Web Services Agreement Specification (WS-Agreement)

Version 1.0

2/21/2004

Authors (alphabetically):

Alain Andrieux, (Globus Alliance / USC/ISI)
Karl Czajkowski, (Globus Alliance / USC/ISI)
Asit Dan (IBM)
Kate Keahey, (Globus Alliance / ANL)
Heiko Ludwig (IBM)
Jim Pruyne (HP)
John Rofrano (IBM)
Steve Tuecke (Globus Alliance / ANL)
Ming Xu (Platform Computing)

Abstract

This document describes Web Services Agreement Specification (WS-Agreement), an XML language for specifying an agreement between a resource/service provider and a consumer, and a protocol for creation of agreement through negotiation using an agreement template.

Status

This document is a draft of the WS-Agreement Specification from the Global Grid Forum (GGF). This is a public document being developed by the participants of the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the GGF.

Full Copyright Notice

Intellectual Property Statement

The GGF 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 GGF Secretariat.

The GGF 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 GGF Executive Director (see contact information at GGF website).

Table of Contents

1 Introduction

In a distributed service-oriented computing environment, service consumers like to obtain guarantees related to services they use, often related to quality of a service. Whether service providers can offer – and meet – guarantees usually depends on their resource situation at the requested time of service. Hence, quality of service and other guarantees that depend on actual resource usage cannot be advertised as an invariant property of a service using, for example, WS-Policy, and then bound to by a service consumer. Resource state-dependent guarantees must be negotiated between a service consumer and provider resulting in an agreement on the service and the associated guarantees. Additionally, the guarantees on service quality must be monitored and failure to meet these guarantees must to be notified to consumers. The objective of the WS-Agreement specification is to define a language for agreements and offers, a mechanism for negotiating agreements, and the ability to monitor agreement compliance at runtime.

An agreement between a service requester and a service provider specifies one or more service level objectives both as expressions of requirements of the service consumer and assurances by the provider on the availability of resources and/or on service qualities. For example, an agreement may provide assurances on the bounds on service response time and service availability. Alternatively, it may provide assurances on the availability of minimum resources such as memory, CPU MIPS, storage, etc.

To obtain this assurance on service quality, the service consumer or an entity acting on its behalf must establish a service agreement with the service provider, or another entity acting on behalf of the service provider. Because the service objectives relate to the definition of the service, the service definition must be part of the terms of the agreement or be established prior to agreement creation. This specification provides a schema for defining overall structure for an agreement document. An agreement includes information on the agreement parties and references to prior agreements, referred to as agreement context, one or more discipline specific service definition terms, and one or more guarantee terms specifying service level objectives and business values associated with these objectives.

The agreement creation process typically starts with a pre-defined agreement template specifying customizable aspects of the documents, and rules that must be followed in creating an agreement. These rules are defined by negotiability constraints. This specification defines a schema for an agreement template.

The creation of an agreement can be initiated by the consumer side or by the provider side. While simple scenarios for agreement creation may involve little or no negotiation, creation of an agreement through negotiation can involve numerous scenarios depending on the consumer or provider side acting as initiator, maintainer of negotiation state and finally maintainer of agreement state. This specification defines a core set of messages and resources modeling these states for supporting many usage scenarios.

We use a coherent example of a hypothetical job submission to illustrate various aspects of the WS-Agreement specification, particularly relationship of service level

objectives with service description, an agreement template specifying alternative service description terms and use of WS-Policy compositor, and negotiability constraints in negotiating service level objectives. Details of the example scenario are described in section 2.

Sections 3, 4, 5 specify the overall agreement structure, service description as agreement terms and guarantee terms, respectively. Section 6 specifies schema for the agreement template and negotiability constraints. Section 7 describes the layered service model and introduces the port types and operations in the specification. Section 8 specifies various phases of the agreement creation process, namely, simple flow for agreement creation, negotiation in initial agreement creation and renegotiation of agreements, respectively.

1.1 Goals and Requirements

The goals of WS-Agreement are to standardize the terminology, concepts, overall agreement structure with types of agreement terms, agreement template with negotiability constraints and protocols for creation, negotiation and renegotiation of agreements, including WSDL needed to express the message exchanges and resources needed to express the state.

1.1.1 Requirements

In meeting these goals, the specification must address the following specific requirements:

- **Must allow use of any service description term:** It must be possible to create agreements for services defined by any domain specific service description terms, such as job specification, data service specification, network topology specification and web service description language (WSDL). Service objective description will reference the elements defined in service description.
- **Must allow creation of agreements for existing and new services:** It must be possible to create agreements for predefined services and resources modeling service state. Additionally, service description can be passed as agreement terms for coordinated creation of agreements and new service specific resources.
- **Must allow use of any condition specification language:** It must be possible to use any domain specific or other standard condition expression language in defining service level objectives and negotiability constraints.
- **WS-Agreement creation must be independent of specific negotiation model:** A large number of negotiation scenarios are possible depending on whether a provider or consumer initiates agreement creation, and also where the agreement state is maintained. The basic messages defined in this document can be applied for modeling various usage specific scenarios.
- **Relationship to other WS-* specifications:** WS-Agreement must be composable with other Web services specifications, in particular WS-Security, WS-Policy, WS-Federation, WS-Addressing, WS-Coordination, WS-ResourceProperties, WS-ResourceLifetime, Web Services for Remote Portals, and WS-ReliableMessaging and the WS-Resource framework [WS-Resource].

1.1.2 Non-Goals

The following topics are outside the scope of this specification:

- Defining domain-specific expressions for service descriptions.
- Defining specific condition expression language for use in specifying guarantee terms and certain negotiability constraints. We assume standards will emerge elsewhere for a common expression definition language. Alternatively, different expression language may be used in different usage domain.
- Defining specific service level objective terms for a specific usage domain such as network, server, applications, etc.
- Defining specification of metrics associated with agreement parameters, i.e., how and where these are measured.
- Protocol and conventions for claiming services according to agreements is considered domain-specific. For example, agreement identification in SOAP headers might suit a Web service, another mechanism is required for networking services, etc.

1.2 Notational Conventions

The keywords “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [RFC 2119].

When describing abstract data models, this specification uses the notational convention used by the [XML Infoset]. Specifically, abstract property names always appear in square brackets (e.g., [some property]). When describing concrete XML schemas, this specification uses the notational convention of [WS-Security]. Specifically, each member of an element’s [children] or [attributes] property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (<xsd:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xsd:anyAttribute/>).

1.3 Namespace

This is an XML or other code example:

```
http://www.ggf.org/namespaces/ws-agreement (Code)
```

The following namespaces are used in this document:

Prefix	Namespace
wsag	http://www.ggf.org/namespaces/ws-agreement (*temporary*)
wsa	http://schemas.xmlsoap.org/ws/2003/03/addressing
wsbf	http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults
wssg	http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup
wsrp	http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties

xs/xsd	http://www.w3.org/2001/XMLSchema
xsi	http://www.w3.org/2001/XMLSchema-instance
wsdl	http://schemas.xmlsoap.org/wsdl/

2 Example Scenarios

WS-Agreement covers a wide scope of application scenarios relating to the establishment of an agreement between a service provider and a service consumer. This is achieved by using a single format of document and a protocol comprising few states. Two examples are chosen here to illustrate the range of applications that this specification covers. These examples are referred to throughout the specification.

2.1 Job submission

A typical application scenario is the request for executing a computing job. A service provider may post an agreement template available to interested requesters. In this scenario, the agreement template defines the list applications to be executed, and the software execution environment typically specified in a job submission. Service consumers are given a quality of service guarantee in terms of number of nodes and/or per node memory and storage for a specific time period. Alternatively, the guarantees can be on the completion time. A service consumer requesting a submitted job must fill in the name of the application to be executed, input and output files. In addition, a service consumer chooses the number of nodes (or any other resource requirements) that are guaranteed for the application to be executed on.

To submit a job, a service consumer retrieves the template from the provider, selects the application name, and provides URL of the input and output files as well as the details of resource guarantees. The filled template is sent as an offer to the provider. The provider decides whether to accept or reject the requested job. This may depend on the queue of jobs waiting to be processed and the current allocation of resources. The service provider answers the offer with a confirmation or a fault. In due time, the service provider processes the job and writes the output file to the URL defined in the agreement.

2.2 Service Parameterization

In the second scenario, the service contracted is an authentication and access control service. The service exposes an interface to register a new user, set an access control policy, manage a user's passwords, authenticate a user and check a requested user action against the corresponding access control policy. In an access control environment, quality of service aspects such as response to for access verification and service availability is critical. Depending on particular needs, service consumers require different service quality levels and are prepared to pay differently for their quality of service requirements.

The service is very convenient for event organizers or other temporary projects. For example, sports events such as

an athletics meeting or a soccer tournament require access control services for a limited amount of time to a large and diverse group of constituents such as athletes, journalists, jurors, and spectators who access the event's Web site or applications.

A service provider offers an agreement template describing the service and its guarantees, including the options available to the customer. The service description includes the WSDL of the service interface. Customer can choose among a service using Kerberos-based authentication or a proprietary authentication system. Furthermore, customers can choose how many users ID should be managed. Customers can add availability and response time guarantees to individual operations of the interface, e.g., to distinguish quality requirements for management and access control operations. For operation availability, customers choose between 95%, 98%, 99%, and 99.9%, defined as receiving an reply in 15 seconds. For average response time guarantees, customers choose between 0.5, 1 or 2 seconds, and set the number of operations per minute for which the response time goal must hold. Also, customers can set the time when the service will be available.

This template offers many options to service consumers. Service consumers send a completed offer to the service provider. Based on capacity limitations, the provider may accept the offer or counter-proposes. For example, if a service consumer asks for 1 sec response time for up to 1000 requests per minute, the provider might only have capacity for up to 500 requests and counter-proposes an agreement for 500 requests, maybe for a lower price, suggesting that the service consumer can buy the rest of the capacity from a different provider.

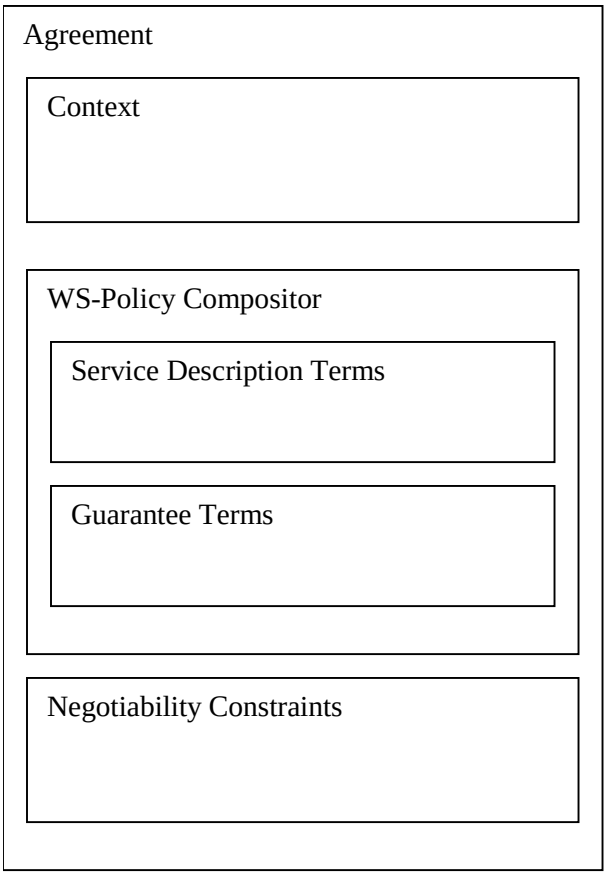
Once the agreement is “signed”, the provider provisions the service and exposes status information on guarantee compliance to the user. The service consumer may shop for the remaining capacity needs at different providers.

In the course of the event, it may turn out that more or less capacity is needed. Hence, the service consumer want to be able to renegotiate the agreement.

3 Agreement Structure

An agreement document is composed of three distinct parts. We summarize the structure in the following diagram:

Figure 1: Agreement structure.



Structure of an agreement template document

The first section is the context, which contains the meta-data describing the agreement as a whole. It names the participants in the agreement, the agreement's lifetime and links to other agreements related to this agreement. The next section contains the terms that describe the agreement itself. The terms are contained within a WS-Policy compositor allowing the terms to be combined in logical groups providing the possibility of creating agreements with alternative. We define two types of terms: Service Description Terms and Guarantee Terms. The Service Description Terms provide information needed to instantiate or otherwise identify a service to which this agreement pertains. The guarantee terms specify the service levels that the parties are agreeing to. Management systems may use the guarantee terms to monitor the service and enforce the agreement. Lastly, the constraints provide guidelines as to how the values of the terms may be changed during the lifetime of the agreement by specifying the valid ranges or distinct values that the terms may take. The constraints refer back to individual terms they apply to using XPATH.

Ultimately, a WS-Agreement document has the following structure:

```
<wsag:Agreement Name="xs:NCName"?>
   <wsag:AgreementContext>
      wsag:AgreementContextType
   </wsag:AgreementContext>
   <wsp:all>
         <wsag:ServiceDefinitionTerm> ...
```

```
        </wsag:ServiceDefinitionTerm> ?
        <wsag:GuaranteeTerm> ... </wsag:GuaranteeTerm> ?
   </wsp:all>
   <wsag:NegotiabilityDescription>…</wsag:NegotiabilityDescription> ?
</wsag:Agreement>
```

The following describes the attributes and tags listed in the schema outlined above:

/wsag:Agreement

This is the outermost document tag which encapsulates the entire agreement. An agreement contains and agreement context and a collection of agreement terms.

/wsag:Agreement/@Name

This is an OPTIONAL name that can be given to an agreement

/wsag:Agreement/AgreementContext

This is a REQUIRED element in the agreement and provides information about the agreement that is not specified in the terms such as who the involved parties are, what the services is that is being agree to, the length of the agreement, and references to any related agreements.

/wsp:all

The terms of an agreement comprises one or more service definition terms, and zero or more guarantee terms grouped using the WS-Policy compositor.

/wsag:Agreement/ServiceDefinitionTerms

These terms are OPTIONAL and MAY specify the parameters used to instantiate a service which will fulfill this agreement or to describe a service to be used by the agreement.

/wsag:Agreement/GuaranteeTerms

These terms are OPTIONAL and MAY specify the guarantees (both promises and penalties) that are associated with the other terms in the agreement.

/wsag:Agreement/NegotiabilityDescription

These are OPTIONAL elements that MAY provide constraints on the values that the various terms may take.

4 Agreement Context

An agreement is scoped by its associated context that SHOULD include parties to an agreement, and additionally, SHOULD include reference to the service(s) provided in support of the agreement. The context MAY also include other prior and/or related agreements. The new agreement thus augments prior related agreements, between the service consumer and the service provider.

The <wsag:AgreementContext> element is used to describe the involved parties and the identify the service that the agreement is about. It can also optionally contain references to other related agreements.

```
<wsag:AgreementContext>
  <wsag:AgreementInitiator>xs:AnyURI</wsag:AgreementInitiator>
  <wsag:AgreementProvider>xs:anyURI</wsag:AgreementProvider>
```

```
  <wsag:TerminationTime>xs:DateTime</wsag:TerminationTime>
  <wsag:ServiceReference>
     <wsa:EndpointReference xmlns:wsa="..." xmlns:fabrikam="...">
        <wsa:Address>http://www.fabrikam123.com/acct</wsa:Address>
        <wsa:PortType>fabrikam:JobSubmissionPortType</wsa:PortType>
     </wsa:EndpointReference>
  </wsag:ServiceReference>
  <wsag:RelatedAgreements>...</wsag:RelatedAgreements>
</wsag:AgreementContext>
```

The following describes the attributes and tags listed in the schema outlined above:

/wsag:AgreementContext

This is the outermost tag which encapsulates the entire agreement context

/wsag:AgreementContext/AgreementInitiator

This is element identifies of the initiator of the agreement. The URI for an agreement provider MAY be an wsa:EndpointReference from WS-Addressing or MAY identify the initiator by more abstract naming, e.g. by security identity of the owner or operator.

/wsag:AgreementContext/AgreementProvider

This is element identifies the provider of the agreement. The URI for an agreement provider MAY be an wsa:EndpointReference from WS-Addressing or a Grid Service Handle (GSH) [OGSI] to an existing service or MAY instead identify the provider by more abstract naming, e.g. by security identity of the owner or operator.

/wsag:AgreementContext/TerminationTime

This element specifies the time at which this agreement is no longer valid. Agreement initiators MAY use this mechanism to negotiate Agreement service lifetime. Extended negotiation languages MAY define other mechanisms to negotiate lifetime integrated with other negotiation terms. The resulting negotiated lifetime MUST be exposed as wsag:TerminationTime and further negotiation MUST be possible through the basic OGSI mechanisms.

/wsag:AgreementContext/ServiceReference

This element is OPTIONAL and defines references to the provided service(s) for which the agreement terms are defined.

/wsag:AgreementContext/RelatedAgreements

This element defines references to any number of related agreements that define existing agreement terms which are being augmented via this agreement. The related agreements are represented in the agreement service as related agreement services (see Section 4.6).

/wsp:AgreementContext/{any}

Additional child elements MAY be specified to make additional agreement contexts but MUST NOT contradict the semantics of the parent element; if an element is not recognized, it SHOULD be ignored.

/wsp:AgreementContext/@{any}

Additional attributes MAY be specified but MUST NOT contradict the semantics of the owner element; if an attribute is not recognized, it SHOULD be ignored.

A wsag:AgreementContext element of type wsag:AgreementContextType MAY be used in an agreement to define an agreement context. Alternatively, the agreement context MAY be extended, through XSD extension of wsag:AgreementContextType, to define other attributes of the parties or services to an agreement.

5 Agreement Terms

The terms of an agreement comprises one or more service definition terms, and zero or more guarantee terms grouped using the WS-Policy compositor. The specification defines schema for service description and agreement terms as abstract types, that must be extended for specific usage domain.

5.1 Service Description Terms

Service description terms are a fundamental component of an agreement: at the very least the service provider agrees to provide a service described by service description terms. Providing this service may be qualified, and additional service level objectives on how the service is performed may be imposed by the service guarantee; service terms define the functionality that will be delivered under an agreement. The service description content itself is dependent on the particular domain. A ServiceDescriptionTerm consists of two parts,

Service EPR: Any number of EPRs referring to service instances, and

ServiceDescriptions, which describe the service and are typically expressed in a domain-specific language.

This ServiceDescriptionTerm type and element defined here encompass a general top-level attribute and they are expected to be extended to capture a domain-specific form of service specification. An Agreement MAY contain any number of SDTs, as an agreement can refer to multiple different service components.

5.1.1 Service Description Term Definition

The following definition describes the simple generic content of this type:

```
<wsag:ServiceDescriptionTerm name="xs:NCName">
   <wsa:EndpointReference>…</wsa:EndpointReference> *
   <wsag:ServiceDescription>…</wsag:ServiceDescription> *
   <wsag:Variables>…</wsag:Variables>
</wsag:ServiceDescriptionTerm>
```

The following describes the elements of the schema above:

/wsag:ServiceDescriptionTerm

ServiceDescriptionTerm encloses a description of a service.

/wsag:ServiceDescriptionTerm/@name

The name attribute (of type xs:NCName) represents the name given to a term. Since an Agreement MAY encompass multiple ServiceDescriptionTerms each term MUST be given a unique name for convenient referencing (see guarantee term section).

/wsag:ServiceDescriptionTerm/wsa:EndpointReference

The element is taken from the WS-Addressing specification. An Endpoint Reference points to a particular instance of a resource. An EPR can be provided if the service has been instantiated before the agreement is concluded. This applies either to known services that are made available by a provider to different service consumers or the service provider creates the service instance in the course of the negotiation prior to accepting an offer.

/wsag:ServiceDescriptionTerm/wsag:ServiceDescription

ServiceDescriptions contain information to either explain the semantics of the ERP, e.g., by means of a WSDL, or contain information that is needed to instantiate the agreed service. The ServiceDescriptionType is expected to be extended by domain-specific types and the ServiceDescriptionTerm element MAY be substituted by element of a subtype of ServiceDescriptionTermType.

/wsag:ServiceDescriptionTerm/wsag:Variables

This element, of type VariableSetType, defines variables that can be referred to in guarantee expressions.

The EPR(s) of the service instance may be passed by the initiator, when an existing instance is being referred (possibly, based on agreement template). However, when a new service instance is created based on the ServiceDescriptions, the EPR of a newly created service instance is (most likely) being generated by the agreement factory. Note that the initiator can always pass a reference to be assigned to the newly created service instance. For a more complex service, one or more EPRs may be returned.

5.1.2 Service Descriptions

Service descriptionss are intended to describe the service in a domain-specific way, for example by using domain-specific term languages such as JSDL. A job execution service may for example include a job name together with all the argument values for a specific job execution. The service description MAY be associated with a WSDL definition if the service is to be implemented by a Web service.

The definition of the XML Infoset of this supertype is:

```
<wsag:ServiceDescription >
...
</wsag:ServiceDescription>
```

It must be extended to contain domain-specific descriptions.

To illustrate one way of describing the service, we provide a Service descriptions extension containing a reference to a WSDL file:

```
<wsag:WSDLFileReference >
   <wsag:URL> … </wsag:URL>
</wsag:WSDLFileReference >
```

The following describes the elements above:

/wsag WSDLFileReference

WSDLFileReference, of WSDLFileReferenceType, contains a pointer to a WSDL file.

/wsag:Filereference/wsag:URL

This element (of type xs:anyURI) is optional and contains a reference in the form of a URL to a WSDL file that describes the service to which the Endpoint Reference points.

In a job submission context, the service descriptions MAY contain a job description, as illustrated below corresponding to the first example scenario:

```
<job:JobDescription>
   <job:InputFileURL> … </job:InputFileURL>
   <job:Diskspace> … </job:Diskspace>
   <job:Memory> … </job:Memory>
      …
</job:JobDescription>
```

In this example, the job parameters are described in a job description language, which is not part of WS-Agreement but contained in a ServiceDescription.

5.1.3 Variables

Guarantees contain conditions, which are logic expressions that refer to attributes of a service such as metrics for availability and response time that are subject to the guarantee. The semantics of those variables must be defined to interpret the condition expression. Individual variables are defined as defined below:

```
<wsag:Variable name=″xsd:NCName″ metric=″xsd:QName″>…</wsagVariable>
```

/wsag:Variable

This element, of type xsd:string, is an XPATH to a service definition term or any point within a service definition term where the semantics of this variable is defined.

/wsag:Variable/@name

This element, of type xsd:NCName, is the unique identifier of the variable in the scope of this WS-Agreement.

/wsag:Variable/@metric

This element, of type xsd:QName, is an identification of a metric, e.g., availability, which is domain-specific. This element is optional and intended for cases where the XPATH does not sufficiently explain the semantics of a variable.

Example:

```
<wsag:Variable name=″numberOfNodes″
>/wsag:Agreement/job:JobDescription</wsag:Variable>
```

In this example, a variable numberOfNodes, whose interpretation is assumed to be understood in the domain, refers to the service term JobDescription.

Variables are grouped into a set:

```
<wsag:VariableSet>
   <wsagVariable> … </wsag:Variable> *
</wsag:VariableSet>
```

/wsag:VariableSet

This element, of type VariableSetType, contains one or more variables.

/wsag:VariableSet/wsag:Variable

Variables are specified above.

5.2 Guarantee Terms

The primary motivation for creating a service agreement between a provider and a service consumer is to provide assurance to a service consumer on the service quality and/or resource availability by the provider. Guarantee terms define this assurance on service quality, associated with the service defined by the service definition terms. For the job submission example, an agreement may provide assurance on the bounds (e.g., minimum) on the availability of resources such as memory, CPU MIPS, storage and/or job execution start or completion time. These bounds are referred to as the service level objectives (SLO).

An expression of assurance also includes qualifying conditions on external factors such as time of the day as well as the conditions that a service consumer must meet. For example, a bound on the average response time of the authorization service (as per the second example) is assured only if the request rate is below a specified threshold during weekdays.

An assurance also includes specification of one more forms of business values associated with an SLO. For example, a business value may represent the strength of this commitment by the provider. Another example of business value is the importance of this assurance to the consumer and/or to the provider.

An agreement MAY contain zero or more GuaranteeTerm, where each GuaranteetTerm consists of three parts,

QualifyingCondition: an optional condition that must be met (when specified) for a guarantee to be enforced,

ServiceLevelObjective: a condition expressed over service descriptionss, and

BusinessValueList: one or more business values associated with this objective.

Note that a single ServiceLevelObjective can be a complex of objectives expressed as a complex condition expressing bounds over many service attributes. Meeting the overall objective may imply meeting all the individual objectives. However, if the business values associated with individual objectives are different, (for example, if not all objectives are equally important), then each objective should be expressed as a separate GuaranteeTerm. Similarly, a QualifyingCondition can be a complex condition if multiple qualifying conditions need to be met for a guarantee to be honored.

5.2.1 Guarantee Term Definition

The *GuaranteeTerm* comprises three elements as defined below:

```
<wsag:GuaranteeTerm>
   <wsag:QualityingCondition>…</wsag:QualifyingCondition>?
   <wsag:ServiceLevelObjective>…</wsag:ServiceLevelObjective>
   <wsag:BusinessValueList>…</wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

/wsag:GuaranteeTerm

This element, of type GuaranteeTermType, represents an individual guarantee related to the service described in service description terms.

/wsag:GuaranteeTerm/wsag:QualifyingCondition

A qualifying condition, of type ConditionType, represents the precondition under which a guarantee holds.

/wsag:GuaranteeTerm/wsag:ServiceLevelObjective

This element, of type ConditionType, expresses the condition that must be met to satisfy the guarantee.

/wsag:GuaranteeTerm/wsag:BusinessValueList

This is the higher level element that contains a list of business value elements associated with a service level objective. Two standard business value types are defined later. Customized business value types can be expressed extending an abstract business value type, defined here.

The detailed description of the types associated with a GuaranteeTerm follows in the subsections.

5.2.2 Qualifying Condition and Service Level Objective

QualifyingCondition and *ServiceLevelObjective* are expressed as a condition over service attributes and/or external factors such as date time. Expression of arithmetic, Boolean and date-time expression is required in many contexts, and not just in agreements. An example of condition expression language can be found in [XQUERYX]. Hence, the *conditionType* is defined as an abstract type that can be extended with specific condition expression language, addressing the requirements of a particular domain.

```
<wsag:Condition> … </wsag:Condition>
```

5.2.3 Business Value

Associated with each ServiceLevelObjective is a BusinessValueList that contains multiple business values, each expressing a different value aspect of the objective. The values may express relative importance of this objective to a consumer or penalty to be assessed upon failure to meet this objective. Other customized domain specific business values can be defined and associated with a service level objective.

Expression of business value in meeting certain assurances and flexible specification of service consumer requirements may free a provider from fixed allocation of resources. A provider can dynamically allocate resources based on actual measured or estimated service consumer requirements, and evaluation of business values. For

example, a new arrival of a high priority job may result in reduction of allocated resources or suspension of an existing low priority job.

```
<wsag:BusinessValueList>
  <wsag:Importance> xsd:integer </wsag:Importance>?
  <wsag:Penalty> </wsag:Penalty>?
  <wsag:Reward> </wsag:Reward>?
  <wsag:BusinessValue> … </wsag:BusinessValue>*
</wsag:BusinessValue>
```

/wsag:BusinessValueList

This element comprises the set of business value expressions.

/wsag:BusinessValueList/wsag:Importance

This element when present expresses relative importance (defined below) of meeting an objective.

/wsag:BusinessValueList/wsag:Penalty

This element (defined below) when present expresses penalty to be assessed for not meeting an objective.

/wsag:BusinessValueList/wsag:Reward

This element (defined below) when present expresses reward to be assessed for meeting an objective.

/wsag:BusinessValueList/wsag:BusinessValue

Zero or more domain specific customized business values can be defined.

5.2.3.1 Importance

In many cases, all service level objectives (SLO) will not carry the same level of importance. It is necessary therefore, to be able to assign a “business value” in terms of relative importance to an objective so that its importance can be understood, and so tradeoffs can be made by the provider amongst various guarantees when sufficient resources are available. Absolute value of a guarantee on the other hand specifies business impact of meeting or violating an individual SLO, expressed via Reward and Penalty. Relative importance can be though of as a measure of importance with a default measurement unit.

Relative terms, such as high, low, medium, etc. can be used to prioritize across many guarantees. However, to provide stronger semantics and easier comparison of this value, this is expressed using an integer.

5.2.3.2 Penalty and Rewards

In business SLAs, this importance is indirectly expressed by specifying the consequences of not meeting this assurance. Here, each violation of a guarantee term during an assessment window will incur a certain penalty. The penalty assessment is measured in a specified unit and defined by a value expression.

```
<wsag:Penalty>
 <wsag:AssesmentInterval>
   <wsag:TimeInterval> xsd:any </wsag:TimeInterval>?
```

```
   <wsag:Count> xsd:integer </wsag:Count>?
  </wsag:AssesmentInterval>
 <wsag:ValueUnit> xsd:string </wsag:ValueUnit>
 <wsag:ValueExpr> xsd:float </wsag:ValueExpr>?
 <wsag:ValueExpr> xsd:any </wsag:ValueExpr>?
</wsag:Penalty>
```

/wsag:Penalty

This element defines a business value expression for not meeting an associated objective.

/wsag:Penalty/wsag:AssesmentInterval

This element defines the interval over which a penalty is assessed.

/wsag:Penalty/wsag:AssesmentInterval/wsag:TimeInterval

This element when present defines the assessment interval as a time duration.

/wsag:Penalty/wsag:AssesmentInterval/wsag:TimeInterval

This element when present defines the assessment interval as a service specific count, such as number of invocation.

/wsag:Penalty/wsag:ValueUnit

This element defines the unit for assessing penalty, such as USD.

/wsag:Penalty/wsag:ValueExpr

This element defines the assessment amount, which can be an integer, float or an arbitrary domain specific expression.

Alternatively, meeting each objective generates a reward for a provider. The value expression for reward is similar to that of penalty.

6 Agreement Template and Negotiability Constraints

An Agreement MAY contain a Negotiability Description. The Negotiability Description specifies constraints that one party of a negotiation includes in an offer. The specification of a Negotiation Description in an offer does not state a promise that a replying offer fulfilling the constraints will be accepted. It is a voluntary disclosure of a preference to reduce the number of offers to be exchanged to agree or terminate a negotiation. Typically, a provider, e.g., MAY publish an agreement template in the form of an offer containing a Negotiability Description, outlining agreements it is generally willing to accept. Whether the provider accepts a given offer might depend on its current resource situation.

6.1 Negotiability Description Type

The element NegotiabilityDescription is of NegDescriptionType.

```
<wsag:Agreement>
      …
   <wsag:NegotiabilityDescription> ?
      <wsag:Item>…<wsag:Item> *
      <wsag:Constraint>…<wsag:Item> *
   </wsag:NegotiabilityDescription>
```

```
</wsag:Agreement>
```

/wsag:Agreement/wsag:NegotiabilityDescription

This optional element of an Agreement, of type NegDescriptionType, represents to which extent an offer is negotiable. It contains any number of Items and Constraints in any order.

/wsag:Agreement/wsag:NegotiabilityDescription/wsag:Item

Items, of type ItemType are field that are to be filled out in the course of the negotiation.

/wsag:Agreement/wsag:NegotiabilityDescription/wsag:Constraint

A Constraint , of type ConstraintType, defines any restriction that the sender of an offer request relating to values of one or more Items.

6.2 Negotiation Item

A Negotiation Item is a description field of an offer that is expected be filled in the course of the negotiation. It contains a label, a pointer to the position of the field in the terms of the offer and a definition of its acceptable values.

```
<wsag:Item name=″xs:NCName″ location=″xs:string″>
   <xs:restriction>…<xs:restriction> ?
</wsag:Item>
```

/wsag:Item

An Item represents a negotiable field of an offer.

/wsag:Item/@name

The name is a label of the field that uniquely identifies the field in the offer and can be used to refer to item in a convenient way.

/wsag:Item/@location

The location is an XPATH expression that points to the location in the terms of the Agreement that can be changed and filled in. The value currently set at the location to which the XPATH expression points to is the default value of the item.

/wsag:Item/xs:restriction

The restriction, of the group xs:simpleRestrictionModel, is a constraint that restricts the domain beyond the type definition of the particular term syntax of the item, which can be domain-specific. The restriction syntax is taken from the corresponding XML schema schema. It is the responsibility of the sender of the offer to make sure that the restriction defined in the Item is a valid restriction of the type to which the item location points to.

6.3 Constraints

Constraints restrict the possible values of the item set of an offer beyond restrictions of individual items. For example, an offered response time may only be valid for a given range of throughput values of a service. This specification does not define a constraint language but proposes to choose a suitable existing one. Hence, the

Constraint is an empty top-level element that must be extended by a specific, suitable constraint language:

```
   <wsag:Constraint />
```

A general purpose constraint language has been proposed as part of the XQuery and XPATH language. The XML rendering of this expression language, XQueryX, contains a suitable constraint language that can be used to phrase contrains referring to multiple items.

```
   <wsag:XQueryXConstraint>
      <wsag:Expression> … </wsag:Expression>
   </wsag:XQueryXConstraint>
```

/wsag:XQueryXContraint

This element, of type XQueryXConstraintType, substitutes the Constraint element to contain XQueryX expressions.

/wsag:XQueryXContraint/wsag:Expression

This element, of type operatorExpr, taken from the XQueryX schema, contains an operator expression according to this syntax. However, the syntax design of XQueryX is very liberal and, hence, expressions can be phrased that are not semantically valid.
In XQueryX expressions, Item names are mapped to variable names.

Any other constraint language MAY be equally or better suited for particular purposes.

6.4 Example

The following example of a modified JSDL term statement illustrates the use of the NegotiabilityDescription:

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

<wsag:Agreement>
      <job:JobDescription wsp:Usage="wsp:required"
                          wsag:Negotiability=”wsag:Negotiable”>
            <job:InputFile></job:InputFile>
            <job:DiskSize>1000</job:DiskSize>
            …
      </job:CPUUtilization>

      <wsag:GuaranteeTerm name=”NodeGuarantee”
            <wsag:Precondition> … </wsag:Precondition>
            <wsag:ServiceLevelObjective>
```

```
                  <cl:Less>
                        <cl:Variable>nodeNumber</cl:Variable>
                        <cl:Constant><cl:Constant>
                  <cl:Less>
            </wsag:ServiceLevelObjective>
            <wsag:BusinesValue> … </wsag:BusinesValue>
      <wsag>GuaranteeTerm>

      <wsag:NegotiabilityDescription>

         <wsag:Item name="nodeNumber"

location="/wsag:Agreement/wsag:GuaranteeTerm/wsagServiceLevelObjctive/c
l:less/cl/Constant">
            <xsd:restriction>
                  <xsd:minInclusive value="10">
                  <xsd:maxExclusive value=”100">
            </xsd:restriction>
         <wsag:Item>

         <wsag:Item name="file"
          location="/wsag:Agreement/job:JobDescription/job:InputFile">
         <wsag:Item>

      </wsag:NegotiabilityDescription>

</wsag:Agreement>
```

The service consumer of the job description example fills in two values, the items of the NegotiabilityDescription: The input file, which does not have any additional description beyond its type, and the guaranteed number of nodes. The job description language and the constraint language are not part of the WS-Agreement specification. Any language suitable for the application domain can be used.

Both items are defined in the negotiability section. The number of nodes CAN be between 10 and 100.

7 Layered Service Model

7.1 Conceptual Model

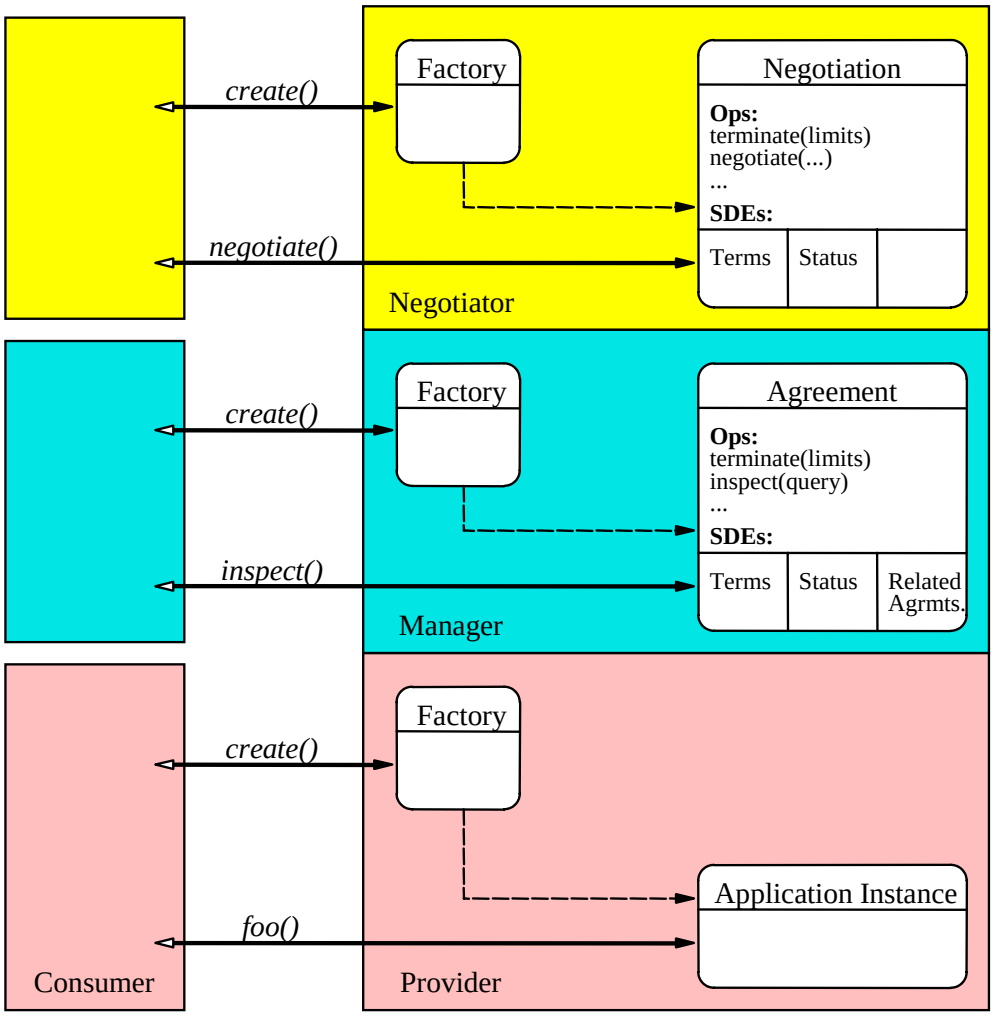


Figure 2: WS-Agreement Conceptual Layered Service Model.

The conceptual model for the architecture of WS-Agreement has three layers (see figure 1), which are from bottom to top:

1. The *service layer* represents the application-specific layer of business service being provided. The class of provided service MAY or MAY NOT be exposed as a Web service interface. For instance, computational jobs may be virtualized as Web service instances, but other legacy services may not be referable as separate instances, let alone be exposed as Web services. Network availability can be seen as a class of service with no Web service representation, but it can be useful to manage its controllable QoS characteristics via agreements defined at layers above the service layer.

   The interface to this layer is domain-specific. This layer MAY be exposed as Web services. If it is, it SHOULD expose port types such as:

   - An application domain-specific service port type virtualizes the concrete service(s) being performed by the provider. It exposes domain-specific operations. For instance the virtualization of a file transfer service into a FileTransfer port type could expose operations such as “suspend”, “resume”, etc. In addition is can expose domain-specific state that the client (which can be a different actor than the initiator) can query or

monitor. For instance a FileTransfer port type could expose a "bytesTransfered" resource property.

- A service is created by a service factory which creation operation takes a set of domain-specific parameters as arguments. For instance: createFileTransferService(sourceURL, destinationURL, ...).

2. The *agreement layer* provides a Web service-based interface that can be used to represent and monitor agreements with respect to provisioning of services implemented in the service layer.
   The agreement layer has the following port types:
   - An agreement port type, without any operation other than getters for state and metadata of the agreement such as the terms, the context, etc….
   - An agreement factory exposes an operation for creating an agreement out of an input set of terms. It returns an EPR to an Agreement service. The agreement factory also exposes resource properties such as the templates of offers acceptable for creation of an agreement.

     The creation parameters can be defined independently of the domain-specific agreement terms defined at the agreement layer. What is merely needed is an unambiguous mapping between the two. The binding between the agreement and the domain-specific service(s) it manages MUST be described in the agreement, and can take alternative forms:

     a. Existing services MAY be referenced by the agreement as part of its terms (thus, these references can be negotiated if it makes sense).
     b. Services MAY be created as per agreement, i.e. the agreement implementation has control over service (instance) creation with the agreement describing the behavior of the newly created service.
     c. Services MAY be created externally but bear domain-specific identifiers enabling the binding of a particular agreement. For instance an agreement on the bandwidth of a computer network can refer to network-specific metadata (such as fields in message headers) as a way to state QoS guarantees on specific network traffic.

3. The *negotiation layer* provides a Web service-based interface for negotiating an agreement so that it eventually satisfies both negotiating parties and become observed, and for renegotiating existing agreements after they have been observed.
   The negotiation layer has the following port types:
   - A negotiation port type exposes a negotiate operation that the initiator can call in order to negotiate the related agreement. Eventually negotiation leads to the agreement being observed (i.e. both parties commit to it). The operation can then be called again in order to renegotiate the agreement (if the implementation service permits it of course).
   - A negotiation factory exposes a negotiation service creation operation which takes an agreement EPR as a parameter. The operation creates a negotiation service related to that specific agreement. The only way for the initiator to modify the agreement is through the negotiation protocol exposed by the negotiation port type.

In this conceptual layering the agreement layer hides the service layer from the negotiation layer. It also decouples the negotiation model from the agreement and service provisioning layers, thus making it possible to swap different negotiation models independently of the agreement model and the service virtualization.

7.2 Practical Model

There are variants in translating this conceptual model of WS-Agreement services into a practical design model. An agreement-based party MAY implement one of the following variants of service design:

1. One factory per layer
2. One factory for both negotiation and agreement creation
3. One factory for both agreement and service creation
4. One factory for all layers

The following subsections explain each design variant.

7.2.1 One Factory per Layer

This design features the same port types as in the conceptual model i.e. each layer is composed of a service port type and its corresponding factory.

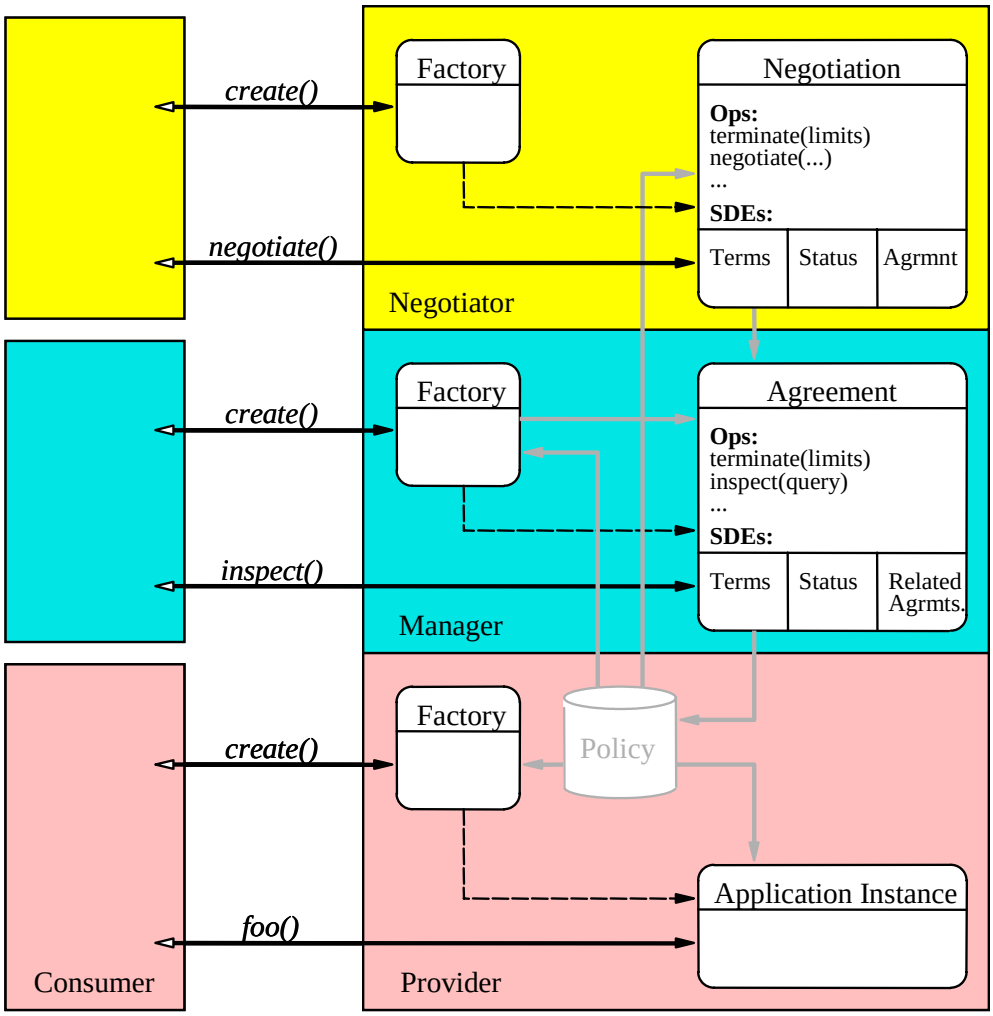


Figure 3: WS-Agreement Service Design 1: One Factory per Layer. Some implementation relationships such as the ones involving a hidden policy repository are shown in grayed.

7.2.2 One Factory for Both Negotiation and Agreement

This variant is the same as variant 1 except that the negotiation factory and the agreement factory are merged into one single factory port type which aggregates the functionality of both, thereby sitting across the negotiation and the agreement layer. Note: in this form of design, swapping negotiation models is not as easy as in variant 1 although the negotiation is still decoupled from the agreement.

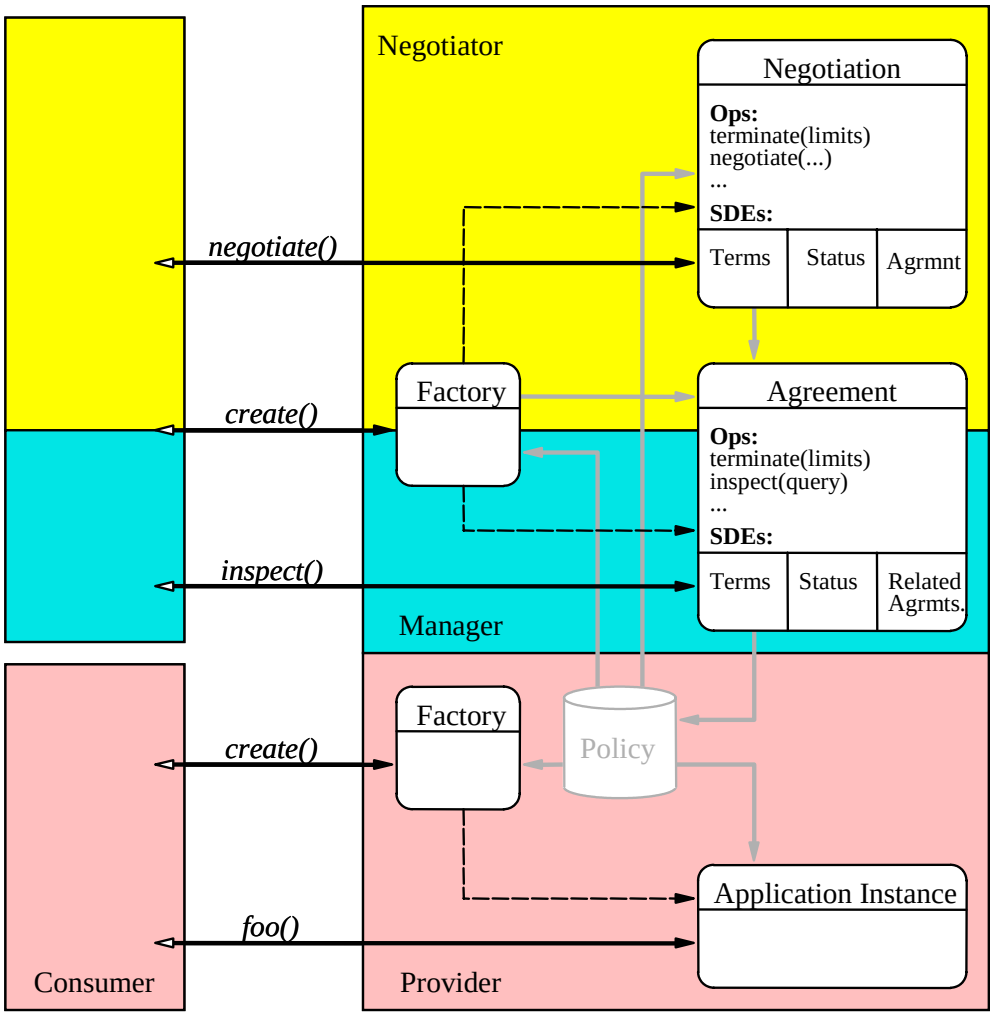


Figure 4: One Factory for both Agreement and Negotiation Creation

7.2.3 One Factory for Both Agreement and Service

This is the same as variant 1 except that the agreement factory and the service factory are merged into one single factory port type which aggregates the functionality of both, thereby sitting across the agreement and the service layer. The factory can expose a service creation operation where the arguments are the agreement terms. Note: such an operation would return an array of services in the case an agreement could manage several services.

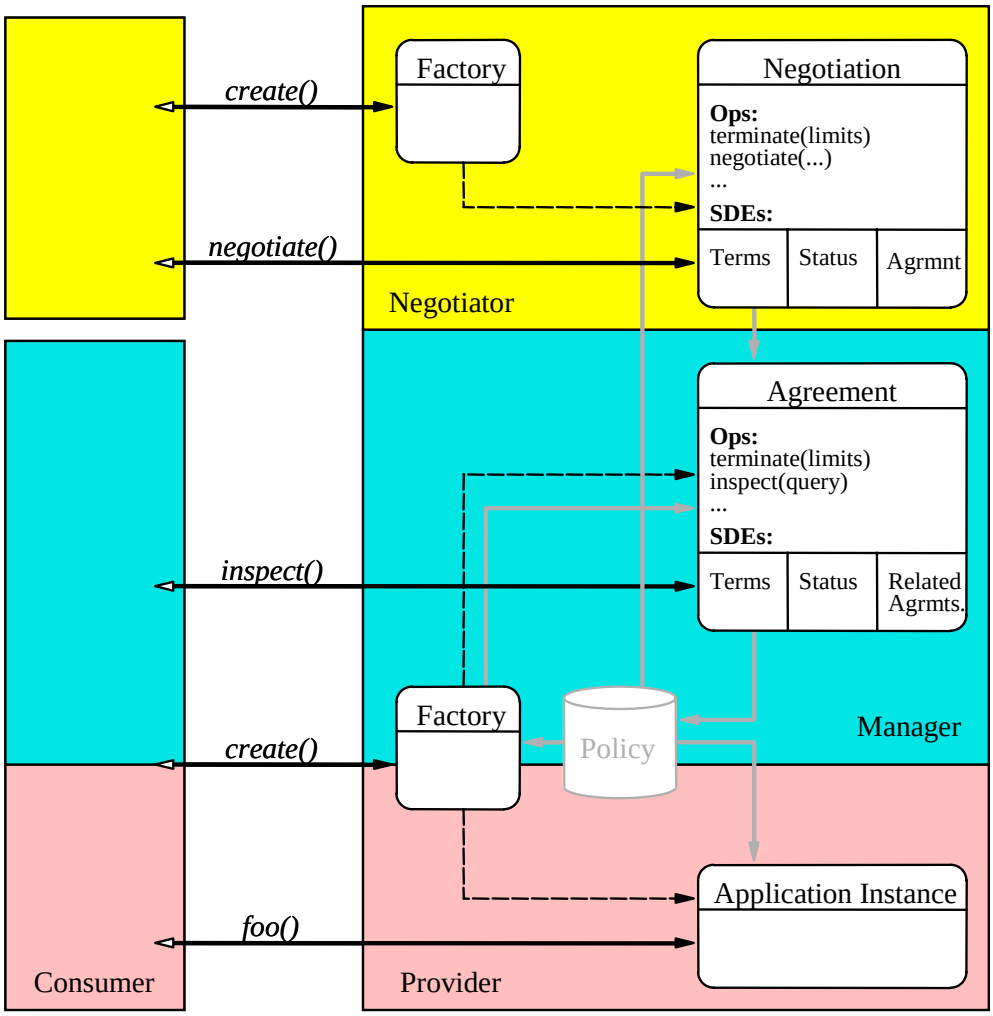


Figure 5: One Factory for both Service and Agreement Creation

7.2.4 One factory for all Layers

In this variant, one single factory port type merges all conceptual factories and acts a façade to the layered system of service creation. This design strategy is applicable only when a default negotiation model is implemented and does not need to be replaced by another.

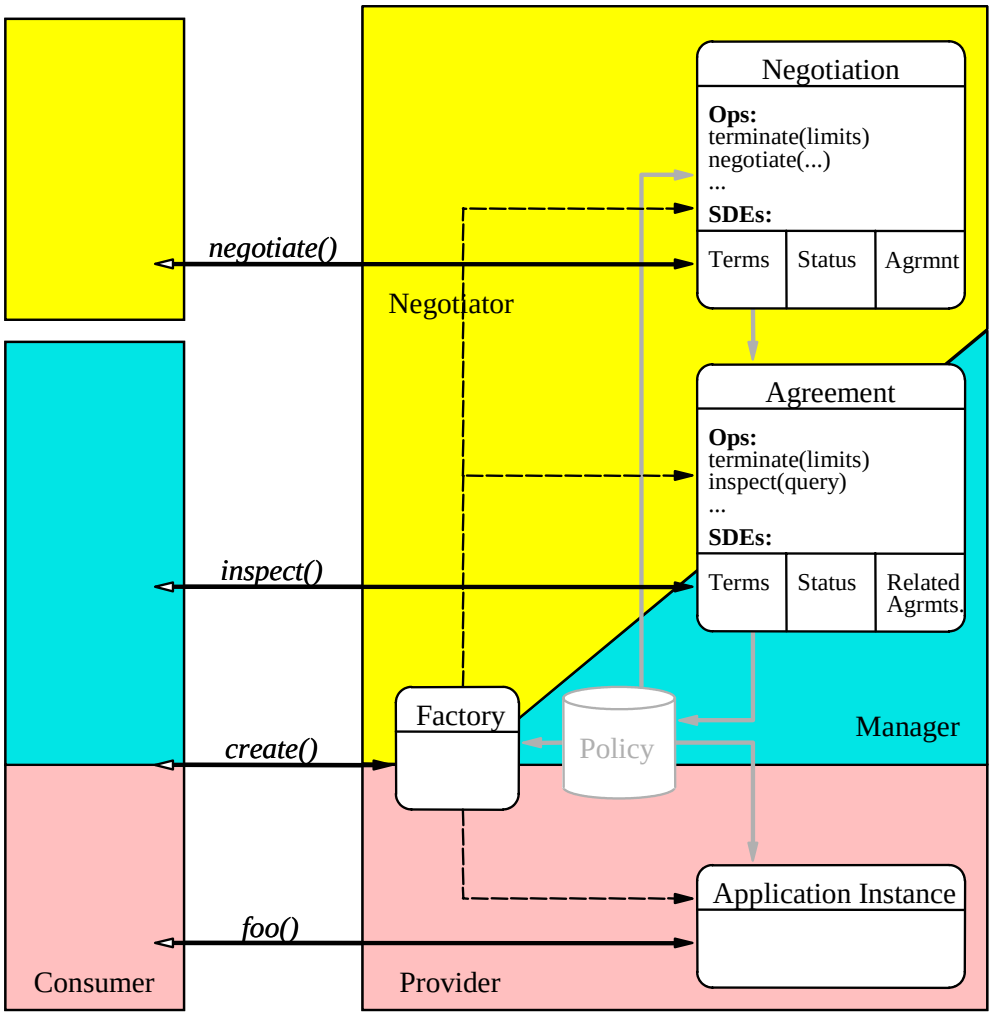


Figure 6: One Factory for all layers

7.2.5 Design Considerations

Each variant preserves the Negotiation and Agreement port types as separate port types, in order to keep the decoupling between negotiation protocol and agreement modeling. The Agreement port type MAY also virtualize the domain-specific service being provided, although the decision to design it as such would depend on the desired strength of the coupling between the agreement and the service.

Variants 2 and 3 increase coupling between layers by merging factories, but potentially increase simplicity of service deployment and reduce the number of network-addressable interfaces exposed publicly.

Because of the multiple possibilities in terms of design of a WS-Agreement system, domain-specific and application-specific decisions SHOULD be made in terms of composition of operation and port type design that cannot be mandated by this specification. This document specifies canonical factories and port types corresponding to the variant 1 explained above. It also specifies one operation (wsg:createNegotiatedAgreement) that can be used in merged factories. Designers of WS-Agreement services MAY reuse WSDL port types, operations, messages, and input/output types specified here although they will always have to define the binding between the agreement and service layer, which is domain-specific.

7.3 Offer Types and Negotiation State

There are six types of offer than can be provided to, or returned from negotiation operations in the WS-Agreement model. For simplicity, each offer type corresponds directly to a state in the negotiation state machine, as depicted in Figure 6.

Note: we define the responder as the party that is invoked by the initiator.

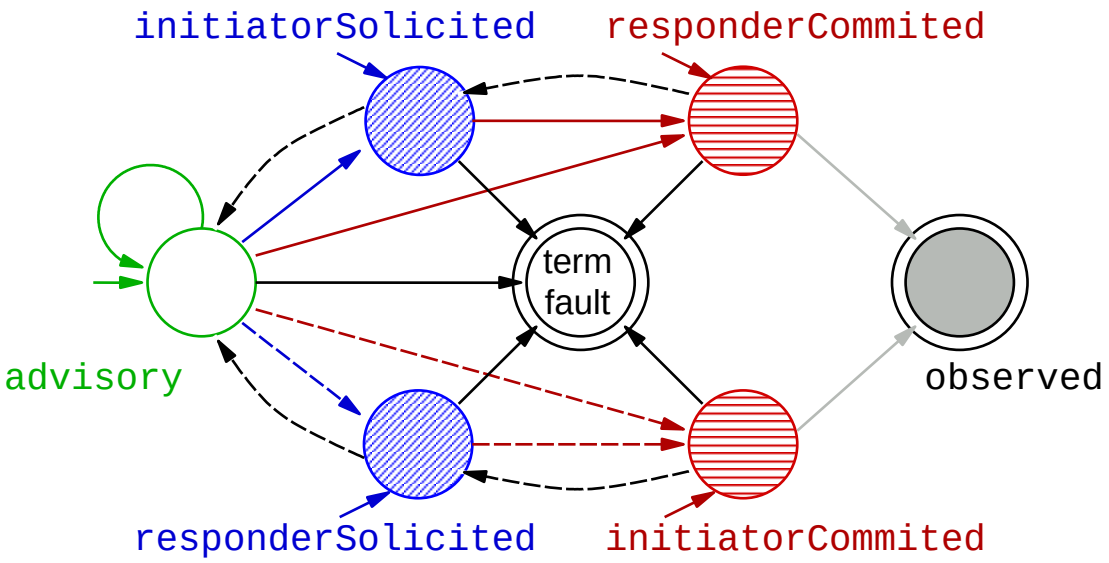


Figure 7: Negotiation Protocol State Machine. The *advisory* start state is changed to *solicited* or *committed* by one of the parties sending an appropriate offer. The terminal *observed* state is reached by acceptance from one of the committed states. The terminal fault state is reached by explicit termination or by terminal faults. A synchronous continuing fault invalidates a state change implied by the faulted offer.

The offer is a suggestion to **enter** the state named in the offer. The offer type is encoded using the ***/@commitment*** attribute:

```
<offer commitment="offer type">
    ...
</offer>
```

The six message types are represented as follows:

1. *Advisory offers* bear the **wsag:advisory** value and indicate no obligations or restrictions on further negotiation.
2. *Soliciting offers* indicate no obligations but require that a counter-offer be committed. There are role-specific solicitation offer types:
    a. *Initiator-solicited* offers are sent by the initiator and bear the **wsag:initiatorSolicited** value.
    b. *Responder-solicited* offers are sent by the responder and bear the **wsag:responderSolicited** value.

3. *Committing offers* indicates that the sender is obligated to the offer terms if the recipient decides to observe. There are role-specific commitment offer types:
    c. *Initiator-committed* offers are sent by the initiator and bear the **wsag:initiatorCommited** value.
    d. *Responder-committed* offers are sent by the responder and bear the **wsag:responderCommited** value.
4. *Accepting offers* bears the **wsag:observed** value and indicates that the sender accepts the offer that has been committed by the recipient.
5. *Termination* uses the underlying WS-RF termination mechanisms and indicates a destruction of all shared Negotiation, Agreement, or Renegotiation state. Third-party resolution, outside the scope of WS-Agreement, may still be used to resolve obligations from terminated Agreements.
6. *Rejection* uses the underlying WS-RF fault mechanisms to signal rejection of an offer, without losing the shared state that existed prior to the rejected offer.

The protocol state machine and operation message requirements restrict the conditions and means by which these offers may be delivered. The protocol states are named according to offer type. Practically speaking, the state of the sender changes when he decides to send an offer of that type, and the state of the receiver changes when he processes a received offer of that type.

Issue 3: Faults are unavoidable in widely distributed systems; we do not wish WS-Agreement to be fragile in the face of such faults, so we include the rejection mechanism. Does there need to be a way to reject offers through an input message, in addition to the fault-response as an output message?

7.4 Canonical Port Types and Operations

In this section we detail the Negotiation and Agreement port types. We also detail the factories of the same layers that correspond to the first variant in designing a WS-Agreement Web service-based interface. Note that designers can reuse the operations defined in those factories and compose them in their own specialized factories.

Per the reuse principles of the WS-Resource Framework on which the Web service expression of this specification is based, interface reuse can be achieved by copying and pasting operation and resource definitions specified here.

Every port type exposes a GetResourceProperty operation as defined in [WS-ResourceProperties]. This enables to expose read-only resource properties. The definition of this operation is identical to the one in [WS-ResourceProperties] and has not been repeated here.

Full WSDL definition of the port types can be found in Appendix.

7.4.1 Port Type wsag:NegotiationFactory

7.4.1.1 Operation wsag:createNegotiation

The wsag:createNegotiation operation is used to generate a wsag:Negotiation related to a wsag:Agreement passed as input.

7.4.1.1.1 Input

Issue : Should we rename “input” messages as “request” messages to follow WSRF habits? I think I prefer input/output as it follows the WSDL naming. What we are using now is a mix: “input”/”response”.

The form of the wsag:createNegotiation input message is:

```
<wsag:createNegotiationInput>
    <initiatorNegotiationEPR>
        EPR1
    </initiatorNegotiationEPR> ?
    <existingAgreementEPR>
        EPR2
    </existingAgreementEPR>
    <firstOffer commitment="offer type">
        ...
    </firstOffer> ?
</wsag:createNegotiationInput>
```

The contents of the input message are further described as follows:

/wsag:createNegotiationInput/initiatorNegotiationEPR

This optional element provides a contact point EPR1 where the invoked party can send messages pertaining to this stateful negotiation (this is applicable only in the case of a symmetric deployment of the wsag:Negotiation port type).

/wsag:createNegotiationInput/existingAgreementEPR

This is the contact point EPR2 of the existing wsag:Agreement to which the wsag:Negotiation to create MUST be related. This element MUST appear.

/wsag:createNegotiationInput/firstOffer

This is the initial offer to start the negotiation with. This is a shorthand for calling the negotiation operation at the returned wsag:Negotiation EPR. If the offer is accepted a counter-offer will be returned. This element MAY be omitted to start a named conversation with no initial state.

/wsag:createNegotiationInput/firstOffer/@commitment

This is the offer type of the agreement offer.

7.4.1.1.2 Result

The successful result of wsag:createNegotiation is the EPR of a newly created wsag:Negotiation.

The form of the response is:

```
<wsag:createNegotiationResponse>
    <createdNegotiationEPR>
        EPR3
    </createdNegotiationEPR>
    <counterOffer commitment="offer type">
        ...
    </counterOffer> ?
```

```
</wsag:createNegotiationResponse>
```

The contents of the response message are further described as follows:

/wsag:createNegotiationResponse/createdNegotiationEPR

This is an endpoint reference EPR3 to a wsag:Negotiation service where the initiator can send messages pertaining to this stateful negotiation. The wag:Negotiation MUST be related to the input wsag:Agreement at EPR2.

/wsag:createNegotiationResponse/counterOffer

This is the agreement offer in response of the optional initial offer.

/wsag:createNegotiationResponse/counterOffer/@commitment

This attribute specifies the offer type of the response offer. The value is governed by the negotiation protocol state machine.

7.4.1.1.3 Faults

A fault response indicates that no wsag:Negotiation was created and may also indicate domain-specific reasons.

7.4.1.2 Operation wsag:createNegotiatedAgreement

The wsag:createNegotiatedAgreement operation is used to generate a wsag:Agreement and a wsag:Negotiation to negotiate it.

7.4.1.2.1 Input

Issue : Should we rename “input” messages as “request” messages to follow WSRF habits? I think I prefer input/output as it follows the WSDL naming. What we are using now is a mix: “input”/”response”.

The form of the wsag:createNegotiatedAgreement input message is:

```
<wsag:createNegotiatedAgreementInput>
    <initiatorNegotiationEPR>
        EPR1
    </initiatorNegotiationEPR> ?
    <initiatorAgreementEPR>
        EPR2
    </initiatorAgreementEPR> ?
    <offer commitment="offer type">
        ...
    </offer> ?
</wsag:createNegotiatedAgreementInput>
```

The contents of the input message are further described as follows:

/wsag:createNegotiatedAgreementInput/initiatorNegotiationEPR

This optional element provides a contact point EPR1 where the invoked party can send messages pertaining to this stateful negotiation. This is applicable only in the case of a symmetric deployment of the wsag:Negotiation port type.

/wsag:createNegotiatedAgreementInput/agreementEPR

This is the optional contact point EPR2 of an existing wsag:Agreement to which the wsag:Negotiation referenced by EPR1 MUST be related. This is applicable only in the case of a symmetric deployment of the wsag:Agreement port type.

/wsag:createNegotiatedAgreementInput/offer

This is the initial offer to start the negotiation with. If the offer is accepted a new wsag:Agreement EPR will be returned as well as the EPR of the new wsag:Negotiation.

/wsag:createNegotiatedAgreementInput/offer/@commitment

This is the offer type of the agreement offer.

7.4.1.2.2 Result

The successful result of wsag:createNegotiatedAgreement is the EPRs of each of a newly created and related wsag:Negotiation and wsag:Agreement.

The form of the response is:

```
<wsag:createNegotiatedAgreementResponse>
    <createdNegotiationEPR>
        EPR3
    </createdNegotiationEPR>
    <createdAgreementEPR>
        EPR4
    </createdAgreementEPR>
    <counterOffer commitment="offer type">
        ...
    </counterOffer> ?
</wsag:createNegotiatedAgreementResponse>
```

The contents of the response message are further described as follows:

/wsag:createNegotiatedAgreementResponse/createdNegotiationEPR

This is an endpoint reference EPR3 to a wsag:Negotiation service where the initiator can send messages pertaining to this stateful negotiation.

/wsag:createNegotiatedAgreementResponse/createdAgreementEPR

This is an endpoint reference EPR4 to a newly created wsag:Agreement service which terms can be negotiated by sending messages to EPR3.

/wsag:createNegotiatedAgreementResponse/counterOffer

This is the counter-offer. This element MAY appear, in which case the agreement MAY include extra negotiability constraints that the next offer sent by the initiator MUST comply with.

/wsag:createNegotiatedAgreementResponse/counterOffer/@commitment

This attribute specifies the offer type of the response offer. This attribute MUST appear in the counter-offer. The value is governed by the negotiation protocol state machine.

7.4.1.2.3 Faults

A fault response indicates that no wsag:Negotiation nor wsag:Agreement was created and may also indicate domain-specific reasons.

7.4.1.3 Resource Property wsag:entry

The wsag:NegotiationFactory port type can create new resource-qualified endpoint references to services (with associated resources) of port types wsag:Negotiation (beginning of a new negotiation in order to eventually reach an agreement). It may be desirable to expose in the interface the created services, for instance for monitoring clients to use. The wsag:NegotiationFactory port type is therefore modeled as a service group with respect to the [WS-ServiceGroup] specification, and records information about each service-resource pair it creates in a new wsag:entry resource property instance. The entry typically includes the EPR of the new qualified service and MAY contain optional information (see the WS-ServiceGroup specification for more information) that this specification does not define.

7.4.1.4 Resource Property wsag:membershipContentRules

This resource property is defined so as to assert the specific content of the wsag:entry resource property and is mandated by [WS-ServiceGroup].

The wsag:membershipContentRules resource property contains a set of wssg:MembershipContentRule elements that specify the intentional constraints on each member service of the service group (see resource property wsag:entry). Each wsgg:membershipContentRule specifies at least a port type that every member service in the service group must implement.

In the context of the wsag:NegotiationFactory, there must be one wsgg:membershipContentRule specifying wsag:Negotiation as the member port type.

The form of the wsag:membershipContentRules resource property is:

```
<wsag:membershipContentRules>
    <wsgg:MembershipContentRule
        MemberInterface="port type"
        ContentElements="qnames"/> *
    <wsgg:MembershipContentRule
        MemberInterface="wsag:Agreement
        ContentElements="qnames"/> +
    <wsgg:MembershipContentRule
        MemberInterface="port type"
        ContentElements="qnames"/> *
</wsag:membershipContentRules>
```

See the [WS-ServiceGroup] specification for more information on the wsgg:MembershipContentRuleType.

7.4.2 Port Type wsag:Negotiation

7.4.2.1 Operation wsag:Negotiate

The wsag:negotiate operation is used to send offers for purpose of negotiating the agreement represented by the wsag:Agreement that the wsag:Negotiation is related to.

7.4.2.1.1 Input

The form of the wsag:negotiate input message is:

```
<wsag:negotiateInput>
   <offer commitment="offer type">
       ...
   </offer>
</wsag:negotiateInput>
```

The contents of the input message are further described as follows:

/wsag:negotiateInput/offer

The input agreement submitted as an offer for negotiation.

/wsag:negotiateInput/offer/@commitment

This is the offer type, which MUST be one of wsag:advisory, wsag:initiatorSolicited, wsag:responderSolicited, wsag:initiatorCommitted, wsag:responderCommited, or wsag:observed as governed by the protocol state machine depicted in Figure 6.

7.4.2.1.2 Result

The successful result of wsag:negotiate is any counter-offer:

```
<wsag:negotiateResponse>
    <counterOffer commitment="offer type">
        ...
    </counterOffer>
</wsag:negotiateResponse>
```

The contents of the result message are further described as follows:

/wsag:negotiateResponse/counterOffer

This is the counter-offer. It MAY be nil, if the wsag:Negotiation wishes to return successfully without issuing a counter-offer. In that case, the wsag:Negotiation is in the state defined by the input offer. The agreement MAY include extra negotiability constraints that the next offer sent by the initiator MUST comply with.

/wsag:negotiateResponse/counterOffer/@commitment

This is the commitment type of the response offer. If the response offer is not nil, it MUST bear an offer type governed by the negotiation protocol state machine.

7.4.2.1.3 Faults

A continuing fault indicates that the wsag:Negotiation was unable to accept the input offer and the state remains unchanged from before the invocation. A terminal fault indicates that the wsag:Negotiation was unable to accept the offer and the wsag:Negotiation will terminate immediately.

7.4.2.2 Resource Property wsag:relatedAgreementEPR

The wsag:relatedAgreementEPR resource property is the contact point of the wsag:Agreement the wsag:Negotiation is related to, i.e. the wsag:Agreement that

the wsag:Negotiation updates if its wsag:negotiate operation is successfully invoked. This resource property is of type wsa:EPRType.

7.4.2.3 ResourceProperty wsag:terms

This resource property represents the offer terms being currently negotiated. They can be changed via the wsag:negotiate operation.

7.4.2.4 ResourceProperty wsag:negotiabilityConstraints

This resource property represents the negotiability constraints on the terms being currently negotiated. They can be changed via the wsag:negotiate operation.

7.4.2.5 Resource Property wsag:negotiationState

The wsag:negotiationState is the status of the negotiation with respect to the commitment of the negotiating parties to the current offer. It is governed by the negotiation protocol state machine (see Figure 6).

7.4.3 Port Type wsag:AgreementFactory

7.4.3.1 Operation wsag:createAgreement

The wsag:createAgreement operation is used to directly generate an Agreement without any intervening Negotiation.

7.4.3.1.1 Input

The form of the wsag:createAgreement input message is:

```
<wsag:createAgreementInput>
    <initiatorAgreementEPR>
        EPR1
    </initiatorAgreementEPR> ?
    <offer commitment="wsag:initiatorCommitted">
        ...
    </offer>
</wsag:createAgreementInput>
```

The contents of the input message are further described as follows:

/wsag:createAgreementInput/initiatorAgreementEPR

This optional element is an endpoint reference (EPR) providing a contact point EPR1 where the invoked party can send messages pertaining to this negotiated Agreement. The invoked party MUST NOT invoke operations on EPR1 after returning a fault on this operation.

/wsag:createAgreementInput/offer

The agreement offer made by the sending party. It MUST satisfy the constraints explicated in one or more of the templates the AgreementFactory exposes. Also, the offer MUST NOT contain negotiability constraints (they do not make sense here since the invoked party is not supposed to, and cannot, reply to this request with a counter-offer).

/wsag:createAgreementInput/offer/@commitment

The value of this attribute specifies the offer type, in terms of commitment. It MUST be wsag:initiatorCommitted. TODO: percolate this reference to state machine into the negotiation layer, because the agreement layer has no concept of negotiation. The offer terms MUST satisfy the negotiability constraints exposed in one or more templates, or be empty.

7.4.3.1.2 Result

The successful result of wsag:createAgreement is a combination of the optional EPR of a newly created Agreement and the acceptance of the initiator's offer:

```
<wsag:createAgreementResponse>
    <createdAgreementEPR>
        EPR2
    </createdAgreementEPR>
</wsag:createAgreementResponse>
```

The contents of the response message are further described as follows:

/wsag:createAgreementResponse/createdAgreementEPR

This is the EPR to a newly created Agreement bearing the same observed terms. This element MUST appear.

/wsag:createAgreementResponse/agreement

The response offer MUST be textually equivalent to the input offer except that the offer type MUST follow the rules of the protocol state machine.

7.4.3.1.3 Faults

A fault response indicates that the offer was rejected and may also indicate domain-specific reasons.

7.4.3.2 Resource Property wsag:template

The templates resource property represents 0 or more templates of offers that can be accepted by the wsag:AgreementFactory operations in order to create an Agreement. A template defines a grouping of certain agreement terms along with negotiability constraints.

Issue: the definition of the template XML Schema type remains to be defined?

7.4.3.3 Resource Property wsag:entry

The wsag:AgreementFactory port type can create new resource-qualified endpoint references to services (with associated resources) of port types wsag:Agreement. It may be desirable to expose in the interface the created Agreements, for instance for monitoring clients to use. The wsag:AgreementFactory port type is therefore modeled as a service group with respect to the [WS-ServiceGroup] specification, and records information about each service-resource pair it creates in a new wsag:entry resource property instance. The entry typically includes the EPR of the new qualified service and MAY contain optional information (see the WS-ServiceGroup specification for more information) that this specification does not define.

7.4.3.4 Resource Property wsag:membershipContentRules

This resource property is defined so as to assert the specific content of the wsag:entry resource property and is mandated by [WS-ServiceGroup].

The wsag:membershipContentRules resource property contains a set of wssg:MembershipContentRule elements that specify the intentional constraints on each member service of the service group (see resource property wsag:entry). Each wsgg:membershipContentRule specifies at least a port type that every member service in the service group must implement.

In the context of the wsag:AgreementFactory, there must be one wsgg:membershipContentRule specifying wsag:Agreement as the member port type.

The form of the wsag:membershipContentRules resource property is:

```
<wsag:membershipContentRules>
    <wsgg:MembershipContentRule
        MemberInterface="port type"
        ContentElements="qnames"/> *
    <wsgg:MembershipContentRule
        MemberInterface="wsag:Agreement
        ContentElements="qnames"/> +
    <wsgg:MembershipContentRule
        MemberInterface="port type"
        ContentElements="qnames"/> *
</wsag:membershipContentRules>
```

See the [WS-ServiceGroup] specification for more information on the wsgg:MembershipContentRuleType.

7.4.4 Port Type wsag:Agreement

The wsag:Agreement port type does not expose any WS-Agreement-specific operations.

7.4.4.1 Resource Property wsag:context

The wsag:context resource property is of type wsag:AgreementContextType. The context is static information about the agreement such as the parties involved in the agreement. See the section in this document about the agreement context.

7.4.4.2 Resource Property wsag:terms

This property specifies the terms of the agreement.

Note: In some application cases it might be worthwhile to decorate a specialized Agreement port types with a QueryResourceProperty operation as defined in [WS-ResourceProperties], in order to expose the terms of the agreement in a more granular way.

Issue: declaration of this resource property requires the existence of a wsag:TermSetType. Should we define such a type or merely reuse wsag:AgreementType and merge the 4 resource properties into one? Should we then create a QueryResourceProperties taking an XPath argument? It seems better to define separate resource properties for the top-level elements of an agreement.

7.4.4.3 ResourceProperty wsag:negotiabilityConstraints

This resource property specifies the constraints that MUST be satisfied by any offer when renegotiating this agreement.

7.4.4.4 Resource Property wsag:agreementState

The commitment state is the state of the agreement. It has a simple value which can be one of the following: wsag:observed, wsag:beforeObserved, wsag:afterObserved. It is of type wsag:AgreementType.

Issue: Do we want to have this as a separate property like this, or merely as an attribute in the element wsag:agreement that we would expose as a resource property? If we want to expose the commitment state by itself, we can also define QueryResourceProperty so that clients can do XPath queries (but it put some burden on implementers to implement XPath query, and I don't know if mandating it is a good idea). Or we can have an operation wsag:getCommitmentState. Or we can just say that designers are free to implement one of these two operations in specialized Agreement port types if they so desire, but we are not mandating it in the wsag:Agreement port type.

7.4.4.5 Resource Property wsag:entry

A wsag:Agreement can be related to others wsag:Agreement for chaining or composition. (how much do we want on this topic in the spec?). This one-to-many relationship is modeled as a service group (see [WS-ServiceGroup]), and records information about each service-resource pair in a wsag:entry resource property instance. An entry includes the EPR of a related wsag:Agreement and MAY contain optional information that this specification does not define.

7.4.4.6 Resource Property wsag:membershipContentRules

This resource property is defined so as to assert the specific content of the wsag:entry resource property (see [WS-ServiceGroup]).

The wsag:membershipContentRules resource property contains a set of wssg:MembershipContentRule elements that specify the intentional constraints on each member service of the service group (see resource property wsag:entry). Each wsgg:membershipContentRule specifies at least a port type that every member service in the service group must implement.

In the context of the wsag:Agreement, there must be one wsgg:membershipContentRule specifying wsag:Agreement as the member port type.

The form of the wsag:membershipContentRules resource property is:

```
<wsag:membershipContentRules>
    <wsgg:MembershipContentRule
        MemberInterface="port type"
        ContentElements="qnames"/> *
    <wsgg:MembershipContentRule
        MemberInterface="wsag:Agreement
        ContentElements="qnames"/> +
    <wsgg:MembershipContentRule
        MemberInterface="port type"
        ContentElements="qnames"/> *
</wsag:membershipContentRules>
```

See the [WS-ServiceGroup] specification for more information on the wsgg:MembershipContentRuleType.

8 Common Use Cases

In this section we present common usage patterns of the WS-Agreement service model.

Note: the binding between the agreement and service layer being out of the scope of this specification, we omit the steps and operations that expose service layer services or application functionality. Suggestions include using the [WS-ServiceGroup] idiom to have the Agreement service expose the list of services it binds to.

8.1 Simple Agreement Creation

Note: In this simple use case where no multi-round negotiation capability needs to be implemented, we assume a design based on variant 3 (as explained in the previous chapter) where we omit the negotiation layer of the WS-Agreement service stack.

The merged Factory MAY be a domain-specific specialization of the AgreementFactory described in the port types section of this document. In particular it MAY choose to replicate/reuse the wsag:createAgreement operation.

Process:

1. The initiator is interested in obtaining an agreement for service provisioning with the party implementing the factory. In order to create an agreement in one shot, the initiator calls the createAgreement operation on the Factory service, passing in offer terms that satisfy the negotiability constraints of one the templates exposed by the Factory as resource properties. Since there is no negotiation layer, the offer is committed by the initiator. If it is not accepted by the Factory, the createAgreement operation will not return any counter-offer but merely a fault.
2. Assuming the factory accepts the terms, it returns an endpoint reference (EPR) to an *observed* Agreement service.

8.2 Agreement Negotiation

Note: In this use case with negotiation, we assume a design based on variant 2.

The merged Factory MAY compose the wsag:createNegotiatedAgreement operation defined in the canonical wsag:NegotiationFactory port type.

Process:

1. The initiator calls the createAgreementAndNegotiation operation on the Factory service.
1. The Factory service returns an EPR to an Agreement and an EPR to a Negotiation.
2. The initiator calls the negotiate operation on the Negotiation service in order to change the current state of the agreement: the terms being negotiated or the commitment status. The Negotiation service either rejects the offer using

a non-terminating i.e. continuing fault or accepts the offer and updates the state of the Agreement.

3. Step 3 is repeated until one party decides to stop negotiation or both parties commit to the current offer. For example the Negotiation service can send a terminal fault, indicating unwillingness to accept any further message.
4. Eventually both parties commit to an offer and the agreement becomes *observed*.

8.3 Agreement Renegotiation

Since renegotiation can occur whether initial negotiation took place or not, we can illustrate renegotiation of an existing agreement by reusing either of the two previous designs exemplified respectively in the simple agreement creation use case and in the agreement negotiation use case. However, there SHOULD be an additional operation in the Factory port type for the initiator to obtain the EPR to a Negotiation service in case it lost the Negotiation EPR obtained when requesting creation of the agreement (agreement negotiation use case) or if it never requested a Negotiation in the first place (simple agreement creation use case). Therefore, the Factory MAY choose to compose an equivalent of the wsag:createNegotiation operation defined in the wsag:NegotiationFactory.

Process:

1. The steps in the simple agreement creation use case or the steps in the negotiation use case are used here.
2. The initiator calls the createNegotiation operation on the Factory in order to retrieve the EPR to a Negotiation service related to the agreement which EPR was supplied to the operation.
3. Negotiation iterative process: Steps 3-5 of the negotiation use case are performed.

9 Acknowledgements

This document is the work of the GRAAP Working Group GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Scheduling and Resource Management (SRM) Area of the GGF.

Members of the Working Group are (at the time of writing, and by alphabetical order): Alain Andrieux, (Globus Alliance / USC/ISI), Takuya Araki (ANL), Carl Czajkowski, (Globus Alliance / USC/ISI), Asit Dan (IBM), Kate Keahey (Globus Alliance / ANL), Chris Kurowski (PSNC), Heiko Ludwig (IBM), Jon MacLaren (University of Manchester), Steven Newhouse (London e-Science Centre), Steven Pickles (University of Manchester), Jim Pruyne (HP), John Rofrano (IBM), Volker Sander (*Forschungszentrum Jülich *), Chris Smith (Platform Computing), Steve Tuecke (Globus Alliance / ANL), Alan Weissberger (NEC), Ming XU (Platform Computing), Wolfgang Ziegler (*Fraunhofer–Institute*).

Contributions of the following people are also acknowledged (alphabetically): Ian Foster (ANL), Robert Kearney (IBM), David Kaminsky (IBM), Carl Kesselman (ANL/USC/ISI), Miron Livny (University of Wisconsin), Jeff Nick (IBM), Ellen Stokes (IBM), John Sweitzer (IBM).

10 References

[SOAP 1.2]
http://www.w3.org/TR/soap12-part1/

[URI]
T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.
http://www.ietf.org/rfc/rfc2396.txt

[WS-Agreement-old]
http://forge.gridforum.org/docman2/ViewProperties.php?group_id=71&document_content_id=358

[SNAP]
K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke:
"SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems"
http://www.isi.edu/~karlcz/papers/snap-lncs-25370153.pdf

[WS-Addressing]
http://www.ibm.com/developerworks/webservices/library/ws-add/

[WS-Resource]
http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

[WS-ResourceLifetime]
http://www.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf

[WS-ResourceProperties]
http://www.ibm.com/developerworks/library/ws-resource/ws-resourceproperties.pdf

[WS-BaseFaults]
URL to specification when on-line

[WS-ServiceGroup]
URL to specification when on-line

[WS-Notification]
http://www.ibm.com/developerworks/library/ws-resource/ws-notification.pdf

[WS-Security]
http://www.ibm.com/developerworks/webservices/library/ws-secure/

[XML-Infoset]
http://www.w3.org/TR/xml-infoset/

[XML]
http://www.w3.org/TR/REC-xml

[XML-ns]
http://www.w3.org/TR/1999/REC-xml-names-19990114

[XPath]
http://www.w3.org/TR/xpath

Appendix 1 - Document Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.ggf.org/ws-agreement"
        xmlns:wsag="http://www.ggf.org/ws-agreement"
        xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">


      <import schemaLocation="addressing.xsd"
namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"></import>
      <import schemaLocation="XMLSchema.xsd"
namespace="http://www.w3.org/2001/XMLSchema"></import>

      <complexType name="AgreementType">
            <sequence>
            <element name="Context"
                  type="wsag:AgreementContextType"></element>
                  <element name="Terms"
                        type="wsag:TermCompositorType"></element>
                  <element name="NegotiabilitySection"
                        type="wsag:NegotiationSectionType"
                        minOccurs="0"></element>
            </sequence>
      </complexType>

      <element name="Agreement" type="wsag:AgreementType"></element>

      <complexType name="AgreementContextType">
            <sequence>
                  <element name="AgreementInitiator"
                        type="anyURI"></element>
                  <element name="AgreementProvider"
                        type="anyURI"></element>
                  <element name="TerminationTime"
                        type="dateTime"></element>
            </sequence>
      </complexType>

      <complexType name="TermCompositorType">
```

```
            <sequence>
                  <choice>
                        <element name="ExactlyOne"
                              type="wsag:TermCompositorType"></element>
                        <element name="OneOrMore"
                              type="wsag:TermCompositorType"></element>
                        <element name="All"
                              type="wsag:TermCompositorType"></element>
                        <element ref="wsag:Term"
                              maxOccurs="unbounded"></element>
                  </choice>
            </sequence>
      </complexType>

      <complexType name="TermType" abstract="true">
            <attribute name="name" type="string"></attribute>
      </complexType>

      <element name="Term" type="wsag:TermType"
            abstract="true"></element>

      <complexType name="GuaranteeTermType">
            <complexContent>
                  <extension base="wsag:TermType">
                        <sequence>
                              <element
                              ref="wsag:QualifyingCondition"></element>
                              <element
                                    ref="wsag:ServiceLevelObjective">
                              </element>
                              <element name="BusinessValueList"
                                    type="wsag:BusinessValueListType">
                              </element>
                        </sequence>
                  </extension>
            </complexContent>
      </complexType>

      <element name="GuaranteeTerm"
            type="wsag:GuaranteeTermType"
            substitutionGroup="wsag:Term"></element>

      <element name="QualifyingCondition" type="anyType"></element>
```

```
<element name="ServiceLevelObjective" type="anyType"></element>

<complexType name="BusinessValueListType">
      <sequence>
            <element name="Importance" type="integer"
                  minOccurs="0"></element>
            <element name="Penalty" type="wsag:CompensationType"
                  minOccurs="0"></element>
            <element name="Reward" type="wsag:CompensationType"
                  minOccurs="0"></element>
            <element ref="wsag:BusinesValue" minOccurs="0"
                  maxOccurs="unbounded"></element>
      </sequence>
</complexType>

<element name="BusinesValue" type="anyType"></element>

<complexType name="CompensationType">
      <sequence>
            <element name="AssessmentInterval">
                  <complexType>
                        <sequence>
                              <choice>
                                    <element name="TimeInterval"
                                    type="anyType"></element>
                                    <element name="Count"
                                    type="integer"></element>
                              </choice>
                        </sequence>
                  </complexType>
            </element>
            <element name="ValueUnit" type="string"
                  minOccurs="0"></element>
            <choice>
                  <element name="Value" type="float"></element>
                  <element name="ValueExpression"
                        type="anyType"></element>
            </choice>
      </sequence>
</complexType>

<complexType name="ServiceDescriptionTermType">
```

```
            <complexContent>
                  <extension base="wsag:TermType">
                        <sequence>
                              <element ref="wsa:EndpointReference"
                                    minOccurs="0"></element>
                              <element ref="wsag:ServiceDescription"
                                    maxOccurs="unbounded"
                                    minOccurs="0"></element>
                              <element name="Variables"
                                    type="wsag:VariableSetType"
                                    minOccurs="0"></element>
                        </sequence>
                  </extension>
            </complexContent>
      </complexType>

      <element name="ServiceDescriptionTerm"
            type="wsag:ServiceDescriptionTermType"
            substitutionGroup="wsag:Term"></element>

      <element name="ServiceDescription" type="anyType"></element>

      <complexType name="VariableSetType">
            <sequence>
                  <element name="Variable" type="wsag:VariableType"
                        maxOccurs="unbounded" />
            </sequence>
      </complexType>

      <complexType name="VariableType">
            <simpleContent>
                  <extension base="string">
                        <attribute name="name" type="NCName" />
                        <attribute name="metric" type="QName" />
                  </extension>
            </simpleContent>
      </complexType>

      <complexType name="NegotiationSectionType">
            <sequence>
                  <element name="Item"
                  type="wsag:NegotiationItemType"></element>
            </sequence>
```

```
      </complexType>

      <complexType name="NegotiationItemType">
            <sequence>
                  <group ref="simpleRestrictionModel"
                        minOccurs="0"></group>
            </sequence>
            <attribute name="name" type="string"></attribute>
            <attribute name="path" type="string"></attribute>
      </complexType>

      <element name="Constraint" type="anyType"></element>

</schema>
```

Appendix 2 - WSDL

Appendix 3 - Example