domain-specific and MUST be understood from the extension content itself.

/wsag-neg:InitiateNegotiationInput/xs:any##other
These optional elements MAY be used to carry critical extensions which control additional (re-)negotiation and agreement creation mechanisms. All extensions are considered mandatory, i.e. the responder MUST return a fault if any extension is not understood or the responder is unwilling to support the extension. The meaning of extensions and how to obey them is domain-specific and MUST be understood from the extension content itself.

5.4.4.1.2 Result

```
<wsag-neg:InitiateNegotiationOutput>

      <wsag-neg:CreatedNegotiationEPR>

            wsa:EndpointReferenceType

      </wsag-neg:CreatedNegotiationEPR>

      <xs:any> … </xs:any> *

</wsag-neg:InitiateNegotiationOutput>
```

/wsag-neg:InitiateNegotiationInput/wsag-neg:CreatedNegotiationEPR
This element is the EPR of the newly created negotiation. The created negotiation instance MUST bear the same context as provided in the input. This element MUST appear in an initiate negotiation response.

/wsag-neg:InitiateNegotiationInput/xs:any##other
The response MAY carry additional domain specific elements that are associated with the corresponding extensions of the input message.

5.4.4.1.3 Faults

A fault response indicates that the request for creating a negotiation was rejected and may also include domain specific reasons.

5.4.5 Negotiation Port Type

5.4.5.1 Operation wsag-neg:Negotiate

The wsag-neg:Negotiate operation is used to negotiate offers based on an offer-counter offer model.

5.4.5.1.1 Input

```
<wsag-neg:NegotiateInput>

      <wsag-neg:NegotiationOffer>

            wsag-neg:NegotiationOfferType

      </wsag-neg:NegotiationOffer>  +

      <xs:any> … </xs:any> *

</wsag-neg:NegotiateInput>
```

/wsag-neg:NegotiateInput/wsag-neg:NegotiationOffer
The input of the negotiation operation MUST contain at least one negotiation offer. All negotiation offers must refer to one of the templates provided by the agreement factory specified in the negotiation context.

/wsag-neg:NegotiateInput/{any}
The Negotiate input message MAY contain optional elements to control the negotiation process in a domain specific way. A responder MAY choose to ignore this content if it does not understand it or it is not willing to support the extensions. If responder is willing and able to understand these extensions it SHOULD support them.

5.4.5.1.2 Result

```
<wsag-neg:NegotiateOutput>

      <wsag-neg:NegotiationCounterOffer>

            wsag-neg:NegotiationOfferType

      </wsag-neg:NegotiationCounterOffer>  *

      <xs:any> … </xs:any> *

</wsag-neg:NegotiateOutput>
```

/wsag-neg:NegotiateOutput/wsag-neg:NegotiationCounterOffer
This element contains the created counter offers. Each counter offer SHOULD refer to an offer provided in the input message. For each provided offer zero or more counter offer MAY be created. The responder MUST NOT create any counter offer for offers that are in rejected state.

/wsag-neg:NegotiateOutput/{any}
The Negotiate output message MAY contain optional elements in order to include domain specific content to control the negotiation process. These extensions are in control of the extension provided in the input message.

5.4.5.1.3 Faults

A fault indicates that negotiation is not possible, the provided input is not valid, or another failure prevents negotiation. The fault may also include some domain specific reasons.

5.4.5.2 Operation wsag-neg:Terminate

This operation terminates a negotiation process, if permissible. All offers negotiated in the context of this negotiation process are invalidated.

5.4.5.2.1 Input

```
<wsag-neg:TerminateInput>

      <xs:any> … </xs:any> *

</wsag-neg:TerminateInput>
```

/wsag-neg:TerminateInput/{any}
: These OPTIONAL elements contain domain specific content that may be used to decide whether or not a termination is permissible.

5.4.5.2.2 Result

```
<wsag-neg:TerminateOutput>

</wsag-neg:TerminateOutput>
```

The result of the terminate operation does not contain any data.

5.4.5.2.3 Faults

This operation does not throw any faults.

5.4.5.3 Resource Property wsag-neg:NegotiationContext

The wsag-neg:NegotiationContext property is of the type wsag-neg:NegotiationContextType. It represents the context used to initiate the negotiation process. The content of the context is described in section Negotiation.

5.4.5.4 Resource Property wsag-neg:NegotiationOffer

The *wsag-neg:NegotiationOffer* property is of the type wsag-neg:NegotiationOfferType. The cardinality of this resource property is 0 to n. It represents a collection of all offers and counter offers exchanged in the context of this negotiation. Therefore, it has the function of a negotiation history. If an implementation is not capable or willing to support this feature, this list SHOULD be empty.

5.4.6 Offer Advertisement Port Type

The advertisement port type is used in order to advertise offers to a negotiation participator.

5.4.6.1 Operation wsag-neg:Advertise

The wsag-neg:Advertise operation is used to notify a negotiation participator of an offer where no counter offer is expected. Typical usage scenarios of the Advertise method are notification of new negotiation offers, the explicit rejection of a previously made offer, the response to a solicited offer, or the handover of the negotiation control.

5.4.6.1.1 Input

```
<wsag-neg:AdvertiseInput>

      <wsag-neg:NegotiationOffer>

            wsag-neg:NegotiationOfferType

      </wsag-neg:NegotiationOffer>  +

      <xs:any> … </xs:any> *

</wsag-neg:AdvetiseInput>
```

/wsag-neg:AdvertiseInput/wsag-neg:NegotiationOffer
: This element MUST appear in the input of the Advertise operation. The input may contain one or more negotiation offers of which a responder is notified.

5.4.6.1.2 Result

```
<wsag-neg:AdvertiseOutput>

</wsag-neg:AdvertiseOutput>
```

The result of the wsag-neg:Advertise operation is always empty.

5.4.6.1.3 Faults

A fault indicates that advertisement of offers for this specific negotiation resource is not possible and may also include some domain specific reasons.

6 Contributors

Dominic Battré
TU Berlin
Email: dominic.battre@tu-berlin.de

Francis Brazier
Delft University of Technology
Email: F.M.Brazier@tudelft.nl

Kassidy Clark
Delft University of Technology
Email: K.P.Clark@tudelft.nl

Michel Oey
Delft University of Technology
Email: m.a.oey@tudelft.nl

Alexander Papaspyrou
Dortmund University of Technology
Email: alexander.papaspyrou@tu-dortmund.de

Oliver Wäldrich
Fraunhofer SCAI
Email: oliver.waeldrich@scai.fraunhofer.de

Philipp Wieder
Dortmund University of Technology
Email: philipp.wieder@udo.edu

Wolfgang Ziegler
Fraunhofer SCAI
Email: Wolfgang.Ziegler@scai.fraunhofer.de

7 Glossary

Recommended but not required.

8 Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

9 Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

10 Full Copyright Notice

11 References

Note that only permanent documents should be cited as references. Other items, such as Web pages or working groups, should be cited inline (i.e., see the Open Grid Forum, http://www.ogf.org). References should conform to a standard such as used by IEEE/ACM, MLA, Chicago or similar. Include an author, year, title, publisher, place of publication. For online materials, also add a URL. It is acceptable to separate out “normative references,” as IETF documents typically do.

[RHJ08] Rosenberg, I., Heek, R., and Juan, A. An SLA Framework for the GT4 Grid Middleware, Collaboration and the Knowledge Economy: Issues, Applications, Case Studies (eChallenges 2008), 2008.

[SOZ+07] Seidel, J., Wäldrich, O., Ziegler, W., Wieder, P., and Yahyapour, R., Using SLA for resource management and scheduling – a survey, CoreGRID Technical Report TR-0096.

[BRADNER1] Bradner, S. Key Words for Use in RFCs to Indicate Requirement Levels, RFC 2119. March 1997.

11.1 Appendix 1: XML Schema and WSDL

11.1.1 Negotiation Types Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema

    elementFormDefault="qualified" attributeFormDefault="qualified"

    targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"

    xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"

    xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"

    xmlns:wsa="http://www.w3.org/2005/08/addressing"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">


    <xsd:import namespace="http://schemas.ggf.org/graap/2007/03/ws-
agreement"

        schemaLocation="agreement_types.xsd" />

    <xsd:import namespace="http://www.w3.org/2001/XMLSchema"

        schemaLocation="http://www.w3.org/2001/XMLSchema.xsd" />

    <xsd:import namespace="http://www.w3.org/2005/08/addressing"

        schemaLocation="http://www.w3.org/2005/08/addressing/ws-
addr.xsd"/>


    <xsd:element name="NegotiationContext"

         type="wsag-neg:NegotiationContextType" />

    <xsd:element name="NegotiationOffer"

         type="wsag-neg:NegotiationOfferType" />

    <xsd:element name="NegotiationCounterOffer"

         type="wsag-neg:NegotiationOfferType" />

    <xsd:element name="NegotiationOfferContext"

         type="wsag-neg:NegotiationOfferContextType" />

    <xsd:element name="NegotiationExtension"
```

```
        type="wsag-neg:NegotiationExtensionType" />
    <xsd:element name="RenegotiationExtension"
        type="wsag-neg:RenegotiationExtensionType" />

    <xsd:complexType name="NegotiationContextType">
      <xsd:sequence>
        <xsd:element name="NegotiationType"
            type="wsag-neg:NegotiationType" />
        <xsd:element name="ExpirationTime"
            type="xsd:dateTime" minOccurs="0" />
        <xsd:element name="NegotiationInitiator"
            type="xsd:anyType" minOccurs="0" />
        <xsd:element name="NegotiationResponder"
            type="xsd:anyType" minOccurs="0" />
        <xsd:element name="AgreementResponder"
            type="wsag-neg:NegotiationRoleType"/>
        <xsd:element name="AgreementFactoryEPR"
            type="wsa:EndpointReferenceType" minOccurs="0"/>
        <xsd:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:simpleType name="NegotiationRoleType">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="NegotiationInitiator" />
        <xsd:enumeration value="NegotiationResponder" />
      </xsd:restriction>
    </xsd:simpleType>
```

```
<xsd:complexType name="NegotiationType">
  <xsd:choice>
    <xsd:element name="Negotiation">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:any namespace="##other" processContents="lax"
               minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="Renegotiation">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ResponderAgreementEPR"
               type="wsa:EndpointReferenceType" />
          <xsd:element name="InitiatorAgreementEPR"
               type="wsa:EndpointReferenceType" minOccurs="0" />
          <xsd:any namespace="##other" processContents="lax"
               minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="NegotiationOfferType">
    <xsd:complexContent>
        <xsd:extension base="wsag:AgreementType">
            <xsd:sequence>
              <xsd:element name="NegotiationOfferContext"
```

```
                    type="wsag-neg:NegotiationOfferContextType"/>
                <xsd:element name="NegotiationConstraints"
                    type="wsag:ConstraintSectionType" />
            </xsd:sequence>
            <xsd:attribute name="OfferId" type="xsd:string" />
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="NegotiationOfferContextType">
  <xsd:sequence>
    <xsd:element name="CounterOfferTo"
        type="xsd:string"/>
    <xsd:element name="ExpirationTime"
        type="xsd:dateTime" minOccurs="0" />
    <xsd:element name="Creator"
        type="wsag-neg:NegotiationRoleType"/>
    <xsd:element name="State"
        type="wsag-neg:NegotiationOfferStateType" />
    <xsd:any namespace="##other" processContents="lax"
        minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="NegotiationOfferStateType">
  <xsd:choice>
    <xsd:element name="Advisory"
        type="wsag-neg:InnerNegotiationStateType"/>
    <xsd:element name="Solicited"
        type="wsag-neg:InnerNegotiationStateType"/>
```

```
        <xsd:element name="Accepted"
             type="wsag-neg:InnerNegotiationStateType"/>
        <xsd:element name="Rejected"
             type="wsag-neg:InnerNegotiationStateType"/>
      </xsd:choice>
    </xsd:complexType>

    <xsd:complexType name="InnerNegotiationStateType">
      <xsd:sequence>
        <xsd:any namespace="##other" processContents="lax"
             minOccurs="0" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="NegotiationExtensionType">
      <xsd:sequence>
        <xsd:element name="ResponderNegotiationEPR"
             type="wsa:EndpointReferenceType" minOccurs="0" />
        <xsd:element name="InitiatorNegotiationEPR"
             type="wsa:EndpointReferenceType" minOccurs="0" />
        <xsd:any namespace="##other" minOccurs="0"
             processContents="lax"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="RenegotiationExtensionType">
      <xsd:sequence>
        <xsd:element name="ResponderAgreementEPR"
             type="wsa:EndpointReferenceType" minOccurs="1" />
        <xsd:element name="InitiatorAgreementEPR"
```

```
                type="wsa:EndpointReferenceType" minOccurs="0" />
            <xsd:any namespace="##other" processContents="lax"
                minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>

</xsd:schema>
```

11.1.2 Negotiation Factory WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
    xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"
    xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
    xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
    targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation">

    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/rw-2"
        location="http://docs.oasis-open.org/wsrf/rw-2.wsdl"/>

    <wsdl:types>
        <xs:schema
            targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"
            xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"
            xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-
agreement"
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
```

```
            elementFormDefault="qualified"
            attributeFormDefault="qualified">

            <xs:import
namespace="http://www.w3.org/2005/08/addressing"
schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd"/>

            <xs:import
namespace="http://schemas.ggf.org/graap/2007/03/ws-agreement"
                schemaLocation="agreement_types.xsd" />

            <xs:include
schemaLocation="agreement_negotiation_types.xsd" />

            <xs:element name="InitiateNegotiationInput"
                type="wsag-neg:InitiateNegotiationInputType"/>
            <xs:complexType name="InitiateNegotiationInputType">
                <xs:sequence>
                    <xs:element ref="wsag-neg:NegotiationContext" />
                    <xs:element name="InitiatorNegotiationEPR"
                        type="wsa:EndpointReferenceType"
minOccurs="0" />
                    <xs:element name="NoncriticalExtension"
                        type="wsag:NoncriticalExtensionType"
                        minOccurs="0" maxOccurs="unbounded" />
                    <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>

            <xs:element name="InitiateNegotiationOutput"
```

```
                type="wsag-neg:InitiateNegotiationOutputType"/>
            <xs:complexType name="InitiateNegotiationOutputType">
                <xs:sequence>
                    <xs:element name="CreatedNegotiationEPR"
                        type="wsa:EndpointReferenceType"
                        minOccurs="1" maxOccurs="1" />
                    <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
    </wsdl:types>

    <wsdl:message name="InitiateNegotiationInputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:InitiateNegotiationInput" />
    </wsdl:message>

    <wsdl:message name="InitiateNegotiationOuputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:InitiateNegotiationOutput" />
    </wsdl:message>

    <wsdl:message name="InitiateNegotiationFaultMessage">
        <wsdl:part name="fault" element="wsag:ContinuingFault"/>
    </wsdl:message>

    <wsdl:portType name="NegotiationFactory">
        <wsdl:operation name="InitiateNegotiation">
            <wsdl:input
```

```
                    message="wsag-
neg:InitiateNegotiationInputMessage"/>

             <wsdl:output

                    message="wsag-
neg:InitiateNegotiationOuputMessage"/>

             <wsdl:fault name="ResourceUnknownFault"

                    message="wsrf-rw:ResourceUnknownFault" />

             <wsdl:fault name="ResourceUnavailableFault"

                    message="wsrf-rw:ResourceUnavailableFault" />

             <wsdl:fault name="NegotiationInitiationFault"

                    message="wsag-neg:InitiateNegotiationFaultMessage"
/>

         </wsdl:operation>

     </wsdl:portType>

</wsdl:definitions>
```

11.1.3 Negotiation WSDL

```
<?xml version="1.0" encoding="UTF-8"?>

<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

    xmlns:xs="http://www.w3.org/2001/XMLSchema"

    xmlns:wsa="http://www.w3.org/2005/08/addressing"

    xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"

    xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"

    xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"

    xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"

    targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation">


    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/rw-2"

        location="http://docs.oasis-open.org/wsrf/rw-2.wsdl"/>


    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/rpw-2"

        location="http://docs.oasis-open.org/wsrf/rpw-2.wsdl" />


    <wsdl:types>

        <xs:schema

            targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"

            xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-
agreement-negotiation"

            xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-
agreement"

            xmlns:wsa="http://www.w3.org/2005/08/addressing"

            elementFormDefault="qualified"

            attributeFormDefault="qualified">


            <xs:import
namespace="http://schemas.ggf.org/graap/2007/03/ws-agreement"
```

```
                schemaLocation="agreement_types.xsd" />


            <xs:include
schemaLocation="agreement_negotiation_types.xsd" />


            <xs:element name="NegotiationProperties"
                type="wsag-neg:NegotiationPropertiesType" />
            <xs:complexType name="NegotiationPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag-neg:NegotiationContext" />
                    <xs:element ref="wsag-neg:NegotiationOffer"
                        minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>


            <xs:element name="NegotiateInput"
                type="wsag-neg:NegotiateInputType"/>
            <xs:complexType name="NegotiateInputType">
                <xs:sequence>
                    <xs:element ref="wsag-neg:NegotiationOffer"
                        minOccurs="1" maxOccurs="unbounded" />
                    <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>


            <xs:element name="NegotiateOutput"
                type="wsag-neg:NegotiateOutputType"/>
            <xs:complexType name="NegotiateOutputType">
                <xs:sequence>
```

```
                    <xs:element ref="wsag-
neg:NegotiationCounterOffer"
                        minOccurs="0" maxOccurs="unbounded" />
                    <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>

            <xs:element name="TerminateInput"
                type="wsag-neg:TerminateInputType" />
            <xs:complexType name="TerminateInputType">
                <xs:sequence>
                    <xs:any processContents="lax" namespace="##other"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>

            <xs:element name="TerminateResponse"
                type="wsag-neg:TerminateOutputType" />
            <xs:complexType name="TerminateOutputType" />
        </xs:schema>
    </wsdl:types>

    <wsdl:message name="NegotiateInputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:NegotiateInput" />
    </wsdl:message>

    <wsdl:message name="NegotiateOuputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:NegotiateOutput" />
```

```
    </wsdl:message>

    <wsdl:message name="NegotiationFaultMessage">
        <wsdl:part name="fault"
            element="wsag:ContinuingFault"/>
    </wsdl:message>

    <wsdl:message name="TerminateNegotiationInputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:TerminateInput" />
    </wsdl:message>

    <wsdl:message name="TerminateNegotiationOuputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:TerminateResponse" />
    </wsdl:message>

    <wsdl:portType name="Negotiation"
                   wsrf-
rp:ResourceProperties="wsag:NegotiationProperties">

        <wsdl:operation name="Negotiate">
            <wsdl:input
                message="wsag-neg:NegotiateInputMessage" />
            <wsdl:output
                message="wsag-neg:NegotiateOuputMessage" />
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
            <wsdl:fault name="ResourceUnavailableFault"
                message="wsrf-rw:ResourceUnavailableFault" />
            <wsdl:fault name="NegotiationFault"
```

```
                message="wsag-neg:NegotiationFaultMessage" />
        </wsdl:operation>
        <wsdl:operation name="Terminate">
            <wsdl:input
                message="wsag-neg:TerminateNegotiationInputMessage"
/>
            <wsdl:output
                message="wsag-neg:TerminateNegotiationOuputMessage"
/>
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
            <wsdl:fault name="ResourceUnavailableFault"
                message="wsrf-rw:ResourceUnavailableFault" />
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```

11.1.4 Advertisement WSDL

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://www.w3.org/2005/08/addressing"
    xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
    xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-agreement-negotiation"
    xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
    xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
    targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-agreement-negotiation">

    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/rw-2"
        location="http://docs.oasis-open.org/wsrf/rw-2.wsdl"/>

    <wsdl:import namespace="http://docs.oasis-open.org/wsrf/rpw-2"
        location="http://docs.oasis-open.org/wsrf/rpw-2.wsdl" />

    <wsdl:types>
        <xs:schema
            targetNamespace="http://schemas.ogf.org/graap/2009/11/ws-agreement-negotiation"
            xmlns:wsag-neg="http://schemas.ogf.org/graap/2009/11/ws-agreement-negotiation"
            xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
            xmlns:wsa="http://www.w3.org/2005/08/addressing"
            elementFormDefault="qualified"
            attributeFormDefault="qualified">

            <xs:import
namespace="http://schemas.ggf.org/graap/2007/03/ws-agreement"
                        schemaLocation="agreement_types.xsd" />
```

```
            <xs:include
schemaLocation="agreement_negotiation_types.xsd" />

            <xs:element name="AdvertiseInput"
                type="wsag-neg:AdvertiseInputType"/>
            <xs:complexType name="AdvertiseInputType">
                <xs:sequence>
                    <xs:element ref="wsag-neg:NegotiationOffer"
                        minOccurs="1" maxOccurs="unbounded" />
                    <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>

            <xs:element name="AdvertiseOutput"
                type="wsag-neg:AdvertiseOutputType"/>
            <xs:complexType name="AdvertiseOutputType" />
        </xs:schema>
    </wsdl:types>

    <wsdl:message name="AdvertiseInputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:AdvertiseInput" />
    </wsdl:message>

    <wsdl:message name="AdvertiseOuputMessage">
        <wsdl:part name="parameters"
            element="wsag-neg:AdvertiseOutput" />
    </wsdl:message>
```

```
    <wsdl:message name="AdvertiseFaultMessage">
        <wsdl:part name="fault"
            element="wsag:ContinuingFault"/>
    </wsdl:message>

    <wsdl:portType name="Advertise">
        <wsdl:operation name="Advertise">
            <wsdl:input
                message="wsag-neg:AdvertiseInputMessage" />
            <wsdl:output
                message="wsag-neg:AdvertiseOuputMessage" />
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
            <wsdl:fault name="ResourceUnavailableFault"
                message="wsrf-rw:ResourceUnavailableFault" />
            <wsdl:fault name="Advertise"
                message="wsag-neg:AdvertiseFaultMessage" />
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```