GWD-R-P
GRAAP-WG

Authors:
Oliver Waeldrich (Editor), Fraunhofer SCAI
Dominic Battré, TU-Berlin
Francis Brazier, Delft University of Technology
Kassidy Clark, Delft University of Technology
Michel Oey, Delft University of Technology
Alexander Papaspyrou, Dortmund University of Technology
Philipp Wieder, Dortmund University of Technology
Wolfgang Ziegler, Fraunhofer SCAI

January 31, 2011

WS-Agreement Negotiation Version 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). It describes WS-Agreement Negotiation, an extension to the WS-Agreement Specification Version 1 (GFD.107). Distribution is unlimited.

Copyright Notice

Trademark

Abstract

This document describes Web Services Agreement Negotiation Specification (WS-Agreement Negotiation), a Web Services protocol for negotiating a valid agreement offer between two parties, such as between a service provider and consumer. A valid agreement offer may then be input to create an agreement using WS-Agreement (specified in GFD.107). WS-Agreement Negotiation can also be used to renegotiate an existing agreement that needs to be modified. Thus, WS-Agreement Negotiation provides an additional layer when creating agreements with WS-Agreement. WS-Agreement Negotiation provides an additional layer when creating agreements with WS-Agreement is using an extensible XML language for specifying the nature of the agreement offers, and agreement templates to facilitate discovery of compatible agreement parties and ease the process of creating valid agreement offers. Agreement templates conforming to the WS-Agreement specification including a negotiation context and a set of negotiation constraints are used for the negotiation. The specification consists of all schemas required for the negotiation and the necessary port types.
All information for creating, managing and monitoring an agreement, based on a valid negotiated agreement offer is not described in this specification but in the specification of WS-Agreement.

Contents

1 Introduction

In distributed service-oriented systems different services are offered by service providers and used by service consumers. Service consumers use these services as they are or compose (mash) them in order to provide new services with added functionality. Since services are often acquired on demand, service consumers need to predict the behavior of these services before they actually acquire them. This problem leads to a situation in which service consumers do not only have functional requirements for a service, but also demands regarding to the non-functional service properties, such as the average response time of a service, the service availability, or the average recovery time in case of failure. They need standardized ways of defining the required service properties, and guarantees of the service provider to deliver a service with the defined quality, capabilities to monitor the service properties at provisioning time, and enforcement mechanisms in case a service was not provided with the agreed service quality. Service level agreements are one approach to solve this problem. They are bilateral contracts between a service provider and a service consumer that describe the service to be provided and define guarantees regarding the quality this service is provided with.
WS-Agreement is one approach for using service level agreements in distributed service-oriented environments. It allows service consumers to dynamically create service level agreements with service providers in order to acquire services with a well-defined quality of service. Moreover, it defines the basic mechanisms to monitor the state of an agreement and to evaluate the guarantees that are associated with an agreement. WS-Agreement supports the agreement creation over a template mechanism. Service providers can offer their services in the form of agreement templates. These templates guide service consumers in the process of creating valid agreement offers. An agreement template may, for example, contain a number of alternative service descriptions, where each service description offers the same service with a different service quality. In that way the same service may be offered with 99.9%, 99% and 98% availability for example. The service consumer can choose the service offering that fulfills its requirements best and create a new agreement with the service provider. This approach is comparable to a super-market, where consumers choose the desired product out of a set of available products. Even though the template approach is sufficient for a wide range of application scenarios, there are still a number of scenarios that require more flexible and dynamic negotiation capabilities, for instance multi-round negotiation capabilities. A typical example being the negotiation of a service provisioning time in co-allocation scenarios, the renegotiation of existing agreements in order to cope with peaks in a service usage, or the negotiation of related service parameters such as the number of resources that are provided by a service and the price of the service. WS-Agreement Negotiation adds the required functionality for agreement negotiation on top of the WS-Agreement specification. It can therefore be used in conjunction with WS-Agreement without breaking existing systems.

In the WS-Agreement Negotiation model negotiation is done in the context of a separate negotiation process. A negotiation process represents a relationship between a service consumer and a service provider in order to

dynamically exchange information with the goal of creating a valid agreement offer that subsequently leads to an agreement. Negotiation processes are created by a *Negotiation Factory*, which implements the *Negotiation Factory Port Type*. A negotiation process is represented by a *Negotiation Instance*, which implements the *Negotiation Port Type* and optionally the *Advertisement Port Type*. The negotiation port type defines the basic properties of a negotiation instance, a method for exchanging offers and counter offers, and a method to terminate the negotiation process. The advertisement port type additionally specifies a method to notify a negotiation participant of a specific offer. The basic components involved in a negotiation process are depicted in Figure 1.

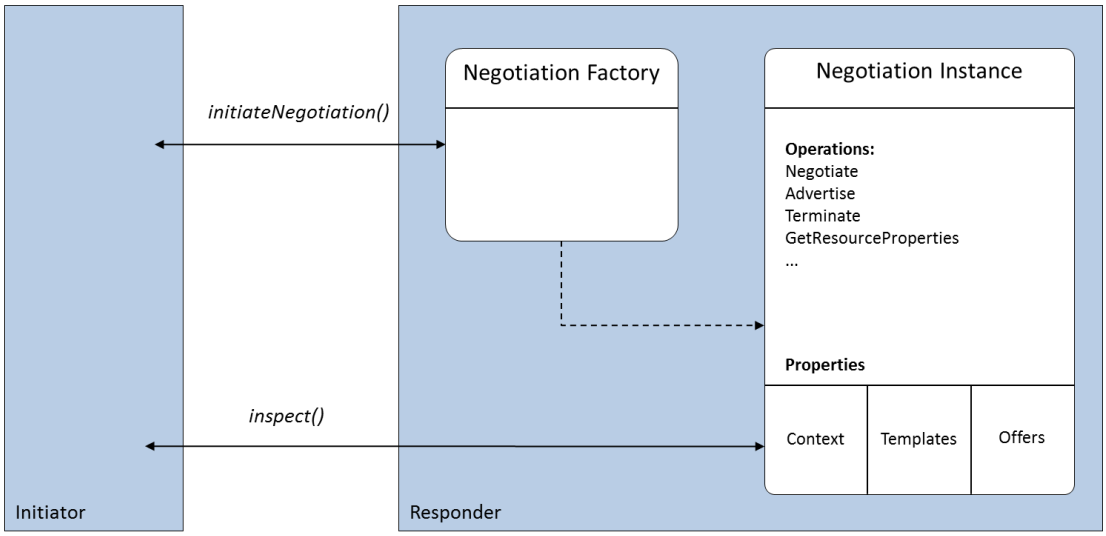

Figure 1: Overview of the WS-Agreement Negotiation components.

The remainder of this document is structured as follows. In section 1.1 the goals and non-goals of WS-Agreement Negotiation are described. Section 1.2 introduces the terms used in the specification and section 2 describes a set of negotiation use cases in more detail. The negotiation model is described in section 3. It consists of two parts, the description of the negotiation offer/counter offer model and the description of the layered negotiation model. In section 4 the properties of the negotiation instance are described. The structure of negotiation offers and counter offers is then described in section 5. Section 6 describes how the negotiation layer is finally coupled with the agreement layer and the creation process of negotiated and renegotiated agreements. Section 7 finally specifies the relevant port types and operations.

1.1 Goals and Requirements

The WS-Agreement Negotiation defines a set of requirements that are covered by the specification as well as a set of non-goals that are out of scope. The requirements and non-goals are described below:

Requirements

- *Must build on top of the WS-Agreement specification*

  WS-Agreement Negotiation must work seamlessly with WS-Agreement.

Therefore, the WS-Agreement language must be used to define negotiation and renegotiation offers and to express negotiation constraints. Moreover, the protocol must be defined as an extension to the WS-Agreement protocol. It must still be possible to use other negotiation protocols with an agreement factory.

- *Must allow negotiation of new and renegotiation of existing agreements*

  The protocol must specify the required interfaces to negotiate new and to renegotiate existing agreements. In the context of this specification, (re-)negotiation of agreements is considered to be a bilateral process, which results in a (re-)negotiated agreement. The specification must define the basic capabilities to create (re-)negotiated agreements based on (re-)negotiation offers.

- *Must provide both a symmetric and an asymmetric protocol*

  There is a wide number of negotiation scenarios, depending on whether a service consumer or a provider initiates the negotiation process, which party creates the negotiated agreement, and where the resulting agreement state is hosted. The same applies to renegotiation scenarios. The interfaces defined in this specification must therefore support symmetric and asymmetric protocol layouts in order to support various usage scenarios.

- *Must provide a simple negotiation state machine*

  The specification must provide a simple state machine that describes valid state transitions of negotiation/renegotiation offers.

- *Must support binding and non-binding negotiations*

  The specification must be usable in binding and non-binding (re-)negotiation scenarios. By default, this specification treats (re-)negotiation as a non-binding process (in case of renegotiation the agreement being renegotiated remains in force until superceeded by the renegotiated one). Binding negotiations are expected to be defined as an extension to this specification.

Out of Scope

- *Definition of compensation methods for negotiated offers*

  Even though binding (re-)negotiation of agreements is in principle foreseen by this specification, there is no compensation model defined for this type of negotiation. It is expected that such models will appear as domain specific extension to this specification.

- *Definition of Auction Protocols*

  This specification focuses on the bilateral (re-)negotiation of agreements. Since auction protocols are one-to-many negotiations they are regarded as alternative negotiation approach.

1.2 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 RF C 2119 [BRADNER1].

Negotiation
Negotiation is a process between an agreement initiator and an agreement responder to reach an acceptable agreement offer from an initial agreement template. Agreement offer negotiation is a non-binding, bi-lateral process that comprises exchange of information in order to find a consensus for acceptable agreement offers.

Negotiation Offer
A negotiation offer is a non-binding proposal for an agreement offer made by one negotiation party to another. Negotiation offers are used to dynamically exchange information in order to reach an acceptable agreement offer. Zero or more negotiation offers may precede a binding agreement offer as defined in the WS-Agreement specification. Negotiation offers describe the services of a SLA, the quality of service properties, and the associated guarantees. Negotiation offers may also contain negotiation constraints that restrict the negotiable terms and their value spaces.

Negotiable Template
Negotiable templates are provided by a negotiation participant in the context of a particular (re)negotiation process. They define which types of agreement offers can be negotiated, the basic structure of these offers, and the basic constraints that each offer must adhere to.

Negotiation Counter Offer
Negotiation offers that are created on the base of a previous negotiation offer are called Negotiation Counter Offers. Counter offers must adhere to the negotiation constraints of the offer they are related to. In a negotiation process each negotiation offer is either created on the base of an agreement template or on the base of another negotiation offer. In the context of this specification the term counter offer describes a negotiation offer that is based on another negotiation offer. It therefore reflects the relationship of a negotiation offer to the offer that it is based on.

Negotiated Offer
The term negotiated offer describes an offer that has reached the acceptable 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 a 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 a 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
The negotiation initiator is the party that initiates the negotiation process. It acts on behalf of the agreement initiator or the agreement responder. The negotiation initiator invokes the negotiation responder's initiateNegotiation method, which is defined in this specification.

Negotiation Responder
The negotiation responder is the party in a negotiation process that responds to an initiateNegotiation request. It acts on behalf of the agreement initiator or the agreement responder. The negotiation responder implements the NegotiationFactory and Negotiation port types defined in this specification.

Negotiation Participant
The negotiation participant is an entity that takes part in the negotiation process. The negotiation participant is either the negotiation initiator or the negotiation responder.

Negotiation Context
The negotiation context defines the type of the negotiation, identifies the negotiation participants, their roles and responsibilities, and optionally specifies additional domain specific negotiation parameters, such as maximum 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 originating negotiation offer, its expiration time, and its state. It may also contain domain specific extensions in order to define augmented negotiation protocols.

Negotiation Constraints
The negotiation constraints provide a method to control the negotiation process. A negotiation participant uses negotiation constraints in order to define structure and value spaces for compliant negotiation counter offers. Negotiation constraints are therefore used to express the requirements of a negotiation participant.

Negotiation Offer State
The negotiation offer state describes the specific state of a negotiation offer. It may include domain specific data that is used by the negotiation participants to exchange state-specific information and to advance the negotiation process. The reason for rejecting a negotiation offer is an example for such state-specific information.

1.3 Namespaces

The following is an example for XML or other code:

```
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

2 Use Cases

WS-Agreement Negotiation supports a large set of use cases. A typical negotiation example is the reservation of computational resources, which is described below.

2.1 Advance Reservation of Compute Resources

A service provider offers computational resources to its customers, which can be reserved for specific time frames. It provides a job submission service to access the reserved resources, and a portal application to manage the job submission service. The job submission service is implemented as a web service that provides the required methods for submitting and managing computational jobs, such as submit a job, start a job, query the state of a job, and cancel a job. These methods are exposed via the Web Service Description Language (WSDL). The portal application provides additional methods to manage the job submission service, such as updating the profiles of registered users, querying the current resource availability, querying usage data for the provided resources, deploying a new application, or managing the storage on the resources.

Agreements that comprise the advance reservation of computational resources define ongoing relationships between a resource provider and a resource consumer. They constitute the general conditions for jobs that are subsequently executed in the context of the agreement. The resource provisioning model is thereby implementation specific; whether resources are exclusively dedicated to a user, prediction models or preemption is used is up to the resource provider.

The computational resource provider offers available resources via an agreement template. The template includes the service description and a set of service levels possible service levels. The service description contains the specification of the available computational resources and the timeframe in which these resources are available. The offered 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 the offered resources in order to satisfy its needs. Moreover, the costumer can select the

desired service levels for resource availability, and availability and average response times of the job submission service and the portal application. The availability of the job submission service is for example 95%, 98%, 99% or 99.9%. It is defined as the probability that a request is processed within 15 seconds. For the average response time of the job submission service, the costumer may select a value of 0.5, 1, or 2 seconds and the number of requests per minute for which this guarantee must hold. These QoS parameters can be specified separately for the job submission service, the portal application, and the reserved resources. The pricing of the overall service is dependent on the selected computational resources and the selected QoS levels.

The template described provides many possibilities to parameterize the computational resource service. Moreover, it contains dynamic parameters, such as pricing, that are dependent on the resources and the QoS guarantees selected. Once the consumer filled in all its requirements, it sends the offer to the resource provider. The provider then checks whether it is capable to provide the requested service. In case the requested resources are available, the provider sends back a completed counter offer with the updated pricing information. The customer can now choose to create an agreement based on this negotiated offer. If the resource provider is not capable to fulfill the requirements stated in the negotiation offer, it can also send back a counter offer indicating an alternative service that can be provided instead. For example, the service 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 consumer may choose to accept the counter offer, to reserve only the 96 nodes that meet its requirements and to purchase the remaining capacity somewhere else. The process of filling in all required fields of a negotiation offer may 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.

3 WS-Agreement Negotiation Model

In this section we describe the WS-Agreement negotiation model. The model consists of two parts, the Negotiation Offer/Counter Offer model, and the layered architecture model. The Negotiation Offer/Counter Offer model describes the dynamic exchange of information in order to reach an acceptable agreement offer that can be used subsequently to create a new agreement, or to create a renegotiated agreement respectively. The layered architecture model describes the relationship of the WS-Agreement Negotiation layer to the WS-Agreement layer and the service layer.

3.1 Negotiation Offer/Counter Offer model

The WS-Agreement Negotiation Offer/Counter Offer model describes the dynamic exchange of information between the negotiation initiator and responder in order to agree on an acceptable agreement offer. A negotiation participant sends a negotiation offer to the other party, which in turn creates a counter offer for the negotiation offer received. Counter offers are always based on a negotiation offer that was previously received from the opposite negotiation party. The only exceptions are initial negotiation offers, which are based on a negotiation template. These initial offers can be regarded as counter offers to negotiation templates.

Each negotiation offer has an associated state, which reflects the view of the party that created that particular offer with respect to its acceptability. The possible state transitions that may occur when a counter offer is created for a particular offer are described in section 5.3.

An offer negotiation process may comprise multiple rounds of negotiation. In each negotiation round offers and counter offers are exchanged. The exchanged negotiation offers can therefore be modeled as a rooted tree with a negotiable template as root node. Each negotiation offer in this negotiation tree is a counter offer to its parent node. Children of the root node are initial negotiation offers, since they are based on a negotiable template. Leaf nodes are negotiation offers where either no further negotiation is required or that are in the terminal rejected state. If a negotiation offer does not require further negotiation it can be one of the following cases:

1. The negotiation offer is in the acceptable state and is used to create an agreement.

2. The negotiation participant does not follow this negotiation branch anymore, e.g. the participant decides that this negotiation branch does not lead to the expected results.

A negotiation process may include the exchange of negotiation offers that are based on different templates. A negotiation process can therefore comprise multiple negotiation trees. In the following example illustrates the concept of a negotiation tree in detail.

A negotiation initiator receives a negotiable template from the negotiation responder. Based on the negotiable template the initiator creates an initial negotiation offer with an offer id 1 (*OID 1*). This offer is then send to the negotiation responder using the responder's negotiate method. After the negotiation responder received the initial negotiation offer (*OID 1*), it examines the incoming offer (*OID 1*) and creates two counter offers with *OID 2* and *OID 3*. These counter offers are returned to the negotiation initiator as result of the negotiate call. The negotiation initiator processes the returned counter offers and decides that both counter offers do not lead to the desired agreement. The negotiation initiator therefore decides start a new negotiation branch by creating another negotiation offer (*OID 4*) based on the template *T1*. This offer

is again send to the negotiation responder that decides that this particular offer is unacceptable. The responder therefore creates a counter offer (*OID 5*), which is in the rejected state. Finally, the negotiation initiator creates a third negotiation branch by generating another negotiation offer based on *T1*. After several rounds of negotiation the negotiation responder returns a counter offer (*OID 9*), which is in the acceptable state. This offer is subsequently used by the negotiation initiator to create a new agreement. This process is depicted in Figure 2.

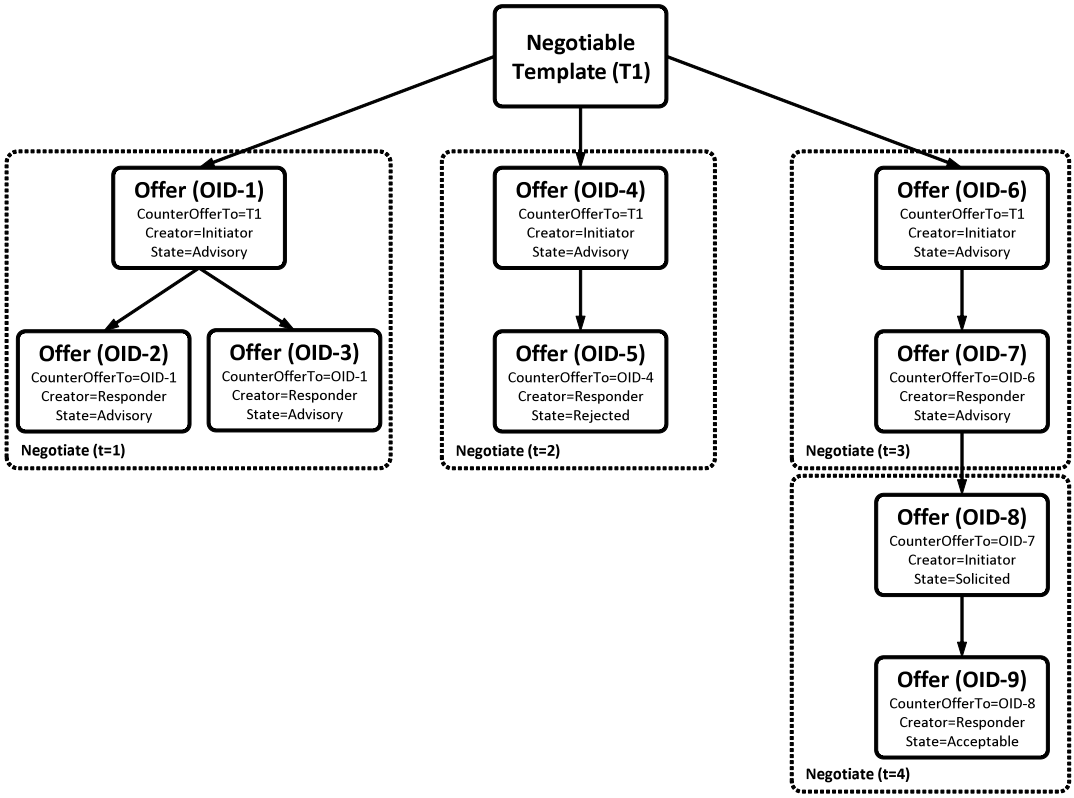

Figure 2: The exchange of multiple negotiation offers and counter offers results in the creation of a negotiation tree

The terms negotiation offer and negotiation counter offers both describe specific negotiation offers that are exchanged in a negotiation process. The distinction of what is a negotiation offer and what is a counter offer depends on the particular view of a negotiation participant. A negotiable template (the root node of a negotiation tree) is always considered as initial negotiation offer. All negotiation offers that are created based on this template are therefore counter offers to this template.
If a negotiation offer with *OID-1* was created based on a template *T1*, then *OID-1* is a counter offer to *T1*. If subsequently a negotiation offer *OID-2* is created based on offer *OID-1*, then *OID-2* is a counter offer to *OID-1*. In case the negotiation responder provides the negotiable template *T1*, it provides an initial negotiation offer to the negotiation initiator. The initiator receives the template *T1* and creates a counter offer with *OID-1* on the base of this template. This counter offer is sent to the negotiation responder. From the negotiation responder's point of view, *OID-1* is a new negotiation offer from the negotiation initiator. The responder therefore creates a counter offer with

OID-2. This process of creating counter offers based on previously received negotiation offers with the different viewpoints is depicted in Figure 3.

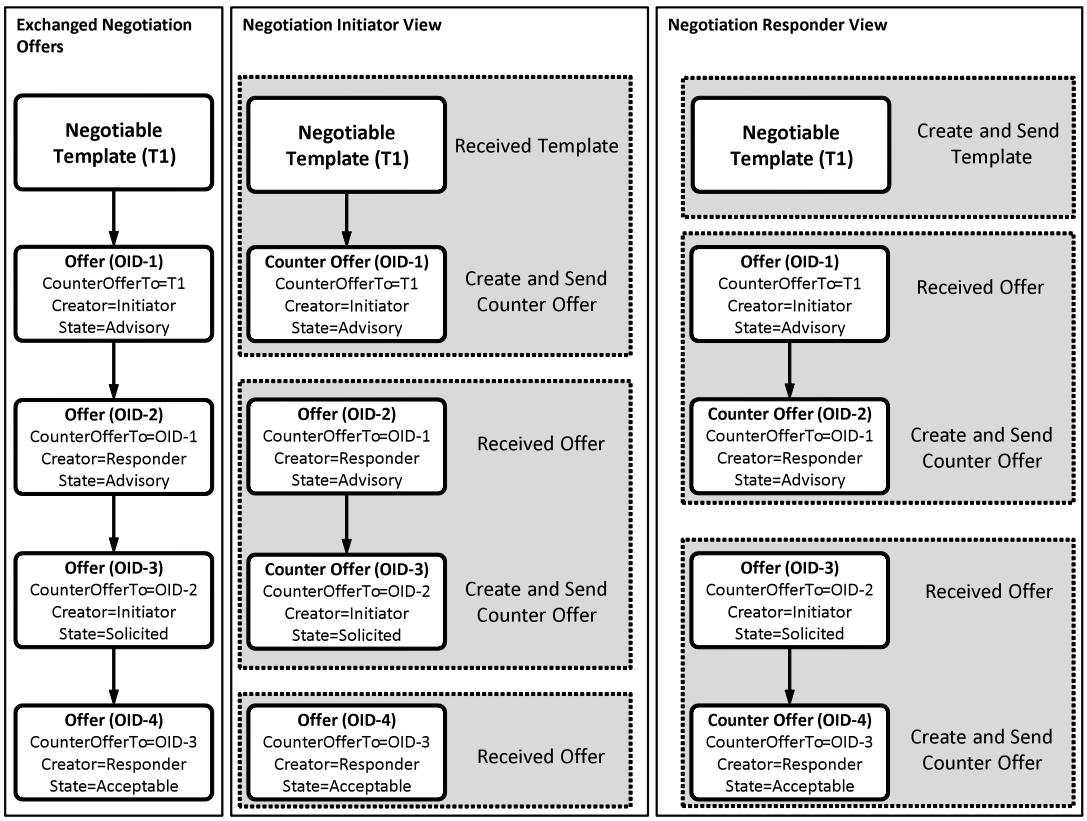

Figure 3: Different views on the negotiation process. An offer send by one negotiation participant is a counter offer to a previously received negotiation offer.

3.2 Layered architectural Model

The WS-Agreement Negotiation layered model consists of three layers, the negotiation layer, the agreement layer and the service layer. These layers are depicted in Figure 4. There is a clear separation between these layers. The negotiation layer sits on top of the agreement layer. It is therefore decoupled from the agreement layer and the service layer. By that, the negotiation layer may change independently of the agreement layer and can be replaced by another negotiation layer that might be better suited for a specific negotiation scenario.

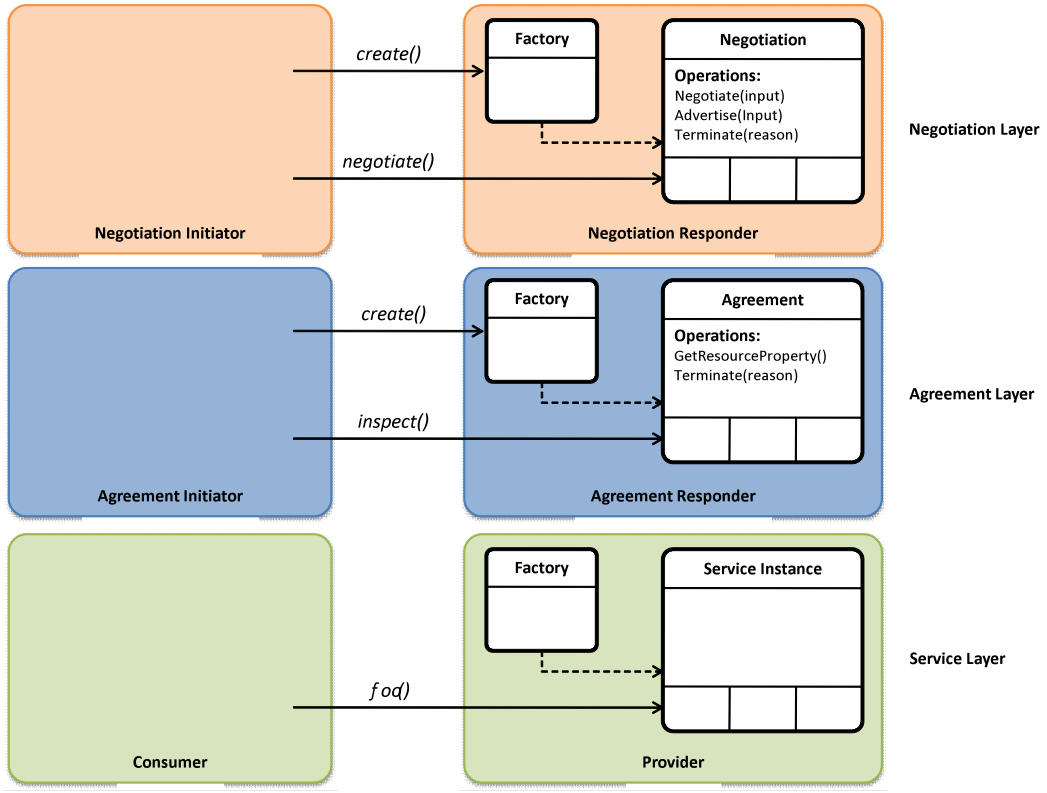

Figure 4: Conceptual overview of the layered negotiation model

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 negotiation offers and counter offers. Negotiation offers, as defined in this specification, are non-binding by nature. They do not comprise any promise of the agreement responder that it will create an agreement based on a negotiated offer. They only indicate the willingness of the two negotiating parties to subsequently create an agreement. However, it is possible to define languages that can be used in conjunction with this specification in order to realize binding negotiation processes.

Agreements based on negotiated offers are either created by calling the *createAgreement* or *createPendingAgreement* operation on the agreement responder's Agreement Factory port type, which is part of the responder's agreement layer.

Agreement layer

The Agreement layer provides the basic functionality to create and monitor agreements. It comprises the port types defined in the WS-Agreement specification. For details refer to the WS-Agreement specification [GDF107].

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, it may consist of multiple services. A resource provisioning service may for example comprise the provisioning of the specified resources and a monitoring service for the

provided resources. The services on the service layer are governed by the agreement layer.

4 Negotiation

The negotiation service defines a service instance that is used by the negotiation participants to dynamically exchange information in order to reach a common understanding of a valid agreement offer. During the negotiation process the participants exchange negotiation offers in order to indicate their negotiation goals and requirements. A negotiation instance may be limited in its lifetime or the maximum negotiation rounds. These limitations are defined in the negotiation context.

4.1 Negotiation Context

The negotiation context defines the roles of the negotiation participants, their obligations, and the nature of the negotiation process. Since 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

A negotiation instance either refers to the negotiation of new agreements or to the renegotiation of an existing agreement. The type of the negotiation must therefore be defined in the negotiation context. Moreover, the negotiation context defines the roles of the parties participating in the negotiation. The negotiation participants 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 participants.

/wsag-neg:NegotiationContext/wsag-neg:NegotiationType
This REQUIRED element specifies the type of the negotiation process and may contain optional, domain-specific parameters. The negotiation type can either be Negotiation or Renegotiation.

/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 negotiation lifetime has expired, this 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 initiateNegotiation request. The negotiation responder implements the NegotiationFactory port type defined in 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. It 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) in the negotiation context within the AgreementFactoryEPR element.

/wsag-neg:NegotiationContext/wsag-neg:AgreementFactoryEPR
This REQUIRED element identifies the endpoint reference of the agreement factory that is 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 agreement 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.

4.1.1 Negotiation Type

The negotiation type defines the nature of a negotiation instance. Two types of negotiation exist; negotiation of a new agreements and 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 indicates that the negotiation process comprises the negotiation of agreement offers.

/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 indicates that the negotiation process comprises the renegotiation of an existing agreement. Renegotiation of existing agreements is again a bilateral process between an agreement initiator and an agreement responder. The wsag-neg:Renegotiation element MUST include an endpoint reference to the responder agreement that is renegotiated. In a symmetric layout of the agreement port types the wsag-neg:Renegotiation element MAY also contain an endpoint reference to the initiator agreement. Additionally, the wsag-neg:Renegotiation element MAY contain domain specific data that can be used to control the negotiation process in a domain-specific way.

/wsag-neg:NegotiationType/wsag-neg:Renegotiation/wsag-neg:ResponderAgreementEPR
This REQUIRED element identifies the agreement responder's copy of the agreement that is renegotiated. The service identified by this endpoint reference MUST implement the Agreement port type. Once a renegotiated agreement is created, this agreement instance must change its state to *Completed*.

/wsag-neg:NegotiationType/wsag-neg:Renegotiation/wsag-neg:InitiatorAgreementEPR
This OPTIONAL element identifies the agreement initiator's copy of the agreement that is renegotiated. In a symmetrical deployment of the agreement layer, the agreement initiator and responder host an instance of the agreement. If a renegotiated agreement is created, both agreement instances must change their state to *Completed*. The service identified by this 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 augmented renegotiation mechanisms or creation mechanisms for renegotiated agreements. All extensions are considered mandatory, i.e. the agreement responder MUST return a fault if any extension is not understood or the responder is unwilling to support this extension. The meaning of the extensions and how to obey them is domain-specific and MUST be understood from the extension content itself.

5 Negotiation Offer

As mentioned before, negotiation comprises the dynamic exchange of information in form of negotiation offers and counter offers. An initial negotiation offer is created on the basis of an agreement template, while counter offers are created on the basis of negotiation offers received by a negotiation participant. The structure of a negotiation offer is basically the same as the structure of an agreement. Agreements are defined in the section *Agreement Structure* of the WS-Agreement specification. However, negotiation offers contain additional elements, namely the *Negotiation Offer Context* and *Negotiation Constraints*.

5.1 Negotiation Offer Structure

When a negotiation participant receives a negotiation offer, it evaluates the offer and creates zero or more counter offers, which are then sent back to the party that issued the negotiation offer. The basic structure of a negotiation offer is shown in Figure 5.

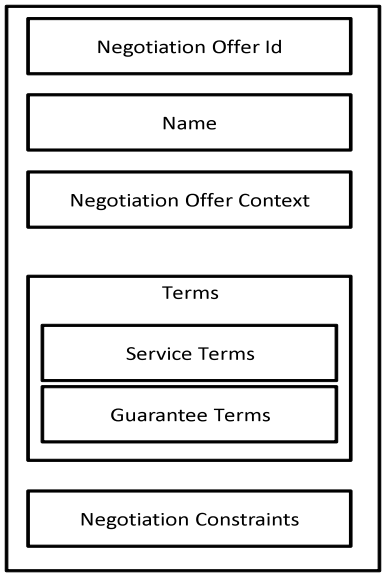

Figure 5: Structure of a negotiation offer

A negotiation offer has basically the same structure as an agreement, but it also contains a Negotiation Offer Id, a Negotiation Context, and a Negotiation Constraints section. It extends the *wsag:AgreementType* and therefore inherits the agreement name, agreement context, and the agreement terms.

Negotiation Constraints define restrictions for structure and values of negotiation counter offers. They must hold true for every counter offer. If this is not the case, the counter offer is rejected. Negotiation Constraints MAY change during the advance of a negotiation process. If, for example, the negotiation initiator chooses one specific service term out of a predefined set (e.g. in an ExactlyOne tag), the negotiation responder may adopt to this choice by changing the negotiation constrains in a counter offer.

Negotiation Constraints are structurally identical to Creation Constraints that are part of 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 element Negotiation Offer Context contains the metadata associated with a negotiation offer. The negotiation offer context contains the id of the originating negotiation offer, its expiration time, and its state. Moreover, the negotiation offer context MAY include domain specific extensions.

/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 REQUIRED element of a negotiation offer specifies the context of the agreement to negotiate. The agreement context SHOULD include parties to an agreement. Additionally, it 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 REQUIRED element specifies the terms of the 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 defines constraints on the structure and values that the agreement terms may take in subsequent negotiation offers. The Negotiation Constraints MUST hold true in any counter offer. Negotiation constraints are of the type *wsag-neg:NegotiationConstraintSectionType*. A negotiation constraint section MAY contain zero or more negotiation item constraints and zero or more free form constraints.

/wsag-neg:NegotiationConstraints/wsag-neg:Item
This OPTIONAL element defines a negotiation item constraint. It extends the *wsag:OfferItemType* which is specified in the section *Creation Constraints* of the WS-Agreement specification. A negotiation item constraint additionally defines two attributes, *Type* and *Importance*.

/wsag-neg:NegotiationConstraints/wsag-neg:Item/wsag-neg:Type
This REQUIRED attribute defines the type of the negotiation item constraint. Valid values are *Required* and *Optional*. If a required negotiation item constraint is violated by a counter offer, this counter offer MUST be rejected. If an optional negotiation item constraint is violated by a counter offer, this item constraint MAY be ignored, depending on the domain specific negotiation strategy. The default value of this attribute is *Required*.

/wsag-neg:NegotiationConstraints/wsag-neg:Item/wsag-neg:Importance
This OPTIONAL attribute defines the importance of a negotiation item constraint. It is intended to be used in conjunction with optional negotiation item constraints. Implementation MAY use this attribute in order to specify the importance of different optional negotiation item constraints. It is therefore possible to implement negotiation strategies that minimize the overall utility of violated optional constraints.

/wsag-neg:NegotiationConstraints/wsag-neg:Constraint
This OPTIONAL element defines a free-form negotiation constraint analog to free-form constrains as specified in the WS-Agreement specification.

5.2 Negotiation Offer Context

The negotiation offer context contains the metadata of a negotiation offer. It refers to the originating negotiation offer, defines the offer expiration time, and the offer state. 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 this counter offer. When a negotiation offer was used to create this offer, the CounterOfferTo specifies the OfferId of the originating negotiation offer. When an agreement template was used to create this offer, the CounterOfferTo refers to the TemplateId of the originating template.

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

/wsag-neg:NegotiationOfferContext/wsag-neg:Creator
This REQUIRED element identifies the party 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 negotiation offer. The negotiation offer state indicates whether further negotiation is required. Negotiation offers must be in the ACCEPTABLE state in order to create an agreement based on it. Each negotiation offer state MAY contain domain specific extensions. E.g. if an offer was rejected for some reason, the REJECTED state may contain information on 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.

5.3 Negotiation Offer States

The negotiation of an agreement offer precedes the final agreement creation process. The party that is defined as agreement initiator in the negotiation context is responsible of creating the agreement. A valid negotiated agreement offer SHOULD be in the ACCEPTABLE state when the agreement is created. Figure 6 shows the possible states of negotiation offers along with valid state transitions.

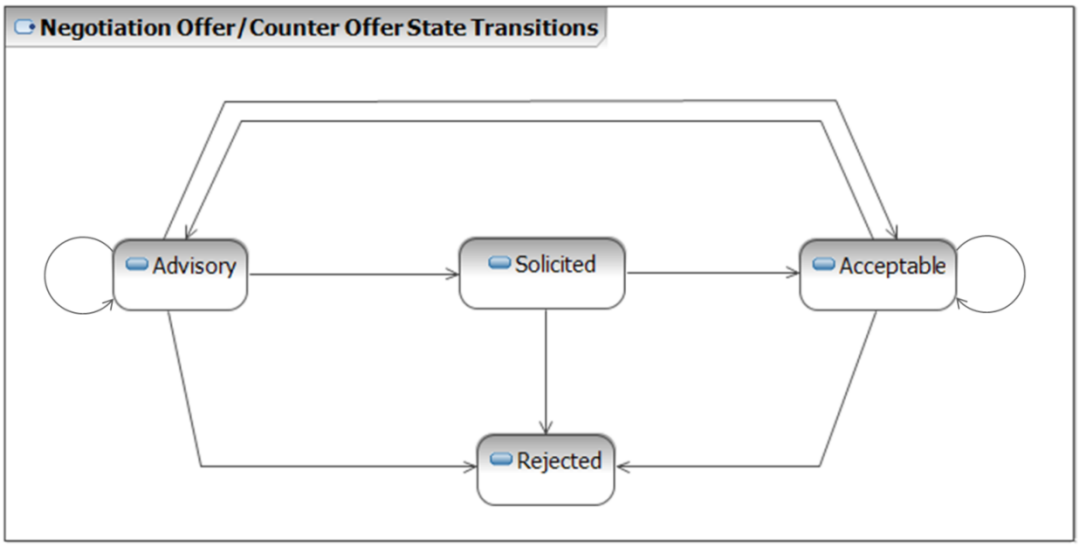

Figure 6: The state machine describes the states of a counter offer in relationship to the state of the offer it refers to.

Advisory State
The ADVISORY state identifies negotiation offers which have 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
Solicited offers indicate that a negotiation participant wants to converge the negotiation process. The SOLICITED state bears no obligations for an offer, but it requires that counter offers are either in the ACCEPTABLE or the REJECTED state.

Acceptable State
The ACCEPTABLE state indicates that a negotiation participant is willing to accept 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, a counter offer is sent back to the inquiring party with the REJECTED state. All terms SHOULD be the same as in the original offer the counter offer refers to. The counter 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.

Extension of Negotiation Offer States
Each state element MAY contain additional child elements to provide domain specific information. This information can be used to optimize the negotiation process. If this information is not understood, it SHOULD be ignored.

5.4 Negotiation Offer State Transitions

The state model abstractly describes the possible state transitions that can occur when a counter offer is created for a negotiation offer. This means that the state of each child node in a negotiation tree must be a valid state transition with respect to its parent node's state. Since negotiation offers and counter offers are exchanged between the negotiation participants over time, this section describes how exactly the state model maps to the exchanged negotiation offers.

The negotiation model allows negotiating multiple negotiation offers at one time. A negotiation initiator may for example create three negotiation offers (*OID-1, OID-2, OID-3*) based on a negotiable template *T1*. In a first negotiation iteration (t=1) these negotiation offers are sent to the negotiation responder in a single negotiate request. The responder creates counter offers for each of the received offers. For the negotiation offer *OID-1* the responder creates two counter offers (*OID-4, OID-5*) which are in the advisory state. The negotiation offer *OID-2* is rejected. The negotiation responder therefore creates a counter offer (*OID-6*), which is in the rejected state. For the negotiation offer *OID-3*, the responder creates one counter offer (*OID-7*) which again is in the advisory state. All states of the counter offers are valid state transitions regarding to the states of the offers they are based on. The counter offers are returned to the negotiation initiator as result of the negotiate call. The negotiation initiator analyses the counter offers received and decides to continue the negotiation process based on the offer *OID-7*. It therefore creates a new negotiation offer (*OID-8*). This time the negotiation offers is in the solicited state, which requires that counter offers are either in the acceptable state or in the rejected state. Negotiation offer OID-8 is then sent in a second negotiation iteration to the negotiation responder, again using the negotiate method. This time the responder decides to accept negotiation offer *OID-8*. It therefore creates a counter offer (*OID-9*) which is in the acceptable state. This process is depicted in Figure 7.

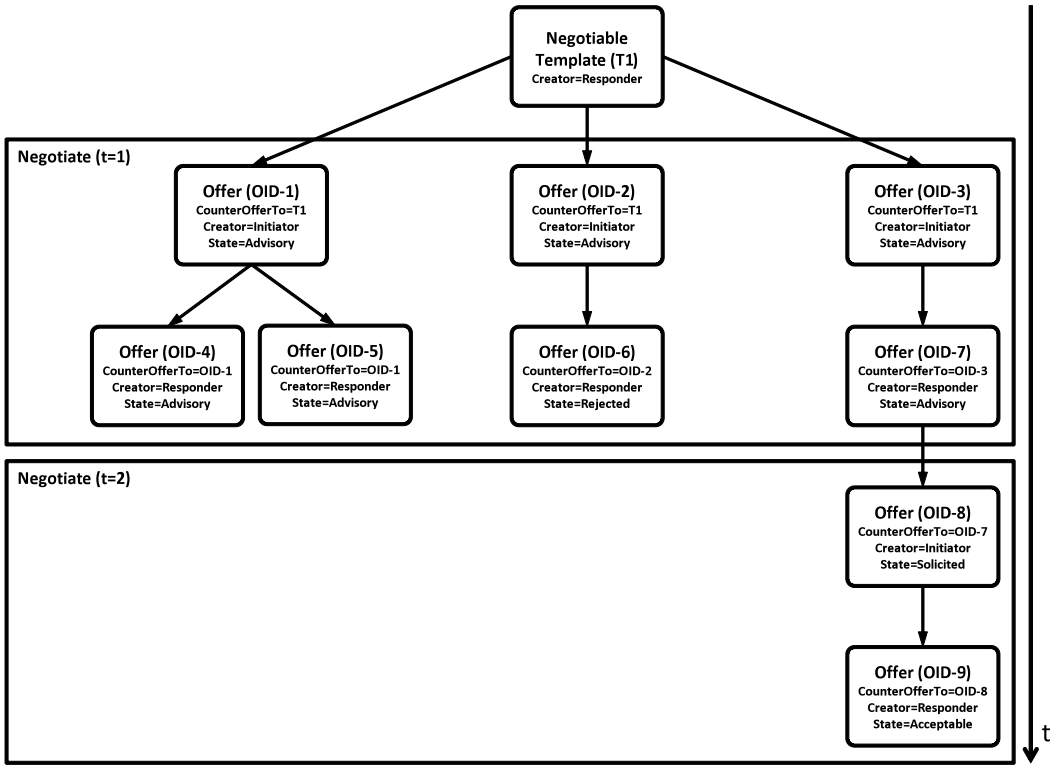

Figure 7: State transitions for parallel negotiation of multiple offers

In a second scenario, the negotiation initiator again creates a negotiation offer *OID-1* based on template *T1*. It sends the negotiation offer in the first negotiation iteration to the responder. The negotiation responder creates two counter offers (*OID-2, OID-3*) for *OID-1* and returns them. The initiator then decides to follow the negotiation process based on offer *OID-3*. It creates a negotiation offer (*OID-4*) and sends it to the responder in the second negotiation iteration. The responder analyses the offer and decides to reject it. It creates a counter offer (*OID-5*) and returns it as result of the second negotiation iteration. Negotiation offer *OID-5* additionally contains a domain specific rejection reason. The negotiation initiator MAY use this information to create a new negotiation offer (*OID-6*), taking the rejection reason into account. The offer *OID-6* MUST NOT be based on the rejected offer *OID-5,* since the negotiation responder already indicated that it is not willing to follow this negotiation branch anymore. Instead *OID-6* is a counter offer to *OID-3*, which is the parent of the rejected offer *OID-4*. The negotiation offer OID-6 is sent in a last iteration to the negotiation responder, which finally decides to accept it. This process is illustrated in Figure 8.

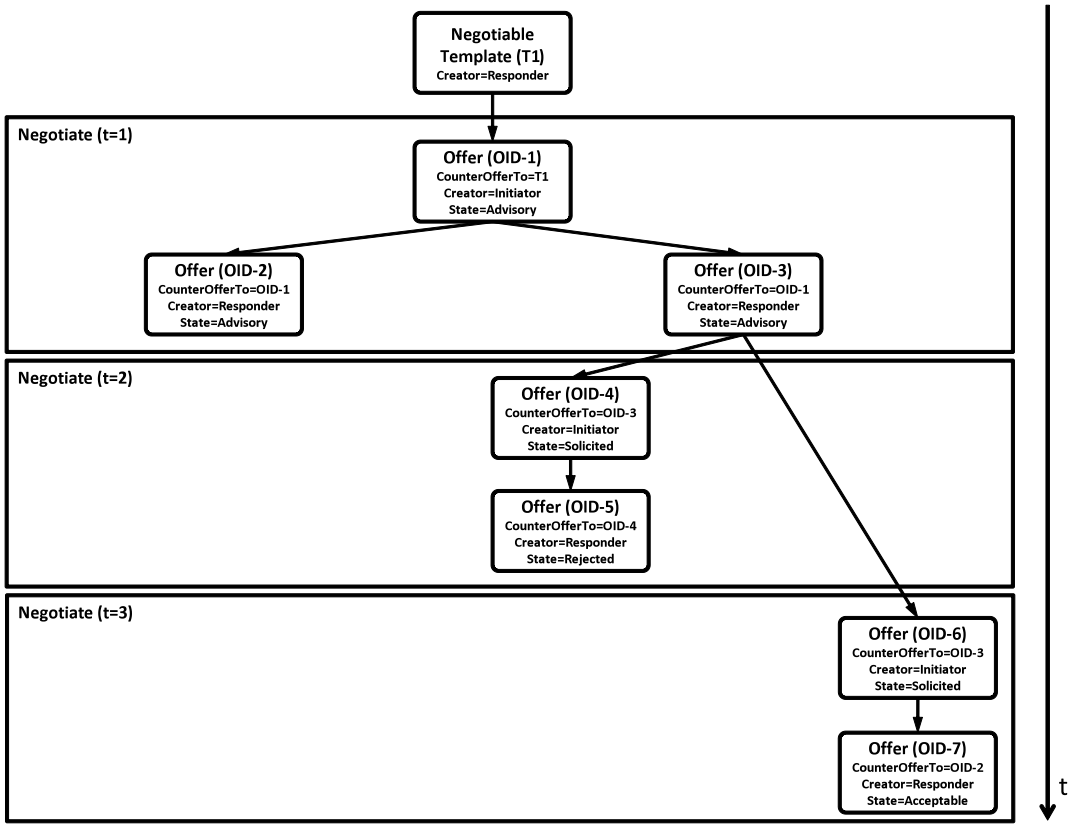

Figure 8: Creation of counter offers taking rejection reasons into account.

6 Creation of Negotiated and Renegotiated Agreements

Negotiation Offers extend the wsag:AgreementType. They can therefore easily be converted into agreement offers. These agreement offers are then used on the agreement layer to create new agreements. Since in a non-binding negotiation scenarios negotiated offers do not bear any obligations for either negotiating party, the creation of agreements based on such a negotiated offer is in principle independent of the negotiation process. The negotiation layer and the agreement layer are therefore completely decoupled and there is no need for additional extensions or control mechanisms for creating 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*.

6.1 Negotiation Extension Document

A negotiation extension document SHOULD be passed to the createAgreement (createPendingAgreement) method of an AgreementFactory (PendingAgreementFactory) when an agreement is created on the base of a negotiated offer. The negotiation extension document SHOULD be passed as critical 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> ?

    <wsag-neg:NegotiationOfferContext>

              wsag-neg:NegotiationOfferContextType

    </wsag-neg:NegotiationOfferContext>

    <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 SHOULD be passed to an agreement factory (pending agreement factory) as a critical extension in the *createAgreement* (*createPendingAgreement*) method.

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

/wsag-neg:NegotiationExtension/wsag-neg:NegotiationOfferContext
This REQUIRED element specifies the negotiation offer context for this agreement offer. It MUST refer to a valid negotiation offer where this agreement offer is a counter offer to.

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

6.2 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> ?

    <wsag-neg:NegotiationOfferContext>

              wsag-neg:NegotiationOfferContextType

    </wsag-neg:NegotiationOfferContext>

    <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 to an agreement factory (pending agreement factory) as a critical extension in a *createAgreement* call (*createPendingAgreement* call). 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 reference to the original instance of the responder agreement. If an *Agreement Responder* decides to accept an offer for a renegotiated agreement, the state of this agreement MUST change to *Completed*.

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

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

/wsag-neg:NegotiationExtension/wsag-neg:NegotiationOfferContext
This REQUIRED element specifies the negotiation offer context for this agreement offer. It MUST refer to a valid negotiation offer where this agreement offer is a counter offer to.

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

7 Negotiation Port Types and Operation

This section describes the Negotiation Factory and the Negotiation port types in detail. These port types can be used in different combinations to support a wide range of signaling scenarios. The examples are not meant to cover all possible combinations of the port types. They illustrate possible signaling scenarios and show how these scenarios are mapped to specific deployments of WS-Agreement Negotiation port types. Furthermore, the interaction of the negotiation layer and the agreement layer is discussed.

7.1 Simple Client-Server Negotiation

The simple client-server negotiation represents an asymmetric signaling scenario. The server domain implements the Negotiation Factory, Negotiation, Agreement Factory, and Agreement port types. The negotiation process is driven by the client. In the first step the client initiates a new negotiation process by calling the server's *initiateNegotiation* operation. The server returns an endpoint reference to a new negotiation instance. The client uses this EPR for the subsequent negotiation process. In the next step the client queries the negotiable templates from the new created negotiation instance and selects the template it wants to negotiate an SLA for. Moreover, the client creates an initial negotiation offer based on the selected template. This offer is then sent to the negotiation instance by calling the server's *Negotiate* method. The server creates one or more counter offers for the negotiation offer received and sends them back to the client. The client chooses the counter offer that fulfills its requirements best and creates a new agreement with the server by calling its *createAgreement* method. The client sends a NegotiationExtensionDocument along with the createAgreement-request in order to identify the originating negotiation instance and the negotiation offer that resulted in this agreement offer.

In this scenario, the server has a passive role. It is not in control of the negotiation process, i.e. it only reacts to negotiation requests. The negotiation process is depicted in Figure 9.

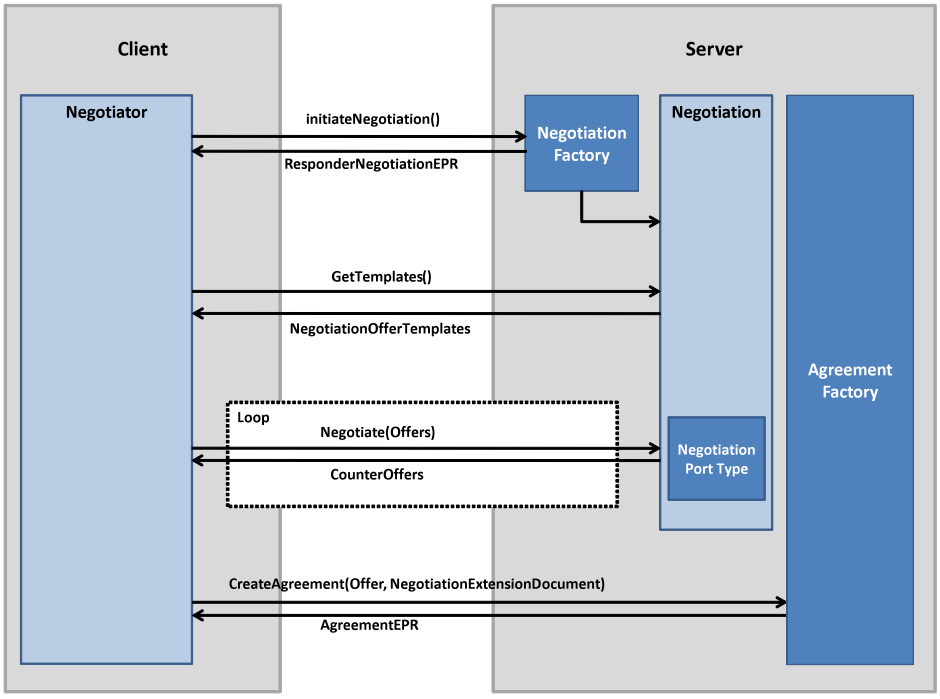

Figure 9: Asymmetric deployment of the WS-Negotiation port types

7.2 Bilateral Negotiation with Asymmetric Agreement Layer

In a bilateral negotiation both parties actively participate in the negotiation process. For that reason both parties implement the WS-Agreement *NegotiationFactory* and *Negotiation port types*. A bilateral negotiation process is initiated as follows. The negotiation initiator creates a new negotiation instance. This instance is a web service resource that implements the WS-Agreement Negotiation port type. The negotiation initiator then invokes the *initiateNegotiation* method of the negotiation responder. The *initiateNegotiation* request includes an endpoint reference to the negotiation instance created beforehand. Moreover, it contains the negotiation context that defines the roles of each party participating in the negotiation process. The negotiation context defines for example which party acts as agreement initiator and which party acts as agreement responder. Once the negotiation instance is created, the negotiation context is fixed and the roles and responsibilities of the negotiation participants do not change anymore.

The negotiation scenario depicted in Figure 10 shows an example of a bi-lateral negotiation. In this scenario the negotiation initiator is also the agreement initiator. The negotiation initiator starts the negotiation by initiating a new negotiation process with the responder. Next the initiator queries the negotiable templates from the negotiation responder and creates an initial negotiation offer based on the template it wants to create a SLA for. The initiator then notifies the responder about the initial negotiation offer. This is done by sending the offer to the responder by invoking its *Advertise* method. The negotiation responder now takes an active role in the negotiation process. It creates counter offers for the received negotiation offer and sends them to the initiator by invoking its negotiate method. After several rounds of negotiation the agreement initiator decides to create an agreement based on one of the negotiated offers. It therefore calls the *createAgreement* method of the responder, passing the negotiated agreement offer along with a NegotiationExtensionDocument. The NegotiationExtensionDocument is passed as a critical extension. It refers to the negotiation instance that was used to negotiate the agreement offer and contains a reference to the originating negotiation offer.

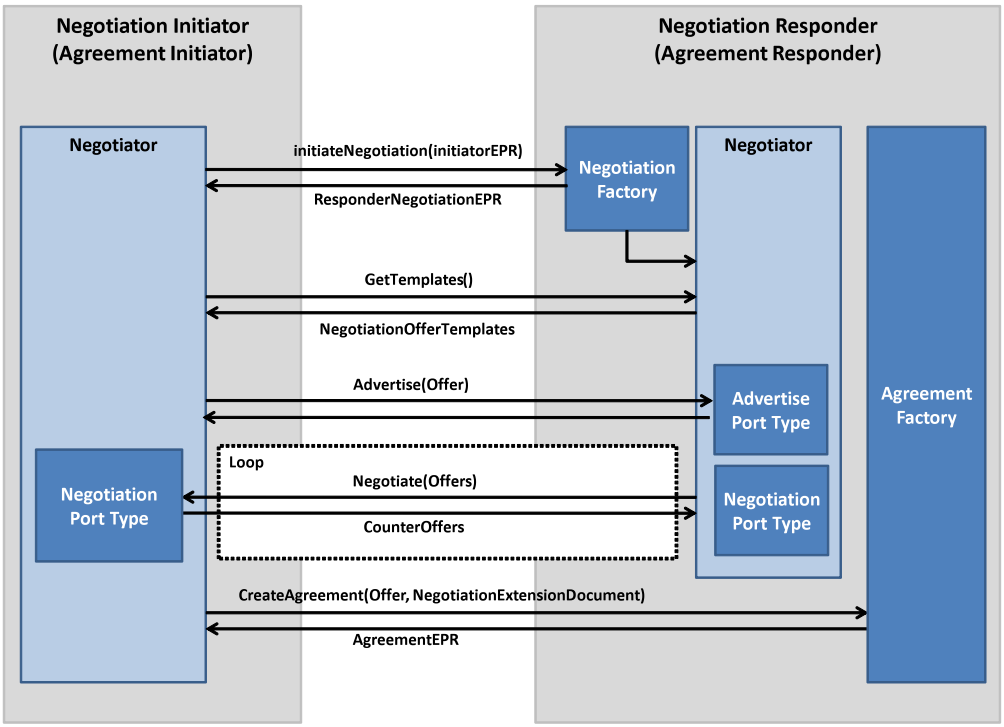

Figure 10: Symmetric deployment of WS-Agreement Negotiation, where the Negotiation Initiator is also the Agreement Initiator and the Negotiation Responder is the Agreement Responder. Both parties have an active role in the negotiation process.

7.3 Re-Negotiation of Existing Agreements

Renegotiation of existing agreements applies the same signaling pattern as negotiation of agreements. If the original agreement initiator matches the initiator of the renegotiated agreement, the roles and obligations of the original agreement also match the roles and obligations of the renegotiated agreement. If the agreement initiator and responder roles are changed, the roles and obligations in the renegotiated agreement must be adopted accordingly. As mentioned before, the roles and the responsibilities of the negotiating parties are specified in the negotiation context as soon a new negotiation is initiated. In a renegotiation process, the negotiation context must also refer to the agreement to renegotiate. It MUST therefore contain an endpoint reference to the original responder agreement instance. In a symmetric deployment of the agreement port type, the negotiation context SHOULD also include a reference to the original initiator agreement. After the initialization of the renegotiation process, both parties negotiate an acceptable agreement offer. In case they succeed negotiating such an offer, the party defined as agreement initiator invokes the *createAgreement* (*create-PendingAgreement*) method of the responder. When the renegotiated agreement is created successfully, the original agreements MUST change their states to *Completed*.

The layout of the agreement layer may either be symmetric or asymmetric. A detailed description of symmetric deployments of the agreement port type is given in the section *Port Types and Operations* of the WS-Agreement specification [GDF107]. Figure 11 shows a symmetric deployment of the negotiation and agreement port types. In this scenario, the initiator of the original agreement becomes the agreement responder for the renegotiated agreement. The roles of the agreement initiator and responder therefore change in the renegotiated agreement and must be adopted accordingly.

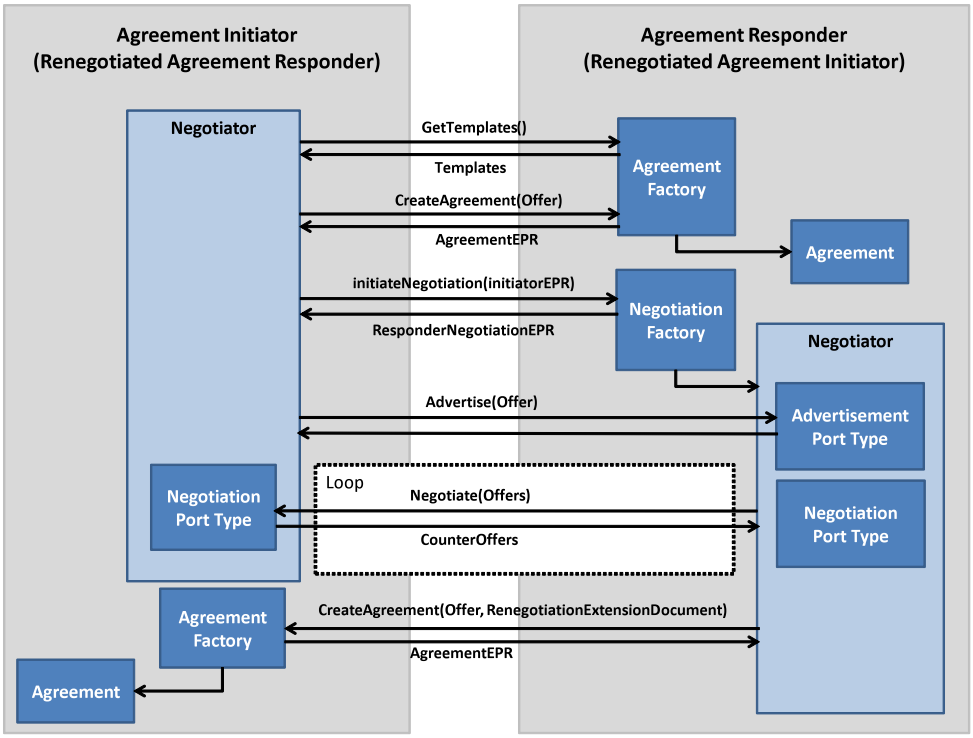

Figure 11: Symmetric signaling on the Negotiation and Agreement Layer. Both parties implement the WS-Agreement Negotiation and WS-Agreement port types. Here, the roles of agreement initiator and responder change for the renegotiated agreement. The responder of the original agreement triggers the creation of the renegotiated agreement instance through the original agreement initiator's agreement factory.

7.4 Negotiation Factory Port Type

7.4.1 Operation wsag-neg:InitiateNegotiation

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

7.4.1.1 Input

```
<wsag-neg:InitiateNegotiationInput>

       <wsag-neg:NegotiationContext>

       …

       </wsag-neg:NegotiationContext>


      <wsag-neg:InitiatorNegotiationEPR>

              <wsa:EndpointReference>

                     wsa:EndpointReferenceType

              </wsa:EndpointReference>

       </wsag-neg:InitiatorNegotiationEPR> ?

       <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:NegotiationContext
This REQUIRED element defines the context of the negotiation that is initiated. The negotiation context applies to the whole lifetime of the negotiation process.

/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 is used in symmetric deployment scenarios of the Negotiation port type in order to initiate a bilateral negotiation.

/wsag-neg:InitiateNegotiationInput/wsag-neg:NoncriticalExtensions
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 were 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.

7.4.1.2 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/{any}
The response MAY carry additional domain specific elements that are associated with the corresponding extensions of the input message.

7.4.1.3 Faults

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

7.5 Negotiation Port Type

7.5.1 Operation wsag-neg:Negotiate

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

7.5.1.1 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. A negotiation offer must reference one template 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.

7.5.1.2 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 SHOULD 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.

7.5.1.3 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.

7.5.2 Operation wsag-neg:Terminate
This operation terminates a negotiation process, if permissible. All offers negotiated in the context of this negotiation process are invalidated.

7.5.2.1 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.

7.5.2.2 Result
```
<wsag-neg:TerminateOutput>

</wsag-neg:TerminateOutput>
```

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

7.5.2.3 Faults
This operation does not throw any faults.

7.5.3 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 4.1.

7.5.4 Resource Property wsag-neg:NegotiationOfferTemplate
The *wsag-neg:NegotiationTemplate* property is of the type wsag:AgreementTemplateType. The cardinality of this resource property is 0 to n. It represents a set of agreement templates that can be used to create negotiation offers within this particular negotiation instance.

7.5.5 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.

7.6 Offer Advertisement Port Type

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

7.6.1 Operation wsag-neg:Advertise

The wsag-neg:Advertise operation is used to notify a negotiation participant 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.

7.6.1.1 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.

/wsag-neg:AdvertiseInput/{any}
The Advertise 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.

7.6.1.2 Result

```
<wsag-neg:AdvertiseOutput>

</wsag-neg:AdvertiseOutput>
```

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

7.6.1.3 Faults

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

8 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

9 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.

10 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.

11 Full Copyright Notice

12 References

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

[GFD107] Andrieux, A. and Czajkowski, K. and Dan, A. and Keahey, K. and Ludwig, H. and Nakata, T. and Pruyne, J. and Rofrano, J. and Tuecke, S. and Xu, M. WS-Agreement specification, GFD.107. May 25, 2007

13 Appendix 1: XML Schema and WSDL

13.1 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="NegotiatiableTemplate"

         type="wsag:AgreementTemplateType" />

    <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" />
        <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-
neg:NegotiationConstraintSectionType" />
                </xsd:sequence>
                <xsd:attribute name="OfferId" type="xsd:string" />
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:complexType name="NegotiationConstraintSectionType">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0"
name="Item"
                type="wsag-neg:NegotiationOfferItemType" />
            <xsd:element maxOccurs="unbounded" minOccurs="0"
                ref="wsag:Constraint" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="NegotiationOfferItemType">
        <xsd:complexContent>
            <xsd:extension base="wsag:OfferItemType">
                <xsd:attribute name="Type"
                               type="wsag-
neg:NegotiationConstraintType"
                               use="required" />
                <xsd:attribute name="Importance" type="xsd:integer"
                               default="0" use="optional"/>
```

```
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>

    <xsd:simpleType name="NegotiationConstraintType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Required" />
            <xsd:enumeration value="Optional" />
        </xsd:restriction>
    </xsd:simpleType>

    <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="Acceptable"
             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:element name="NegotiationOfferContext"
               type="wsag-neg:NegotiationOfferContextType"
minOccurs="1" />
          <xsd:any namespace="##other" minOccurs="0"
               processContents="lax"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="RenegotiationExtensionType">
```

```
      <xsd:sequence>

          <xsd:element name="ResponderNegotiationEPR"

               type="wsa:EndpointReferenceType" minOccurs="1" />

          <xsd:element name="InitiatorNegotiationEPR"

               type="wsa:EndpointReferenceType" minOccurs="0" />

          <xsd:element name="ResponderAgreementEPR"

               type="wsa:EndpointReferenceType" minOccurs="1" />

          <xsd:element name="NegotiationOfferContext"

               type="wsag-neg:NegotiationOfferContextType"
minOccurs="1" />

          <xsd:any namespace="##other" processContents="lax"

               minOccurs="0" />

      </xsd:sequence>

    </xsd:complexType>


</xsd:schema>
```

13.2 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>
```

13.3 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:NegotiatiableTemplate"
/>
                    <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-
neg: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>
```

13.4 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>
```