GFD-R-P.107

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

Grid Resource Allocation Agreement Protocol (GRAAP) WG
http://forge.gridforum.org/sf/projects/graap-wg
March 14, 2007

Web Services Agreement Specification (WS-Agreement)

Status of This Memo

This document is the WS-Agreement Specification from the Open Grid Forum (OGF). This is a public document being developed by the participants of the GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Compute Area of the OGF.

Abstract

This document describes Web Services Agreement Specification (WS-Agreement), a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties. The specification consists of three parts which may be used in a composable manner: a schema for specifying an agreement, a schema for specifying an agreement template, and a set of port types and operations for managing agreement life-cycle, including creation, expiration, and monitoring of agreement states.

graap-wg@ogf.org

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 simply be advertised as an invariant property of a service and then bound to by a service consumer. Instead, the service consumer must obtain state-dependent guarantees from the service provider, represented as an agreement on the service and the associated guarantees. Additionally, the guarantees on service quality should be

monitored and service consumers may be notified of failure to meet these guarantees. The objective of the WS-Agreement specification is to define a language and a protocol for advertising the capabilities of service providers and creating agreements based on creational offers, and for monitoring agreement compliance at runtime.

An agreement between a service consumer and a service provider specifies one or more service level objectives both as expressions of requirements of the service consumer and assurances by the service provider on the availability of resources and/or on service qualities. For example, an agreement may provide assurances on the bounds of 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 a set of terms. The terms MAY comprise one or more service terms and zero 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, which we call agreement creation constraints. This specification defines a schema for an agreement template.

The creation of an agreement can be initiated by the service consumer side or by the service provider side, and the protocol provides hooks enabling such symmetry.

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 specifying alternative service description terms and use of logical grouping operators, and agreement creation constraints in negotiating service level objectives. Details of the example scenario are described in section 2. Section 3 introduces the layered model of WS-Agreement. Section 4 provides the overall agreement structure, service description as agreement terms and guarantee terms, respectively. Section 5 specifies the schema for the agreement template and agreement creation constraints. Section 6 defines compliance and section 7 defines runtime states of the overall agreement and its terms. Section 8 defines acceptance model, i.e., establishment protocol. Section 9 introduces the port types and operations in the specification. Section 10 describes the process leading to the creation of an agreement. Section 11 addresses security considerations with respect to using WS-Agreement.

1.1 Goals and Requirements

The goal of WS-Agreement is to standardize the terminology, concepts, overall agreement structure with types of agreement terms, agreement template with creation constraints and a set of port types and operations for creation, expiration and monitoring 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 term:** It must be possible to create agreements for services defined by any domain specific service terms, such as job specification, data service specification, network topology specification and Web Service Description Language (WSDL **[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.
- **Must provide symmetry of protocol:** A large number of scenarios are possible depending on whether a service 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.
- **Must be composable with various negotiation models:** it must be possible to design negotiation protocols which compose with schemas defined by WS-Agreement.
- **Must be standalone**: simple agreement creation must be supported in the WS-Agreement specification, independent of any negotiation model.
- **Must allow independent use of different parts of the specification**: The specification of the agreement document structure can be used independently of the protocol defined here.

Relationship to other WS-* specifications: The WS-Agreement protocol is dependent on WS-Addressing **[WS_Addressing]** and WS-ResourceProperties **[WS-ResourceProperties]**, WS-ResourceLifetime **[WS-ResourceLifetime]**, and WS-BaseFaults **[WS-BaseFaults]**. In particular, WS-ResourceProperties and WS-ResourceLifetime are used to represent Agreements as Resources. WS-Agreement is also meant to be composable with other Web services specifications.

External Specification	Standards Body	Status	Is used for
WS-ResourceProperties 1.2 (WS-RF RP)	Became an OASIS Standard 1st April 2006	Institutional Standard	Resource properties on port types
Web Services Addressing 1.0 Core (WS-Addressing 1.0 Core)	Became a W3C Recommendation May 2006	Institutional Standard	End point references to resource-qualified services

Web Services Resource Lifetime 1.2 (WS-RF RLF)	Became an OASIS Standard 1st April 2006	Institutional Standard	Factory pattern and destroy operation for resources
Web Services Base Faults 1.2 (WS-RF BF)	Became an OASIS Standard 1st April 2006	Institutional Standard	Defines the Basic faults

Institutional Standard: An *approved* specification from a generally recognized standards development organization with open membership.
(cf. "OGSA Profile Definition 1.0." **[OGSA Profile]**).

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 languages may be used in different usage domains.
- 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.
- Defining a protocol and conventions for claiming domain-specific services according to agreements. For example, agreement identification in SOAP **[SOAP]** headers might suit a Web service, another mechanism is required for networking services, etc.
- Defining a protocol for negotiating agreements.

1.2 Notational Conventions and Terminology

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 **[XPath]** (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of {any} indicates the presence of an element wildcard (<xs:any/>). The use of @{any} indicates the presence of an attribute wildcard (<xs:anyAttribute/>). <xs:any##other> is a notational convention for the XML Schema **[XML Schema]** equivalent of <xs:any namespace=##other/>.

Furthermore, this specification defines and uses the following terms:

Acceptance (Agreement Acceptance). Agreement acceptance is the decision of the agreement responder to participate in an agreement relationship with an initiator as defined in the offer made by the initiator. There are both synchronous and asynchronous mechanisms for the responder to signal this acceptance decision to the initiator.

Agreement. An agreement defines a dynamically-established and dynamically-managed relationship between parties. The object of this relationship is the delivery of a service by one of the parties within the context of the agreement. The management of this delivery is achieved by agreeing on the respective roles, rights and obligations of the parties. The agreement may specify not only functional properties for identification or creation of the service, but also non-functional properties of the service such as performance or availability. Entities can dynamically establish and manage agreements via Web service interfaces.

Business value. The business value is intended to represent the strength of an agreement in domain-specific terms. In general, business value is an assertion representing a value aspect of a service level objective attached to the service that is the subject of the agreement. The value may be specified in terms of domain-specific qualities such as importance, cost and others. Each service level objective may have a list of business values attached to it, representing different value aspects of this objective. Both agreement initiator and agreement responder may specify business values.

Consumer (Service Consumer). A service consumer is an entity entering into an agreement with the intent of obtaining guarantees on the availability of certain services from the service provider. The agreement is established between an agreement initiator and agreement responder acting on behalf of the service consumer and service provider. Either initiator or responder role may act on behalf of a service provider, or consumer, depending on domain-specific signaling requirements. The consuming or providing parties of the service domain may act directly as initiating or responding parties of the WS-Agreement domain or may be represented by proxies.

Constraints (Agreement Creation Constraints). Agreement creation constraints define a set of acceptable values for the agreement terms. They are represented in a separate and optional element of the agreement template and refer back to individual terms to which they apply using XPATH. Agreement constraints do not represent a promise on the part of the agreement responder that an agreement creation request will be accepted; they lay down rules which should be followed in the creation of an agreement, but the acceptance of the individual term values is dependent on the state of the provider.

Context (Agreement Context). Agreement context contains information about agreement parties, the agreement's lifetime, and (optionally) a reference to the template from which the agreement is created.

Creation (Agreement Creation). Agreement creation is the process defined in this specification to allow the two parties of agreement initiator and responder to form a new agreement. The overall process starts (optionally) with the initiator retrieving a template from the responder, continues with the initiator making an offer via one of

the agreement creation operations, and is concluded when the responder accepts the offer.

Expiration (Time). Expiration time defines a time when an agreement is no longer valid, and the parties are no-longer obligated by the terms of the agreement.

Guarantee (Guarantee Terms). Guarantee terms define the assurance on service quality (or availability) associated with the service described by the service definition terms. They refer to the service description that is the subject of the agreement and define service level objectives (describing for example the quality of service on execution that needs to be met), qualifying conditions (defining for example when those objectives have to be met) and business value expressing the importance of the service level objectives.

Initiator (agreement initiator). An agreement initiator is a party to an agreement. The initiator creates and manages an agreement on the availability of a service on behalf of either the service consumer or service provider, depending on the domain-specific signaling requirements. The initiator invokes the createAgreement or createPendingAgreement operations from this specification.

Offer (Agreement Offer). An offer is the description of the agreement relationship that is sent from initiator to responder during agreement creation, indicating the relationship which the initiator would like to form. This offer is accepted or rejected by the responder.

Parties (Agreement Parties). Agreement parties consist of the agreement initiator and agreement responder.

Provider (Service Provider). A service provider is an entity entering into an agreement with the intent of providing a service according to conditions described by the agreement.

Rejection (Agreement Rejection). Agreement rejection is the complement of the acceptance process wherein the responder decides not to participate in the agreement relationship described in the initiator's offer.

Responder (Agreement Responder). The agreement responder is a party to an agreement. The responder implements and exposes an agreement on behalf of either the service provider or service consumer, depending on the domain-specific signaling requirements. The responder implements the WS-Agreement service against which the initiator invokes the createAgreement operation. The createAgreement operation follows the factory pattern.

Service Description Terms. Service Description Terms describe the functionality that will be delivered under the agreement. The agreement description may include also other non-functional items referring to the service description terms.

Service Level Objective (SLO). Service Level Objective represents the quality of service aspect of the agreement. Syntactically, it is an assertion over the terms of the agreement as well as such qualities as date and time.

Template (Agreement Template). An agreement template is an XML **[XML]** document used by the agreement responder to advertise the types of offers it is

willing to accept. Like an agreement document, the template is composed of a template name, a context element, and agreement terms, but additionally also includes information on agreement creation constraints to describe a range of agreements it might accept.

Terms (Agreement Terms). Agreement terms define the content of an agreement. It is expected that most terms will be domain-specific defining qualities such as for example service description, termination clauses, transferability options and others.

1.3 Namespace

This is an XML or other code example:

```
http://schemas.ggf.org/graap/2007/03/ws-agreement (Code)
```

The following namespaces are used in this document:

Prefix	Namespace
wsag	http://schemas.ggf.org/graap/2007/03/ws-agreement
wsa	http://www.w3.org/2005/08/addressing/
wsrf-bf	http://docs.oasis-open.org/wsrf/bf-2
wsrf-rp	http://docs.oasis-open.org/wsrf/rp-2
wsrf-rw	http://docs.oasis-open.org/wsrf/rw-2
wsrf-rpw	http://docs.oasis-open.org/wsrf/rpw-2
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 range 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 document format and a protocol comprising few states. Three examples are chosen here to illustrate the range of applications that this specification covers. These examples are referred to throughout the specification.

Note: in the examples we will assume that the service provider acts as the agreement responder, and the service consumer as the agreement initiator.

2.1 Job Submission

An existing agreement-like scenario is the submission of a job to a batch processing system. This job submission process can be recast as agreement creation, where each agreement represents the requirements and obligations for completing one job. The job hosting service may, as or via an agreement responder, post an agreement template describing the range of job offers it may accept. Job submitters, as or via an agreement initiator, make offers describing jobs to be run. The job hosting

service, via the responder role, has the opportunity to consider the job offer and decide whether to accept or reject it.

The job agreement, agreement offer, and agreement template would all include service definition terms expressed in an appropriate job description language. This language encodes the conventional details of the job such as the nature of the process to be executed, the resources required for execution, and any scheduling requirements such as job-start or job-completion deadlines. Upon acceptance, the resulting agreement service may be used to monitor the delivery of service required by these terms, e.g. the lifecycle of the actual job.

2.2 Advance Reservation or Pre-Establishment of Resource Preferences

Another related scenario is to perform job submission within the context of an existing (or *advance*) reservation of capability or pre-established resource preferences. The primary difference from the earlier example is that the submitting party knows that he has an ongoing relationship with the job hosting service, and can expect his job offer(s) to be accepted as long as the requested terms are kept within certain limits set by the relationship. For example, the reservation might guarantee availability of a certain kind of resource on a certain schedule, or with a particular cost model. Reservation is an abstraction for understanding this refined expectation about the handling of future jobs; whether the job hosting service uses preemption, predictive models, or the literal setting aside of resources is an implementation decision for the service. Another use of pre-established agreement is to specify resource preferences, e.g., choice of nodes with a certain amount of memory over others, via an agreement, that are to be used in all subsequent resource allocation to incoming jobs in the context of this agreement.

Whether or not agreements are used to represent the individual submitted jobs, agreements may be used to represent the formation and management of this ongoing reservation relationship. The job hosting service, as or via an agreement responder, may post an agreement template available to interested initiators. In this scenario, the agreement template defines a resource preference or commitment that can be used to establish an agreement with the resource provider for multiple subsequent job submissions. The template may include limits on available resources, or in some cases, fixed allocation of resource types and quantity. A resource preference and commitment profile may include a quality of service guarantee in terms of number of nodes and/or per node memory and storage for a specific time period. It may also include an expression of preferences over resource amounts, for example, a node with twice the memory size as a basic node could be valued three times more highly.

Alternatively, the guarantees can be on the completion time.

All subsequent job submissions under this resource agreement, (that further specify the name of an executable, input and output files, and additional dependencies on software environment) will be used by the resource provider to allocate resources for execution of these jobs. Resource preference and commitment agreements associated with multiple waiting jobs are used in matching available resources and improving overall business value of the provider.

2.3 Service Parameterization

In this scenario, the service contracted is an application service provided by a financial company. The service consists of online banking and investment, where

online banking service operations are accessible via a web browser and investment operations as web services. Online banking operations, such as UpdateUserProfile, GetAcctBalance, SchedulePayment, GetTransactionHistory and GetPaymentHistory are exposed via a portal. Investment operations, such as StockQuote, BuyShares and SellShares are exposed as web services using the Web Service Description Language (WSDL).

The financial company offers several service levels, such as *Gold, Silver, Bronze* etc, where each service level requires a minimum investment amount and/or account balance, also offers different banking and investment fee structures. Additionally, each service level provides a certain Quality of Service (QoS), described via an agreement template, specifying the service and its guarantees, including the QoS options available to the customer. When new customers open accounts with the financial institution, they select a service level by customizing the options specified in the template. Customers can add availability and response time guarantees to individual operations of the interface. For availability, customers may choose between 95%, 98%, 99%, and 99.9%, defined as the probability of receiving a 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 (especially for web service operations) for which the response time goal must hold. Also, customers can set the time when the service will be available such as 8AM to midnight daily.

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 agreement creation offer or reject it. 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.

If the agreement offer is accepted by the provider, the provider provisions the service and exposes status information on guarantee compliance to the user. If not, the offer is rejected, and the customer may create a new offer with different desired service levels.

3 Layered Model

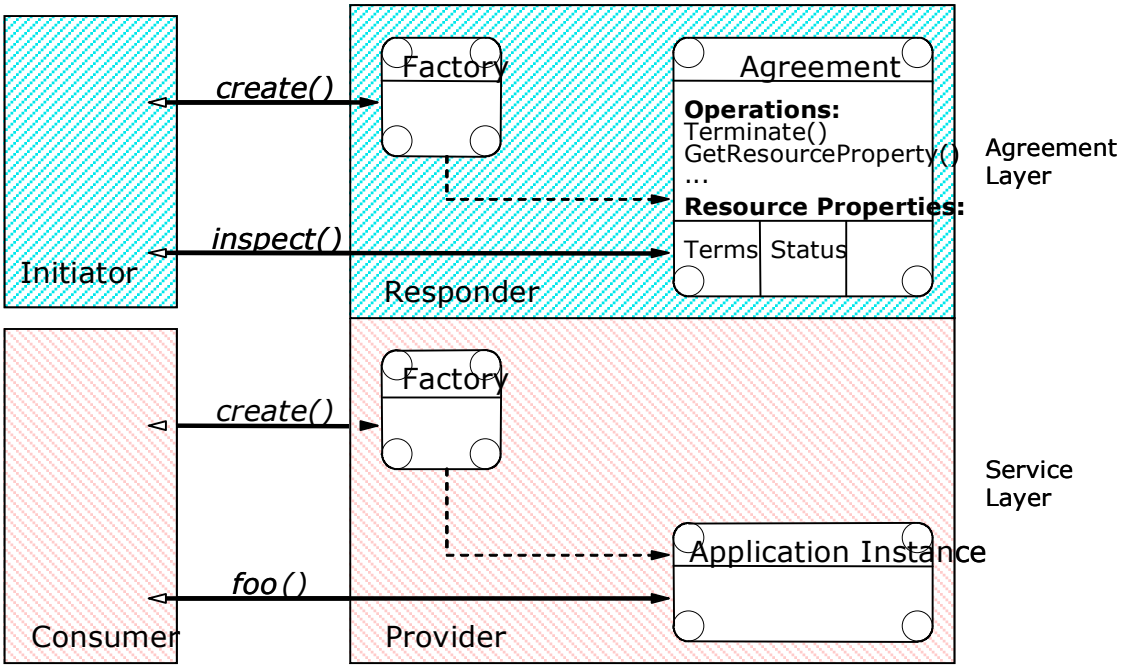


Figure 1: WS-Agreement Conceptual Layered Service Model.
Note: The names of the different operations and "attributes" are not normative, nor is the assignment of initiator and responder roles to service consumer and provider.

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

1. The *agreement layer* provides a Web service-based interface that can be used to create, represent and monitor agreements with respect to provisioning of services implemented in the service layer. The agreement layer has the following port types, as detailed later in this specification:
   - An agreement factory exposes an operation for creating an agreement out of an initial set of terms. It returns an Endpoint Reference (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 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 desired).
     b. Services MAY be created as per agreement, i.e. the agreement layer 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.

- An agreement port type, without any operation other than getters for runtime state and metadata of the agreement. Other domain-specific management operations MAY be added to this resource.

2. The *service layer* represents the application-specific layer of the service being provided. The class of provided service MAY or MAY NOT be a Web service interface. For instance, computational jobs in the advance reservation scenario may be virtualized as Web service instances with additional, domain-specific port-types; these jobs are the service layer associated with the agreement layer that manages the reservation. Other services may not have a service oriented representation. Network availability can be seen as a class of service with no Web service representation, but it can be useful to manage its controllable Quality of Service (QoS) characteristics via agreements defined at layers above the service layer.

   The interfaces in this layer are domain-specific, and need not be altered when the agreement layer is introduced.

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 for the agreement layer. 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.

Once agreements are established in the agreement layer, the service layer is managed according to the terms of these agreements. When different agreements are established on behalf of different consumer-provider relationships for a shared service environment, each service invocation may need to identify the agreement under which the invocation is to be managed. Details of service invocation (or resource usage) is specific to a service domain, and hence, outside the scope of this specification. Viable alternatives include explicit tagging of service messages with agreement identifiers to *claim* an agreement's service levels, or implicit classification of service messages based on domain-specific binding information in the agreements.

4 Agreement Structure

An agreement is conceptually composed of several distinct parts. We summarize the structure in Figure 2:

Figure 2: Structure of an agreement.

The section after the (optional) name is the context, which contains the meta-data for the entire agreement. It names the participants in the agreement, and the agreement's lifetime. The next section contains the terms that describe the agreement itself.

The XML representation of an agreement or an agreement creation offer has the following structure:

```
<wsag:Agreement AgreementId="xs:string">
   <wsag:Name>
       xs:string
   </wsag:Name> ?
   <wsag:AgreementContext>
      wsag:AgreementContextType
   </wsag:AgreementContext>
   <wsag:Terms>
       wsag:TermCompositorType
   </wsag:Terms>
</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 an agreement context and a collection of agreement terms.

/wsag:Agreement/@AgreementId

This is a mandatory identifier of this particular version of the agreement. It MUST be unique between Agreement Initiator and Agreement Responder. Through the effect of extended negotiation mechanisms not defined in this specification, different agreement documents MAY be regarded semantically as updated versions of an existing agreement relationship, potentially having the same Name and being exposed by the same Endpoint Reference. This id attribute helps agreement responder and consumer uniquely identify the version currently in force. If an agreement instance document is modified during the lifecycle of an Agreement resource, the identifier MUST also be replaced with a new, unique identifier.

/wsag:Agreement/wsag:Name

This is an OPTIONAL name that can be given to an agreement. The name of an agreement is independent of the name(s) of the template(s) it is based on (see wsag:Context/wsag:TemplateName below). The Name element is NOT a unique identifier. It MAY be used to provide a human-understandable name to an agreement in addition to the Endpoint Reference of the Agreement Resource that will be created in the protocol.

/wsag:Agreement/wsag: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, and the duration of the agreement.

/wsag:Terms

The terms of an agreement comprises one or more service definition terms, and zero or more guarantee terms grouped using logical grouping operators.

4.1 Agreement Context

An agreement is scoped by its associated context that SHOULD include parties to an agreement. Additionally, the agreement context contains various metadata about the agreement such as the duration of the agreement, and optionally, the template name from which the agreement is created.

```
<wsag:Context xs:anyAttribute>
  <wsag:AgreementInitiator>xs:anyType</wsag:AgreementInitiator> ?
  <wsag:AgreementResponder>xs:anyType</wsag:AgreementResponder> ?
  <wsag:ServiceProvider>wsag:AgreementRoleType</wsag:ServiceProvider>
  <wsag:ExpirationTime>xs:DateTime</wsag:ExpirationTime> ?
  <wsag:TemplateId>xs:string</wsag:TemplateId> ?
  <wsag:TemplateName>xs:string</wsag:TemplateName> ?
  <xs:any/> *
</wsag:Context>
```

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

/wsag:Context

This is the outermost tag which encapsulates the entire agreement context

/wsag:Context/wsag:AgreementInitiator

This optional element identifies the initiator of the agreement creation request. It MAY be a URI **[URI]** or a wsa:EndpointReference from WS-Addressing or MAY identify the initiator by a more abstract type of naming, e.g. by security identity of the owner or operator.

/wsag:Context/wsag:AgreementResponder

This optional element identifies the agreement responder, i.e. the entity that responds to the agreement creation request. It MAY be a URI or a wsa:EndpointReference from WS-Addressing or MAY instead identify the provider by a more abstract type of naming, e.g. by security identity of the owner or operator.

/wsag:Context/wsag:ServiceProvider

This element identifies the service provider and is either *AgreementInitiator* or *AgreementResponder*. The default is *AgreementResponder*.

/wsag:Context/wsag:ExpirationTime

This optional element specifies the time at which this agreement is no longer valid. Agreement initiators MAY use this mechanism to specify an Agreement service lifetime. Extended negotiation languages MAY define other mechanisms to negotiate lifetime integrated with other negotiation terms. The resulting negotiated lifetime MAY be exposed as wsag:ExpirationTime. One should note that ExpirationTime is included in the agreement context because it refers to the whole of the agreement.

/wsag:Context/wsag:TemplateID

This OPTIONAL element refers to the specific version of the template from which this offer or agreement is created. If a template was used to create an offer, the TemplateId in the Context MUST be set.

/wsag:Context/wsag:TemplateName

This OPTIONAL element specifies the name of the template from which this agreement is created. The reference to template is useful both for future modification of the agreement as well as provisioning of the service environment by the service provider. The template name MUST be included in an offer if the offer is based on a template (if no template is published by the agreement responder, this element MUST NOT be present in offers). A responder MAY check for this when doing an offer/template compliance check.

/wsag:Context/{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.

/wsag:Context/@{anyAttribute}

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:Context element of type wsag:AgreementContextType MAY be used in an agreement to define an agreement context. Alternatively, the agreement context MAY be specialized, through derivation of the wsag:AgreementContextType Schema type in order to define other attributes of the parties or services engaged in an agreement.

4.2 Agreement Terms

The main body of an agreement offer, and the consequent agreement, consists of terms. The terms of an agreement are wrapped by a wsag:Terms term compositor.

4.2.1 Term Types

A term expresses the defined consensus or obligations of a party. Terms are typed. Each term in an agreement has a type that is a subtype of the *abstract* wsag:TermType:

```
<wsag:Term Name="xs:string"?/>
```

This specification defines two types of terms: service terms and guarantee terms.

- The *service terms* provide information needed to instantiate or otherwise identify a service to which this agreement pertains and to which guarantee terms can apply. These are further refined as *service description*, *service reference* and *service property* terms.
- 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.

Additional term types – subtypes of wsag:TermType – MAY be defined for specific domains or types of obligations. To introduce additional term types, the abstract type wsag:TermType as defined in the AgreementTypes XML Schema MUST be extended.

4.2.2 Term Compositor Structure

Within the wsag:Terms compositor, special compositor elements can be used as logical AND/OR/XOR operators to combine terms. This enables the specification of alternative branches with potentially complex nesting within the terms of agreement.

The terms consist of one or more service terms and zero or more guarantee terms grouped using the logical grouping compositors.

Term compositors are structural elements of an agreement offer and the agreement. Choices expressed using compositors MUST be exercised by the service provider to satisfy the described requirements through some concrete delivery of service.

The recursive structure of a term compositor, of type wsag:TermCompositorType, is as follows:

```
<wsag:Terms>
    <wsag:All>
        {
            <wsag:All>
                wsag:TermCompositorType
            </wsag:All>       |
            <wsag:OneOrMore>
                wsag:TermCompositorType
            </wsag:OneOrMore> |
            <wsag:ExactlyOne>
                wsag:TermCompositorType
```

```
            </wsag:ExactlyOne> |
            <wsag:ServiceDescriptionTerm>
                wsag:ServiceDescriptionTermType
            </wsag:ServiceDescriptionTerm> |
            <wsag:ServiceReference>
                wsag:ServiceReferenceType
            </wsag:ServiceReference> |
            <wsag:ServiceProperties>
                wsag:ServicePropertiesType
            </wsag:ServiceProperties> |
            <wsag:GuaranteeTerm>
                wsag:GuaranteeTermType
            </wsag:GuaranteeTerm>
        } +
    </wsag:All>
</wsag:Terms>
```

The contents of a term compositor are described as follows:

/wsag:Terms/wsag:All (or wsag:OneOrMore, or wsag:ExactlyOne)

This is a logical AND (or OR, or XOR) operator of type wsag:TermCompositorType which is used to logically group terms and/or other compositors underneath it. This provides a recursive structure to the logical composition of terms.

/wsag:Terms/wsag:ServiceDescriptionTerm

One or more Service Description Terms, and/or Service References and/or Service Properties specify different aspects of a service. A Service Description Term provides an inline full or partial functional description of a new service, i.e. the information necessary to provide the service to the consumer.

/wsag:Terms/wsag:ServiceReference

These terms are OPTIONAL. A service reference contains a domain-specific reference to an existing service.

/wsag:Terms/wsag:ServiceProperties

These terms are OPTIONAL. Service properties specify domain-specific aspects of a service that can be used to express the non-functional requirements (guarantees) of the service.

/wsag:Terms/wsag:GuaranteeTerm

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

4.2.3 Service Description Terms

Service Description Terms (SDTs) are a fundamental component of an agreement: the agreement is about the service(s) — existing or not — described by the SDTs. The provisioning of this service may be conditional to specific run-time constraints, and additional service level objectives on how the service is performed may be imposed by the service guarantee; service description terms define the functionality that will be delivered under an agreement. The service description content itself is dependent on the particular domain. An SDT consists of three parts,

- The name of the SDT.
- The name of the service being described partially or fully by the domain-specific part of this service description term. This allows for semantic grouping of service description terms that may not be structurally grouped together in the agreement.
- A domain-specific description of the offered or required functionality. This element MAY completely describe the service it is about, or it MAY do so only partially.

An Agreement MAY contain any number of SDTs, as an agreement can refer to multiple components of functionality within one service, and can manage several services.

4.2.3.1 Service Description Term Structure

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

```
<wsag:ServiceDescriptionTerm
   wsag:Name="xs:string" wsag:ServiceName="xs:string">
   <xs:any> … </xs:any>
</wsag:ServiceDescriptionTerm>
```

The following describes the elements of the schema above:

/wsag:ServiceDescriptionTerm

ServiceDescriptionTerm encloses a description of a service or part of a service.

/wsag:ServiceDescriptionTerm/@wsag:Name

The MANDATORY name attribute (of type xs:string) represents the name given to a term. Since an Agreement MAY encompass multiple ServiceDescriptionTerms related to the same service each term SHOULD be given a unique name to make structural referencing of service description terms (for instance via XPATH) more convenient (see guarantee term section).

/wsag:ServiceDescriptionTerm/@wsag:ServiceName

This MANDATORY attribute identifies a service across multiple service description terms. The service description term is defined as "being about" the service identified by the wsag:ServiceName attribute. This identifier is scoped within the agreement i.e. it is not meant to identify the service outside of the agreement.

There are two scenarios for which multiple service terms may be used to specify a single service, i.e., the same ServiceName is associated with multiple service terms. First, an agreement may define a packaged service where multiple Service Description Terms may specify different service components. Alternatively, different Service Description Terms may describe different facets of a service, e.g., interface using WSDL, and additional service properties and associated metrics using ServiceProperties.

/wsag:ServiceDescriptionTerm/{xs:any}

This element is a placeholder for a partial or full description of the domain-specific service this service description term is about.

- This element is expressed using a domain-specific language that MAY be independent of WS-Agreement. Service description languages from

different domains or specifications MAY be embedded inside distinct service description terms.

- This element MUST be defined as a global element in the XML schema where it comes from. WS-Agreement does not mandate any restriction on the name or type (which can be simple or complex) of this element.
- This element MAY refer to one or more aspects of functionality of the described service, as granularity of that functionality is a domain-specific concern.

Example: the description of a computational job to execute.

4.2.4 Service Reference

A Service Reference points to a service, e.g., by providing an Endpoint Reference. Both parties understand the semantics of the service that is referred to or know how to query the service about its properties. The following definition describes the simple generic content of this type:

```
<wsag:ServiceReference
   wsag:Name="xs:string" wsag:ServiceName="xs:string">
   <xs:any> … </xs:any>
</wsag:ServiceReference>
```

The following describes the elements of the schema above:

/wsag:ServiceReference/@wsag:Name

This is the name given to this set of service references.

/wsag:ServiceReference/@wsag:ServiceName

This attribute identifies a service across multiple service description terms. The purpose of this attribute has been described previously.

/wsag:ServiceReference/{xs:any}

This element is a domain-specific representation of a reference to a service.

Examples:

- An EPR in an agreement on the performance of an existing Web service
- Metadata identifying a class of packet headers in an agreement on network Quality of Service).

4.2.5 Service Properties

ServiceProperties are used to define measurable and exposed properties associated with a service, such as response time and throughput. The properties are used in expressing service level objectives. The following definition describes the simple generic content of this type:

```
<wsag:ServiceProperties
   wsag:Name="xs:string" wsag:ServiceName="xs:string">
   <wsag:VariableSet>wsag:VariableSetType</wsag:VariableSet>
</wsag:ServiceProperties>
```

The following describes the elements of the schema above:

/wsag:ServiceProperties/@wsag:Name

This is the name given to this set of service properties.

/wsag:ServiceProperties/@wsag:ServiceName

This attribute identifies a service across multiple service description terms. The purpose of this attribute has been described previously.

/wsag:ServiceProperties/wsag:VariableSet

This element is a variable set (see definition below).

4.2.5.1 Variable Set

Guarantees contain logical expressions that refer to aspects of the service(s) subject to the guarantee. For instance, metrics for availability and response time must refer to named concepts (availability, response time) and must be declared as named variables that can be used in assertions. The semantics of those variables must be defined to interpret the condition expression. Each individual variable has the following form:

```
<wsag:Variable wsag:Name="xs:string" wsag:Metric="xs:URI">
    <wsag:Location>xs:anyType</wsag:Location>
</wsag:Variable>
```

/wsag:Variable/wsag:Location

The value of this element is a structural reference to a field of arbitrary granularity in the service terms — including fields within the domain-specific service descriptions.

- This reference gives scope to the concept represented by the variable, i.e. the concept applies at the nesting level of the structural item that is referred.
- This reference MAY be an XPATH expression for instance to use with domain-specific service description languages that are based on XML. XPATH references are relative to the AgreementOffer or Template elements of the document.

/wsag:Variable/@wsag:Name

This element, of type xs:*string*, is the name of the variable and allows the concept represented by this variable to be used in assertions. The name of each variable MUST be unique within the variable set.

/wsag:Variable/@wsag:Metric

This element, of type xs:URI, is an identification of a domain-specific metric. This element is optional and intended for cases where the structural reference of the variable does not sufficiently explain the semantics and typing of a variable. The domain specification where the metric is defined MUST define a namespace and a local name for the metric, as well as its type in logical expressions.
Note: If an XML particle definition exists for the metric, and when a fixed value makes sense for the concept, a wsag:Guarantee is not necessary and the XML particle MAY instead be used inside a wsag:ServiceDescriptionTerm element in order to specify a fixed value.

Examples:

```
<wsag:Variable name="CPUcount" metric="job:numberOfCPUs">
    <wsag:Location>
         //JobDescription/Resources/IndividualCPUCount/Exact
    </wsag:Location>
</wsag:Variable>
```

In this example, we assume a computational job is specified in an agreement offer (or agreement template, or agreement). A variable "CPUCount" refers to the concept of "number of CPUs to be used for the job at execution time", represented as a typed, globally-defined Schema particle "numberOfCPUs" in the namespace assigned to the prefix "job" (domain of computational jobs). "CPUCount" can be used in assertions that express limits, ranges or more complex relationships. Its scope of application is the "job:executable" unique domain-specific term so as to distinguish it from the overall job specification, which may includes other directives such as file transfers.

```
<wsag:Variable wsag:Name="bandwidth"
wsag:Metric="job:networkBandwidth">
    <wsag:Location>
         //JobDescription/Resources/IndividualNetworkBandwidth/Exact
    </wsag:Location>
</wsag:Variable>
```

In this example, the variable "bandwidth" could be used in the qualifying condition of the guarantee term to express a precondition on the file transfer it refers to.

Variables are grouped into a set:

```
<wsag:Variables>
   <wsag:Variable> … </wsag:Variable> *
</wsag:Variables>
```

/wsag:Variables

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

/wsag:VariableSet/wsag:Variable

Variables are specified above.

4.2.6 Guarantee Terms

One motivation for creating a service agreement between a service provider and a service consumer is to provide assurance to the service consumer on the service quality and/or resource availability offered by the service provider. Guarantee terms define this assurance on service quality, associated with the service described by the service definition terms. In the job submission example, an agreement may provide assurance on the bounds (e.g., minimum) on the availability of resources such as memory, type of central processing unit (CPU), storage and/or job execution

beginning or completion time. These bounds are referred to as *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 banking 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 service provider. Another example of business value is the importance of this assurance to the service consumer and/or to the service provider.

An agreement MAY also require a service consumer to give guarantees if the provider's service depends on it. For example, a service consumer of a compute job agreement may be required to provide a stage-in file in time such that the service provider can timely provide the results. To enable consumer-side guarantees, guarantee terms annotate the party that is obligated.

An agreement contains zero or more Guarantee terms, where each Guarantee Term element consists of the following parts:

- Obligated: The obligated party.
- Service Scope: the list of services this guarantee applies to.
- Qualifying Condition: an optional condition that must be met (when specified) for a guarantee to be enforced.
- Service Leve lObjective: an assertion expressed over service descriptions.
- Business Value List: one or more business values associated with this objective.

Note that a single Service Level Objective MAY be a set 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 Guarantee Term. Similarly, a Qualifying Condition MAY be a complex condition if multiple qualifying conditions need to be met for a guarantee to be honored.

4.2.6.1 Guarantee Term Structure

A *Guarantee Term* has the following form:

```
<wsag:GuaranteeTerm Name="xs:string" Obligated="wsag:ServiceRoleType">
    <wsag:ServiceScope ServiceName="xs:string">
        xs:any
    </wsag:ServiceScope>*
    <wsag:QualifyingCondition>…</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:Name

The MANDATORY name attribute (of type xs:string) represents the name given to a guarantee. Since an Agreement MAY encompass multiple GuaranteeTerms each term SHOULD be given a unique name.

/wsag:GuaranteeTerm/@wsag:Obligated

This attribute defines, which party enters the obligation to the guarantee term. The wsag:ServiceRoleType can be either *ServiceConsumer* or *ServiceProvider*.

/wsag:GuaranteeTerm/wsag:ServiceScope

A guarantee term can have one or more service scopes. A service scope describes to what service element specifically a guarantee term applies. It contains a ServiceName attribute and any other XML structure describing a sub-structure of a service to which the scope applies. For example, a performance guarantee might only apply to one operation of a Web service at a particular end point.

If a guarantee term applies to multiple services, a set of service scopes MUST be defined. There are two scenarios under which a single guarantee term may refer to multiple services. First, a single SLO as an expression may reference properties of multiple services, e.g., to define overall average response time or total MIPs, etc. Alternatively, the same property may be associated with multiple services and hence, the SLO must hold for each service.

/wsag:GuaranteeTerm/wsag:ServiceScope/@ServiceName

The name of a service to which the guarantee term refers. A guarantee term service scope applies to exactly one service.

/wsag:GuaranteeTerm/wsag:QualifyingCondition

This element MAY appear in order to express a precondition under which a guarantee holds.

/wsag:GuaranteeTerm/wsag:ServiceLevelObjective

This element, of type as defined in section 4.2.6.3, 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 Guarantee Term follows in the subsections.

4.2.6.2 Qualifying Condition

The optional *Qualifying Condition* in a guarantee term, when present, is expressed as an assertion over service attributes and/or external factors such as date, time, and the service request rate by the client. The type for this element is xs:anyType as a completely open content that can be extended with assertion languages which MAY be designed independently of the WS-Agreement specification but which MUST address the requirements of the particular domain of application of the agreement.

4.2.6.3 Service Level Objective

The *Service Level Objective* element in a guarantee term is also expressed as an assertion over service attributes and/or external factors such as date, time. However, most often a *Service Level Objective* is expressed as a target for a *Key Performance Indicator (KPI)* such as average response time, completion time, availability, etc. Hence, the core specification provides a simple expression structure for specifying a target for any domain specific KPIs defined outside this specification.

A *Service Leve lObjective* has the following form, expressed either using a *KPITarget* or as a customized expression of service level.

```
<wsag:ServiceLevelObjective>
   <wsag:KPITarget> </wsag:KPITarget> |
   <wsag:CustomServiceLevel> … </wsag:CustomServiceLevel>
 </wsag:ServiceLevelObjective>
```

/wsag:ServiceLevelObjective

This element specifies a service level objective in a guarantee term, and contains an element either of type wsag:KPITarget or wsag:CustomServiceLevel.

/wsag:ServiceLevelObjective/wsag:KPITarget

This element defines service level objective as an expression of a target of a key performance indicator associated with the service.

/wsag:ServiceLevelObjective/wsag:CustomServiceLevel

This element is of type xs:anyType and can be customized by using a domain specific expression or assertion language which MAY be designed independently of the WS-Agreement specification.

4.2.6.3.1 KPI Target

A *KPI Target* expresses a service level objective by specifying a target for a key performance indicator (KPI) such as average response time, availability, etc. associated with a service. The definition of a KPI is outside the scope of the current specification. Such definitions may include information such as unit of measurement, in addition to the meaning of this KPI.

A *KPI Target* is of the form

```
<wsag:KPITarget>
    <wsag:KPIName>xs:string</wsag:KPIName>
    <wsag:Target>xs:any</wsag:Target>
</wsag:KPITarget>
```

/wsag:KPITarget

This element defines service level objective as an expression of a target of a key performance indicator associated with the service.

/wsag:KPITarget/wsag:KPIName

This name of a key performance indicator associated with the service.

/wsag:KPITarget/wsag:Target

This element defines the target value for a KPI.

4.2.6.4 Business Value List

Associated with each wsag:ServiceLevelObjective is a wsag:BusinessValueList that contains multiple business values, each expressing a different value aspect of the objective. Depending on the scenario and value type, each value represents an assertion by one or both parties. For example, in an agreement representing resource reservation for job submission, the submitter may express "importance" of meeting an objective, while a provider may specify "confidence" or likelihood of meeting that objective. In an untrusted or cross-organizational scenario, the business value may be expressed as a joint assertion using "penalty" or "reward" value type. A penalty expresses indirectly both the importance to a consumer, where a higher penalty is more likely to induce provider to meet this objective, and also specifies compensation to be assessed for failing to meet the objective.

"Preference" is used to describe a list of fine-granularity business values for different alternatives, where satisfying each alternative results in a different business value. For example, a job submission may specify many resource configuration alternatives, each resulting in a different utility. Depending on the available resources, other competing jobs and the utility to be achieved, the resource provider allocates appropriate resources in order to maximize the overall utility.

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 service provider from fixed allocation of resources. A service 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> xs:integer </wsag:Importance>?
  <wsag:Penalty> </wsag:Penalty>*
  <wsag:Reward> </wsag:Reward>*
  <wsag:Preference> </wsag:Preference>?
  <wsag:CustomBusinessValue> … </wsag:CustomBusinessValue>*
</wsag:BusinessValueList>
```

/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 the penalty to be assessed for not meeting an objective. If multiple Penalty statements are present, for example penalty statements expressed per week and per month basis for an availability objective, the longest assessment duration resulting in highest assessment value will be applied. In the above example, multiple weekly

violations within a month may result in a higher penalty amount as assessed by the monthly penalty statement rather than the cumulative weekly assessments.

/wsag:BusinessValueList/wsag:Reward

This element (defined below) when present expresses reward to be assessed for meeting an objective. If multiple Reward statements are present, they are applied alternatively, depending on the longest assessment interval applicable.

/wsag:BusinessValueList/wsag:Preference

This element specifies a list of fine-granularity business values for different alternatives, where each alternative refers to a Service Description Term and its associated utility.

/wsag:BusinessValueList/wsag:CustomBusinessValue

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

4.2.6.4.1 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 service 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 thought of as a measure of importance with a default measurement unit.

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

4.2.6.4.2 Penalty and Rewards

In business Service Level Agreements (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>xs:duration</wsag:TimeInterval> |
        <wsag:Count>xs:positiveInteger</wsag:Count>
    </wsag:AssesmentInterval>
    <wsag:ValueUnit>xs:string</wsag:ValueUnit>?
    <wsag:ValueExpr>xs: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 duration. For example, a weekly or monthly interval for defining the assessment.

/wsag:Penalty/wsag:AssesmentInterval/wsag:Count

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

/wsag:Penalty/wsag:ValueUnit

This element defines the unit for assessing penalty, such as USD. This is an optional element since in some cases a default unit MAY be assumed.

/wsag:Penalty/wsag:ValueExpr

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

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

4.2.6.4.3 Preference

"Preference" is used to describe a list of fine-granularity business values for different alternatives, where satisfying each alternative results in a different business value. For example, a job submission may specify many resource configuration alternatives, each resulting in a different utility. Depending on the available resources, other competing jobs and the utility to be achieved, the resource provider allocates appropriate resources in order to maximize the overall utility.

```
<wsag:Preference>
    <wsag:ServiceTermReference>xs:string </wsag:ServiceTermReference>*
    <wsag:Utility>xs:float</wsag:Utility>*
</wsag:Preference>
```

/wsag:Preference

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

/wsag:Preference/wsag:ServiceTermReference

This element can appear multiple times. Each Service Term Reference refers to a service term and represents an alternative way of achieving the associated service level objective. The corresponding, utility (specified below) specifies the utility gained by achieving this objective.

/wsag:Preference/wsag:Utility

This element can appear multiple times, one corresponding to each Service Term Reference.

5 Agreement Template and Creation Constraints

To create an agreement, a client makes an offer to an agreement factory. An agreement creation offer has the same structure as an agreement. The agreement factory advertises the types of offers it is willing to accept by means of agreement *templates*.

An agreement template is composed of three distinct parts. We summarize the structure in Figure 3:

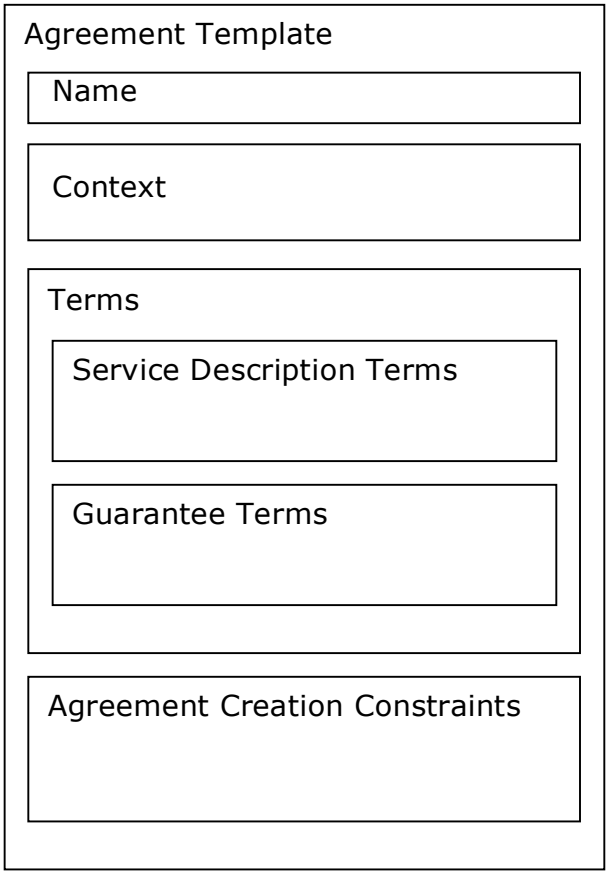


Figure 3: Structure of an agreement template.

The structure of an agreement template is the same as that of an agreement, but an Agreement template MAY also contain a creation constraint section, i.e. a section with constraints on possible values of terms for creating an agreement. The constraints make it possible to specify the valid ranges or distinct values that the terms may take. The constraints refer back to individual terms they apply to using XPATH.

The contents of an agreement template are of the form:

```
<wsag:Template TemplateId=”xs:string”>
   <wsag:Name>
       xs:string
   </wsag:Name> ?
```

```
   <wsag:AgreementContext>
      wsag:AgreementContextType
   </wsag:AgreementContext>
   <wsag:Terms>
       wsag:TermCompositorType
   </wsag:Terms>
   <wsag:CreationConstraints>
       …
   </wsag:CreationConstraints> ?
</wsag:Template>
```

The following describes the contents of the agreement template:

/wsag:Template

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

/wsag:Template/@TemplateId

The MANDATORY TemplateId is a unique identifier of the agreement responder for a specific version of the template. When updating a template having the same Name attribute a new, unique TemplateId MUST be provided.

/wsag:Template/wsag:Name

This is an OPTIONAL name that can be given to an agreement matching this template.

/wsag:Template/wsag:Context

This is a REQUIRED element in the agreement template. This is the template for the context of the agreements matching the containing agreement template.

/wsag:Template/wsag:Terms

This section specifies the possible terms in the agreements matching this template. The description of this section has been made previously in this document (see "Agreement Structure") and is not repeated here.

/wsag:Template/wsag:CreationConstraints

These are OPTIONAL elements that provide constraints on the values that the various terms may take in a concrete agreement.

The specification of a creation constraint section in a template does not state a promise that an agreement creation offer fulfilling the constraints will be accepted. Typically, an agreement responder MAY publish an agreement template containing a creation constraint section, outlining agreements it is generally willing to accept. Whether the agreement responder accepts a given offer might depend on its current resource situation.

5.1 Creation Constraints

The element Creation Constraints is of type wsag:ConstraintSectionType. It has the following form inside the template:

```
<wsag:Template>
```

```
      …
   <wsag:CreationConstraints>
      <wsag:Item>…</wsag:Item> *
      <wsag:Constraint>…</wsag:Constraint> *
   </wsag:CreationConstraints> ?
</wsag:Template>
```

/wsag:Template/wsag:CreationConstraints

This optional element of an Agreement, of type wsag:ConstraintSectionType, expresses the constraints for creating/negotiating an agreement. It contains any number of offer items and constraints in any order.

/wsag:Template/wsag:CreationConstraints/wsag:Item

This element specifies that a particular field of the agreement must be present with a value in the agreement offer, and which values are possible.

/wsag:Template/wsag:CreationConstraints/wsag:Constraint

A constraint, of type wsag:ConstraintType, defines any constraint involving the values of one or more terms.

The wsag:ConstraintSectionType MAY be used by other specifications in order to define constraints that must apply when creating or modifying agreements, for instance in agreement negotiations.

5.1.1 Offer Item

An offer item specifies the requirement for the presence in the agreement offer terms of a field and a value for that field. It contains a label, a pointer to the position of the field in the terms of the offer and also MAY contain a definition of its acceptable values in the form of a restriction of its value space. The restriction is taken from the XML Schema language and follows either the xs:simpleRestrictionModel in the case of simple offer items or the xs:typeDefParticle if the content to be filled in is a complex type in the sense of XML.

```
<wsag:Item Name="xs:string">
    <wsag:Location>
        xs:anyType
    </wsag:Location>
    <wsag:ItemConstraint>
        <xs:restriction>
            xs:simpleRestrictionModel
        <xs:restriction> ?
        <xs:group>xs:groupRef</xs:group> ?
        <xs:all>xs:all</xs:all> ?
        <xs:choice>xs:explicitGroup</xs:choice> ?
        <xs:sequence>xs:explicitGroup</xs:sequence>?
    </wsag:ItemConstraint>
    <xs:any>any#other</xs:any> *
```

```
</wsag:Item>
```

/wsag:Item

A simple restriction represents a simple value constraint on a term of an offer. Besides name and Location it may contain a definition or restriction of the value that an agreement initiator MAY fill in. In the case of a simple type offer item, the xs:restriction MAY be filled in. In the case of a complex offer item one of the elements xs:group, xs:all, xs:choice, or xs:sequence MAY be filled in. A fill in item is either of a simple type or of a complex type.

/wsag:Item/@Name

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

/wsag:Item/wsag:Location

The location is a structural reference, for instance an XPATH expression, which points to the location in the terms of the Agreement that can be changed and filled in. The value set at the addressed template location set at the location referred to is the default value of the item.

/wsag:Item/wsag:ItemConstraint

A constraint on the value that initiators can fill in at the point of the item's location.

/wsag:Item/wsag:ItemConstraint/xs:restriction

A restriction applies to the value that can be filled in by an agreement initiator at the specified location at agreement creation time. If all filled in values adhere to their respective restriction an agreement is compliant with its template. The restriction element, which is a reference to the group simpleRestrictionModel from the XML Schema namespace, 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 definition of the group. It is the responsibility of the author of the template to make sure that the restriction defined in the Item is a valid restriction of the type of the field that the item location attribute points to. Restrictions are not quality of service constraints, which are to be defined in guarantee terms.

/wsag:Item/wsag:ItemConstraint/xs:group

A group statement in an offer item refers to a group in another, domain-specific XML schema file that restricts the possible content of an offer item having a complex type. The xs:group element and all subsequent alternatives for constraining complex values are defined as xs:typeDefParticle in the XML Schema language. In the XML Schema language xs:group is defined as an xs:groupRef element.

/wsag:Item/wsag:ItemConstraint/xs:all

An all statement in an offer item restricts the possible content of an offer item having a complex type. The xs:all element as well as the above-mentioned xs:group element and the subsequent alternatives for constraining complex values are defined as xs:typeDefParticle in the XML Schema language. In the XML Schema language xs:all is defined as an xs:all element.

/wsag:Item/wsag:ItemConstraint/xs:choice

A choice statement in an offer item restricts the possible content of an offer item having a complex type. The xs:choice element as well as the above-mentioned xs:group and xs:all elements and the subsequent xs:sequence element for constraining complex values are defined as xs:typeDefParticle in the XML Schema language. In the XML Schema language xs:choice is defined as an xs:explicitGroup element.

/wsag:Item/wsag:ItemConstraint/xs:sequence

A sequence statement in an offer item restricts the possible content of an offer item having a complex type. The xs:sequence element as well as the above-mentioned xs:group, xs:all and xs:choice elements for constraining complex values are defined as xs:typeDefParticle in the XML Schema language. In the XML Schema language xs:sequence is defined as an xs:explicitGroup element.

/wsag:Item/{any}

Any other content MAY be added to an item and will be interpreted domain-specifically as a constraint to the offer item.

An item MAY have at most one of the restriction, group, all, choice or sequence elements.

5.1.2 Free–form Constraints

Free-form constraints make it possible to restrict the possible values of the term set of an offer beyond restrictions of individual terms. 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 allows a suitable existing one to be chosen. Hence, the Constraint is an empty top-level element that must be extended by a specific, suitable constraint language:

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

5.2 Relationship between the Agreement Template and the Final Agreement

As will be discussed in Section 6, Agreement templates provide hints for the Agreement Initiator to create the final agreement. Service Description Terms which appear in the Agreement Template would specify default values for a specific SDT. Likewise creation constraints provide a mechanism to provide a range of values that the Agreement Responder is willing to serve. Agreement Initiator MAY specify terms which are not present in the agreement templates (such as names of the job binaries etc.)

6 Compliance of Offers with Templates

The purpose of templates is to give guidance on what forms of offer an agreement responder wishes to receive. As such, offers SHOULD in general comply with one of the templates advertised by the responder. However, the responder MAY accept offers which do not match any template, and the responder also MAY reject offers that do match for other policy reasons. In this section we define the concept of agreement template compliance.

Definition: An agreement offer is *compliant* with a template advertised by an agreement responder if and only if each term of service described in the Terms section of the agreement offer complies with the term constraints expressed in the wsag:CreationConstraints section of the agreement template.

In addition, certain portions of the Context section of the offer have a required relation to corresponding portions of the Context in the template. These are:

- wsag:AgreementResponder: The AgreementResponder value provided in the offer MUST match the value, if any, specified in the template.
- wsag:TemplateId: The TemplateId in the offer must exactly match the name provided in the template document against which compliance is being checked. If the TemplateId is not provided, the provider MAY use any policy to determine compliance. These MAY include rejecting all, testing against all templates, or evaluating independently of the templates advertised.

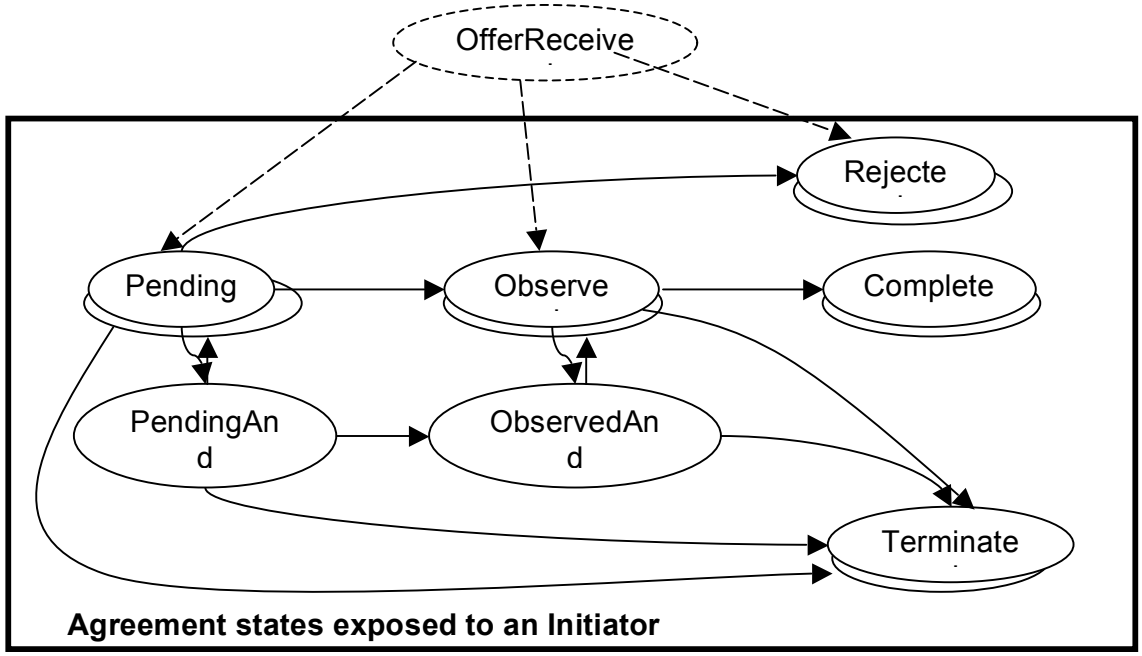
7 Runtime States

Agreements and Terms have a runtime state that can be monitored. The objective of term status monitoring is to observe agreement compliance at runtime. To interpret the state of a guarantee, the service term state must be known. If a service is not running, a guarantee term might not be determined. To interpret the state of a service term, the overall Agreement state must be known. If the Agreement is not accepted, the service and guarantee term states are not determined.

Verifying agreement and – particularly – terms states requires significant infrastructure and is dependent on the application environment and the domain. Hence, the verification of agreement and, term states is outside the scope of this specification.

7.1 Agreement States

The overall Agreement has a state derived from the Agreement protocol. The Agreement State observes the following state model.



Pending, **PendingAndTerminating**, **Observed**, **ObservedAndTerminating**, **Rejected**, **Complete** and **Terminated** are the normative primary states of an Agreement State. Each state can be extended with one or more sub-states in a specific usage domain. **OfferReceived** is an initial transition state (that is not

exposed to the initiator — represented by the dashed lines) to clarify that exposed initial states can be "Pending", "Observed" or "Rejected".

- **Pending** - The **Pending** state means that an Agreement offer has been made but it has been neither accepted nor rejected.
- **PendingAndTerminating** - The **PendingAndTerminating** state means that an Agreement offer has been made and it has not been accepted or rejected and furthermore a Terminate operation has been issued by the Agreement Initiator and is being processed. This state MAY follow **Pending.** This state MAY be followed by the **Pending** state in a case where a termination request is made but not accepted by the responder.
- **Observed** - The **Observed** state means that an Agreement offer has been made and accepted. This state MAY follow **Pending**.
- **ObservedAndTerminating** – The **ObservedAndTerminating** state means that that an Agreement offer has been made and accepted. Furthermore, a Terminate operation has been issued from the Agreement Initiator and is being processed by the Agreement Responder. This state MAY follow **Observed** or **PendingAndTerminating**. This state MAY be followed by the **Observed** state in a case where a termination request is made but not accepted by the responder.
- **Rejected** - The **Rejected** state means that an Agreement offer has been made and rejected. This state MAY follow **Pending**.
- **Complete** - The Complete state means that an Agreement offer has been received and accepted, and that all activities pertaining to the Agreement are finished This state MAY follow **Observed**.
- **Terminated** - The terminated state means that an Agreement offer has been terminated by the Agreement Initiator and that the obligation no longer exists. This state MAY follow **Pending**, **PendingAndTerminating**, **Observed** or **ObservedAndTerminating** when the termination decision is made. The fact that the Agreement is in this state MAY imply that a domain specific penalty is imposed.

The **Pending** and **Rejected** states indicate that the responder is not obligated in any way. The **Pending** state indicates that the initiator is obligated if and only if the responder accepts the offer. The **Observed** and **ObservedAndPending** states indicate that both parties are obligated with respect to the service and guarantee terms of the agreement. The **Complete** and **Terminated** states indicate that both parties MAY still have non-normative obligations related to business transactions e.g. accounting, billing and payment.

The accepted Agreement is **Complete** only when all service states are completed, and the accepted Agreement is otherwise **Observed**. The contract between an AgreementInitiator and an Agreement Responder is fulfilled when the Agreement state has transitioned to **Complete** or one of the terminating states.

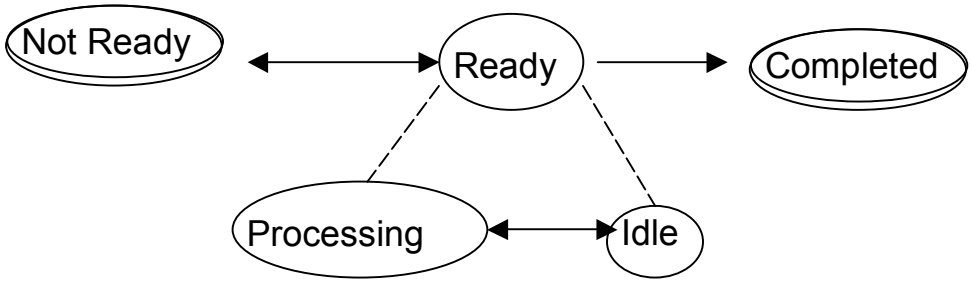
There may be domain dependent cases where an agreement completes normally while in **PendingAndTerminating** or **ObservedAndTerminating** states. These cases can be handled by allowing a domain-dependent sub-state of **Terminated** to indicate a normal completion prior to termination completion.

7.2 Service Runtime States

The property exposes a **service state** for each service description term that abstractly describes the state of a service, independent of its domain. The service states are only applicable when the agreement is in an Observed or

ObservedAndTerminating state. Each list element is a tuple (term ID, service term state).

The service term state observes the following state model:



Not Ready, **Ready** and **Completed** are the normative primary states of a service description term. Each state can be extended with one or more sub-states in a specific usage domain. **Processing** and **Idle** are two normative sub-states of the primary state **Ready.**

The semantics of the states is as follows:

- **Not Ready** – The service cannot be used yet.
- **Ready** – The service can now start to be used by a client or to be executed by the service provider.
- **Processing** – The service is ready and currently processing a request or is otherwise active.
- **Idle** – The service is ready, however currently not being used.
- **Completed** – The service cannot be used any more and any service provider activity, e.g. performing a job, is finished. This state does not express whether an execution of a job or service was successful.

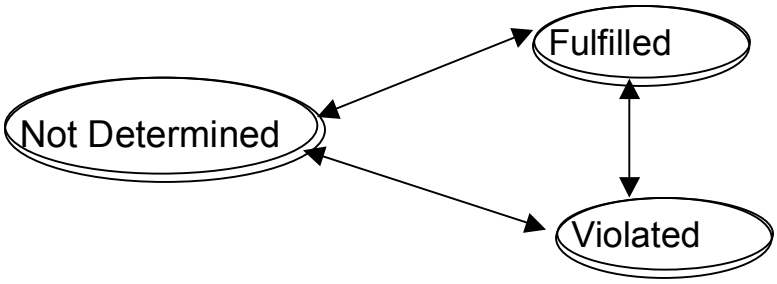
Not Ready is the initial state of a service description term while the service is being activated or provisioned. Once a service is ready, it may cycle through the periods of active use and idling, represented by the sub-states of **Processing** and **Idle,** respectively. Once a service is completed and can not be reused further, the service description term reaches the terminal state, marked **Completed**.

If a service is not ready or ready, the state of a guarantee relating to this service term is not determined. If the service description term is processing or completed, the guarantee term can expose the states fulfilled or violated.

7.3 Guarantee States

This property represents a state of fulfillment for each guarantee term of the agreement. Each list element is a tuple (term ID, guarantee term state).

The **guarantee states** follow a simple state model:



The semantics of the states are as follows:

- **Fulfilled** – Currently the guarantee is fulfilled.
- **Violated** – Currently the guarantee is violated.
- **NotDetermined** – No activity regarding this guarantee has happened yet or is currently happening that allows evaluating whether the guarantee is met.

NotDetermined is the initial state of a guarantee term, until a service is invoked or fulfilled and assessment is made. Depending on the assessment the terminal state can be either **Fulfilled** or **Violated.** For guarantee terms, that require recurring assessment, the term state after every assessment period may be in **Fulfilled**, **Violated** or **Not Determined** state. If there was no service activity in the preceding window, then the term state will be in **Not Determined** state**.**

8 Acceptance Model

The WS-Agreement protocol is based on a single round "offer, accept" message exchange. The basic synchronous wsag:CreateAgreement operation directly captures this exchange, but additional asynchronous patterns are supported as well. The semantics of this abstract exchange is that the initiator sends an *obligating* offer, with explicit terms of agreement, which the responder may accept or reject by unilateral decision.

The agreement relationship is in place as soon as the responder decides to accept, and the subsequent return messages inform the initiator of this decision. It is important to understand that the initiator may or may not be obliged to the terms of the agreement until this decision is made; furthermore, in practice the initiator may not be aware of the decision until some time after the decision is made. The obligations expressed in the agreement are not dependent on the initiator being informed of the decision, and the use of WS-Agreement to express real-time terms of service causes this hazard which the domain-specific parties must be capable of tolerating.

8.1 Forms of Offer

The initiator may send his obligating offer in the input message to either the wsag:CreateAgreement or wsag:CreatePendingAgreement operations, depending on availability of these operations at the responder's endpoint. Either form represents the same obligations towards the domain-specific service terms.

8.2 Forms of Acceptance

In response to a wsag:CreateAgreement offer, the responder accepts the offer by sending the Agreement EPR rather than a fault. The new Agreement MUST already be in an accepted state (Observed or Complete).

In response to a wsag:CreatePendingAgreement offer, the responder also sends an Agreement EPR but the new Agreement MAY be in any state by the time the Agreement EPR is received by the Agreement Initiator. The Agreement MUST remain in the Pending state until the responder decides whether to accept, and following a decision to accept, the Agreement MUST transition to an accepted state (Observed or Complete). If the initiator specified an AgreementAcceptance EPR with the offer, the responder also MUST invoke the wsag:Accept operation to indicate its decision. If the wsag:Accept operation returns a fault, the responder MUST transition to an accepted state. The initiator MAY still determine the responder's response to wsag:CreatePendingAgreement offer by other means such as querying the status of the resulting Agreement. To reduce the likelihood of spurious agreements being accepted, the responder MAY wish to validate the EPR prior to invoking the wsag:Accept operation.

8.3 Forms of Rejection

In response to a wsag:CreateAgreement offer, the responder rejects the offer by sending a fault response.

In response to a wsag:CreatePendingAgreement offer, the responder MAY reject by sending a fault response, and if it returns an Agreement EPR it MAY later indicate rejection by changing the Agreement state from Pending to Rejected. If the initiator specified an AgreementAcceptance EPR with the offer and the responder returns an Agreement EPR, the responder also MUST invoke the wsag:Reject operation to indicate his decision. If the wsag:Reject operation returns a fault, the responder MAY transition the agreement to a Rejected state.

8.4 Partial Ordering of Responses

In the case where the responder sends an Agreement EPR and must also invoke an operation on an initiator-supplied AgreementAcceptance service, the creation response and the Accept or Reject input may arrive in either order.

9 Port Types and Operations

In this section we detail the AgreementFactory, PendingAgreementFactory, Agreement and AgreementAcceptance port types. These port types can be used in various combinations to support a number of signaling scenarios:

1. Simple client-server. The AgreementFactory can be invoked by initiators to create a responder-side Agreement and to perform monitoring and management using only WS client mechanisms. Both port types would be implemented by the responder.
2. Symmetric. As with the simple client-server scenario, the initiator invokes the AgreementFactory to create a responder-side Agreement. As additional input to the createAgreement call, the initiator passes the address of an initiator-side Agreement representing the offer he is making. As a result, both parties have an Agreement resource representing the same accepted agreement relationship between them, to allow symmetric monitoring and management by the, respective, other party. Differences in current state of the agreement

resources of initiator and responder are interpreted and dealt with domain-specifically, e.g., by state replication or dispute handling.

3. Deferred polling. The PendingAgreementFactory can be invoked by initiators to create a responder-side Agreement on which the responder's decision to accept or reject will later be expressed. The initiator may check the status of the Agreement by any available ResourceProperty query mechanism.
4. Deferred call back. As with the deferred client-server scenario, the initator invokes the PendingAgreementFactory. As additional input to the createPendingAgreement call, the initiator passes the address of an AgreementAcceptance service to which the responder will send an explicit Accept or Reject message to communicate his decision (in addition to updating his Agreement status as in the basic deferred case). This allows for true asynchronous messaging over any binding technology where the initiator can present addressable endpoints.
5. Deferred and symmetric. With either polling or callback-based deferred signaling, the initiator may also pass the address of an initiator-side Agreement to enable fully symmetric monitoring and management as in symmetric case (2) in addition to decoupling the responder's acceptance decision from the creation step.

Per the reuse principles of the WS-Resource Framework **[WS-Resource]** 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 hersigners can reuse the messages and resource properties defined in the AgreementFactory and Agreement port types and compose them in their own specialized, domain-specific port types. They can also compose agreement state-related resource properties as defined in the AgreementState placeholder port type into their own Agreement port type.

Every port type exposes a wsag:GetResourceProperty operation based on the operation of the same NCName as defined in **[WS-ResourceProperties]**. This operation enables the port types to expose read-only resource properties. Its definition is identical to the one in **[WS-ResourceProperties]** and has not been repeated here.

The wsrl:Destroy operation as defined in the **[WS-ResourceLifetime]** MAY be used by the Initiator to explicitly destroy no longer used resources.

The wsrf-rp:GetMultipleResourceProperties operation from [WS-ResourceProperties] MAY be composed as well in order to enable retrieval of several resource properties in one request/response message exchange, for instance in order to obtain a complete agreement in one round-trip invocation. Similarly, other operations from **[WS-ResourceProperties]** (and other specifications) such as wsrf-rp:QueryResourceProperties MAY be composed into domain-specific agreement and agreement factory port types.

Full WSDL definition of the port types can be found in Appendix-1.

9.1 Port Type wsag:AgreementFactory

9.1.1 Operation wsag:CreateAgreement

The wsag:CreateAgreement operation is used to generate an Agreement.

9.1.1.1 Input

The form of the wsag:CreateAgreement input message is:

```
<wsag:CreateAgreementInput>
    <wsag:InitiatorAgreementEPR>
        <wsa:EndpointReference>
            wsa:EndpointReferenceType
        </wsa:EndpointReference>
    </wsag:InitiatorAgreementEPR> ?
    <wsag:AgreementOffer>
        ...
    </wsag:AgreementOffer>
    <wsag:NoncriticalExtension>
        <xs:any> ... </xs:any>
    </wsag:NoncriticalExtension> *
    <xs:any> ... </xs:any> *
</wsag:CreateAgreementInput>
```

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

/wsag:CreateAgreementInput/wsag:initiatorAgreementEPR

This optional element is an endpoint reference (EPR) providing a contact point EPR1 where the invoked party can send messages pertaining to this Agreement, in order to view a symmetric representation of the agreement as viewed by the initiator. The responder MUST NOT invoke operations on this EPR after returning a fault on this operation. The Agreement addressed by this EPR SHOULD be in the Pending state until this operation returns successfully.

/wsag:CreateAgreementInput/wsag:AgreementOffer

The offered agreement made by the sending party. It MAY satisfy the agreement creation constraints expressed in one or more of the templates advertised by the AgreementFactory.

/wsag:CreateAgreementInput/wsag:NoncriticalExtension

These optional elements MAY carry *non-critical* extensions which control augmented creation mechanisms. The responder MAY ignore non-critical extensions and behave as if they are not present. A responder SHOULD obey non-critical extensions if it is able and willing. The meaning of extensions and how to obey them is domain-specific and MUST be understood from the extension content itself.

/wsag:CreateAgreementInput/xs:any##other

Additional elements MAY be used to carry *critical* extensions which control augmented creation mechanisms. All unwrapped extensions are considered mandatory or critical, 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.

9.1.1.2 Result

The successful result of wsag:CreateAgreement is the EPR of a newly created Agreement in the Observed state:

```
<wsag:CreateAgreementResponse>
    <wsag:CreatedAgreementEPR>
```

```
        wsa:EndpointReferenceType
    </wsag:CreatedAgreementEPR>
    <xs:any> ... </xs:any> *
</wsag:CreateAgreementResponse>
```

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

/wsag:CreateAgreementResponse/wsag:CreatedAgreementEPR

This is the EPR to a newly created Agreement bearing the same terms as the input agreement offer. This element MUST appear, and the Agreement MUST be in the Observed or Complete states prior to the return of this response. Such an Agreement MUST NOT transition to the Rejected state.

/wsag:CreateAgreementResponse/xs:any##other

These optional items MAY be used to carry extended content that is under the control of corresponding extensions in the input message.

9.1.1.3 Faults

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

9.1.2 Resource Property wsag:Template

The templates resource property represents a sequence of 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 prototype agreement along with creation constraints, as defined in section 5.

9.2 Port Type wsag:PendingAgreementFactory

9.2.1 Operation wsag:CreatePendingAgreement

The wsag:CreatePendingAgreement operation is used to generate an Agreement when the decision process may be delayed too long to use (in practice) the wsag:CreateAgreement operation.

9.2.1.1 Input

The form of the wsag:CreatePendingAgreement input message is:

```
<wsag:CreatePendingAgreementInput>
    <wsag:AgreementAcceptanceEPR>
        <wsa:EndpointReference>
            wsa:EndpointReferenceType
        </wsa:EndpointReference>
    </wsag:AgreementAcceptanceEPR> ?
    <wsag:InitiatorAgreementEPR>
        <wsa:EndpointReference>
            wsa:EndpointReferenceType
        </wsa:EndpointReference>
    </wsag:InitiatorAgreementEPR> ?
    <wsag:AgreementOffer>
        ...
    </wsag:AgreementOffer>
```

```
    <wsag:NoncriticalExtension/> *
    <xs:any> ... </xs:any> *
</wsag:CreatePendingAgreementInput>
```

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

/wsag:CreatePendingAgreementInput/wsag:AgreementAcceptanceEPR

This optional element is an endpoint reference (EPR) representing a contact point where the responder MUST invoke a subsequent wsag:Accept or wsag:Reject operation to communicate his decision regarding the offer. This invocation MAY precede or follow the response message to wsag:CreatePendingAgreement. If this element is omitted, the client SHOULD determine the decision result by other means such as querying the status of the resulting Agreement.

/wsag:CreatePendingAgreementInput/wsag:InitiatorAgreementEPR

This optional element has the same semantics as in wsag:CreateAgreementInput.

/wsag:CreatePendingAgreementInput/wsag:AgreementOffer

This element has the same semantics as in wsag:CreateAgreementInput.

/wsag:CreatePendingAgreementInput/wsag:NoncriticalExtension

These optional elements have the same semantics as in wsag:CreateAgreementInput.

/wsag:CreatePendingAgreementInput/xs:any

These optional elements have the same semantics as in wsag:CreateAgreementInput.

The two optional EPRs, if specified, MAY address the same service which implements the required features of both the wsag:AgreementAcceptance and wsag:Agreement portTypes, or they MAY address separate services which are correlated by implementation-specific mechanisms. The service(s) addressed by AgreementAcceptanceEPR and InitiatorAgreementEPR MUST implement the required features of wsag:AgreementAcceptance and wsag:Agreement, respectively.

9.2.1.2 Result

The successful result of the wsag:CreatePendingAgreement operation is the EPR of a newly created Agreement:

```
<wsag:CreatePendingAgreementResponse>
    <wsag:CreatedAgreementEPR>
        wsa:EndpointReferenceType
    </wsag:CreatedAgreementEPR>
    <xs:any> ... </xs:any> *
</wsag:CreatePendingAgreementResponse>
```

The contents of the response message are understood identically to the response from wsag:CreateAgreement, except that the resulting Agreement MAY be in the Pending, Observed, Complete, or Rejected states. The responder MAY subsequently reject the offer, resulting in a delayed transition from the Pending to Rejected state.

9.2.1.3 Faults

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

9.2.2 Resource Property wsag:Template

The templates resource property represents a sequence of 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 prototype agreement along with creation constraints constraints, as defined in section 5.

9.3 Port Type wsag:AgreementAcceptance

An AgreementAcceptance resource is associated with an offer to allow a deferred decision to be communicated as to whether the offer is accepted or rejected. The AgreementAcceptance shares the same overall state value with Agreement so that they can be composed for fully symmetric scenarios. One should note that this is a port type on the agreement initiator side; **not** on the agreement responder side.

9.3.1 Operation wsag:Accept

An AgreementAcceptance resource that is in the Pending state MAY be accepted to transition to the Observed state.

9.3.1.1 Input

The form of the wsag:Accept input message is:

```
<wsag:AcceptInput>
    <wsag:NoncriticalExtension/> *
    <xs:any> ... </xs:any> *
</wsag:AcceptInput>
```

The input is usually empty, but the wsag:Accept operation follows the same extensibility pattern as is described in wsag:CreateAgreement.

9.3.1.2 Result

The successful result of wsag:Accept indicates that the associated Agreement is now understood to be Observed.

```
<wsag:AcceptResponse>
    <xs:any> … </xs:any> *
</wsag:AcceptResponce>
```

The result is usually empty, but the wsag:Accept operation follows the same extensibility pattern as is described in wsag:CreateAgreement.

9.3.1.3 Faults

A fault indicates that acceptance of this AgreementAcceptance resource is not possible and also MAY include the reason.

9.3.2 Operation wsag:Reject

An AgreementAcceptance resource that is in the Pending state MAY be rejected to transition to the Rejected state.

9.3.2.1 Input

The form of the wsag:Reject input message is:

```
<wsag:RejectInput>
    <wsag:NoncriticalExtension/> *
```

```
    <xs:any> ... </xs:any> *
</wsag:RejectInput>
```

The input is usually empty, but the wsag:Reject operation follows the same extensibility pattern as is described in wsag:CreateAgreement.

9.3.2.2 Result

The successful result of wsag:Reject indicates that the associated Agreement is now understood to be Rejected.

```
<wsag:RejectResponse>
    <xs:any> ... </xs:any> *
</wsag:RejectResponce>
```

The result is usually empty, but the wsag:Reject operation follows the same extensibility pattern as is described in wsag:CreateAgreement.

9.3.2.3 Faults

A fault indicates that rejection of this AgreementAcceptance resource is not possible and also MAY include the reason.

9.4 Port Type wsag:Agreement

9.4.1. Operation wsag:Terminate

Terminates an Agreement, if permissible. Terminating an Agreement may result in domain-specific penalty imposed on the Agreement Initiator.

9.4.1.1 Input

The form of the wsag:Terminate input message is:

```
<wsag:TerminateInput>
      xs:any
</wsag:TerminateInput>
```

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

/wsag:Terminate/xs:any

Any domain-specific content may be added. This content may be used for a variety of purposes such as logging the termination condition, or evaluating if a domain-specific cause for termination is sufficient to permit the agreement to be terminated.

9.4.1.2 Result

The result of the wsag:Terminate operation is always empty.

```
<wsag:TerminateResponse>
</wsag:TerminateResponse>
```

9.4.1.3 Faults

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

It may take some time for the terminate operation to be carried out. To avoid timeouts and other problems, the Agreement Responder MAY immediately return a TerminateResponse and then process the terminate operation. In these cases the AgreementState MUST be in **PendingAndTerminating** or **ObservedAndTerminating** state until the termination decision is made.

9.4.2 Resource Property wsag:Name

The wsag:Name resource property is of type xs:string. It MAY be empty if no name has been defined in the offer submitted.

9.4.3 Resource Property wsag:AgreementId

The wsag:AgreementId resource property is of type xs:string. It MUST be a defined and represents the ID (unique between the parties to the agreement) of the present agreement version.

9.4.4 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.

9.4.5 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 type with a QueryResourceProperty operation as defined in **[WS-ResourceProperties]**, in order to enable queries on the terms of the agreement in a more fine-grained manner.

9.5 Port Type wsag:AgreementState

The purpose of this port type is to define a resource document type for monitoring the state of the agreement. This port type is not meant to be used as is but instead, its resource properties MAY be composed into a domain-specific Agreement port type.

9.5.1 Resource Property wsag:AgreementState

The property exposes an **Agreement state** for the whole agreement as defined in section 7.1.

The set of values as defined in the wsag:AgreementStateDefinition is:

- **Pending**
- **PendingAndTerminating**
- **Observed**
- **ObservedAndTerminating**
- **Rejected**
- **Complete**
- **Terminated**

The set of values can also be extended.

The property has the following structure:

```
<wsag:AgreementState>
    <wsag:State>wsag:AgreementStateDefinition</wsag:State>
    <xs:any##other/> ?
</wsag:AgreementState>
```

/wsag:AgreementState

The agreement state type has open content so that additional domain-specific state information can be expressed.

/wsag:AgreementState/wsag:State

The agreement state must be one of the following values: Pending, PendingAndTerminating, Observed, ObservedAndTerminating, Rejected, Complete or Terminated. The individual states are as described in Section 7.1.

9.5.2 Resource Property wsag:ServiceTermState

The property exposes a **service state** for each service description term that abstractly describes the state of a service, as defined in section 7.2.

The set of values as defined in the wsag:ServiceTermStateDefiniton is:

- **NotReady**
- **Ready**
- **Completed**

The set of values can also be extended.

The primary state **Ready** has associated sub-states with values of **Processing** and **Idle.** All of the states have open-content to support other domain-specific sub-states. If open-content is used in state Ready, the Processing and Idle sub-states MUST NOT appear.

The property has the following structure:

```
<wsag:ServiceTermState termName="xs:string">
    <wsag:State>wsag:ServiceTermStateDefinition</wsag:State>
    {
      <wsag:Processing>
            <xs:any##other/>
      </wsag:Processing>  |
      <wsag:Idle>
            <xs:any##other/>
      </wsag:Idle>  |
      <xs:any##other/>
    } ?
</wsag:ServiceTermState> *
```

/wsag:ServiceTermState

List of service states indexed by name.

/wsag:ServiceTermState/@termName

Name of the term to which it refers.

/wsag:ServiceTermState/wsag:State

The state of a service term. The state can be NotReady, Ready or Completed.

/wsag:ServiceTermState/wsag:Processing

Substate of Ready.

/wsag:ServiceTermState/wsag:Idle

Substate of Ready.

9.5.3 Resource Property wsag:GuaranteeTermState

This property represents a state of fulfillment for each guarantee term of the agreement, as defined in section 7.3.

The set of values for the top-level states as defined in the wsag:GuaranteeTermStateDefinition is:

- **Fulfilled** – Currently the guarantee is fulfilled.
- **Violated** – Currently the guarantee is violated.
- **NotDetermined** – No activity regarding this guarantee has happened yet or is currently happening that allows evaluating whether the guarantee is met.

The set of values can also be extended.

The property has the following structure:

```
<wsag:GuaranteeTermState termName="xs:string">
    <wsag:State>wsag:GuaranteeTermStateDefinition</wsag:State>
    <xs:any##other/> ?
</wsag:GuaranteeTermState> *
```

/wsag:GuaranteeTermState

List of service states indexed by name.

/wsag:GuaranteeTermState/@termName

Name of the term to which it refers.

/wsag:GuaranteeTermState/wsag:State

The state of the guarantee. This can be NotDetermined, Fulfilled or Violated

10 Agreement Creation Use Case

Note: since the binding between the agreement layer and the layer of the service being provided is out of the scope of this specification, we omit the steps and operations that expose service layer services or application functionality. The agreement 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 operation, the initiator calls the CreateAgreement operation on the Factory service, passing in offer terms that satisfy the creation constraints of one of the templates exposed by the Factory as resource properties. If it is

not accepted by the Factory, the CreateAgreement operation will throw a fault message.

2. Assuming the factory accepts the terms, it returns an endpoint reference (EPR) to an *observed* Agreement service.

11 Security Considerations

The WS-Agreement specification does not explicitly address any security considerations. We expect that security issues will be addressed by blending with other security implementations in the web services domain. In particular, agreement participants SHOULD be authenticated to insure their identity of the initiator during creation and management of an agreement. Further, one MAY wish to provide a method for signing or otherwise authenticating a document based on the WS-Agreement schema if that document is to be consumed outside the interactions defined by the WS-Agreement interactions and port-types.

12 Acknowledgements

This document is the work of the GRAAP Working Group GRAAP Working Group (Grid Resource Allocation and Agreement Protocol WG) of the Compute Area of the OGF.

Members of the Working Group, and other contributors to this specification include: Takuya Araki, Dominic Battre, Christopher Dabrowski, Donal Fellows, Robert Filepp, Ian Foster, Robert Kearney, David Kaminsky, Carl Kesselman, Chris Kurowski, Jon MacLaren, Tianchao Li, Miron Livny, Steven Newhouse, Jeff Nick, Shamima Paurobally, Stephen Pickles, Volker Sander, Art Sedighi, Chris Smith, Ellen Stokes, John Sweitzer, Oliver Wäldrich, Alan Weissberger, Philipp Wieder, Yuichiro Yonebayashi, Wolfgang Ziegler.

13 References

[WSDL]

R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana:

"Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language",

W3C Candidate Recommendation, W3C, 6 January, 2006.
http://www.w3.org/TR/2006/CR-wsdl20-20060106

[WS-ResourceProperties]

S. Graham, J. Treadwell:

"Web Services Resource Properties 1.2 (WS-ResourceProperties)",

OASIS Standard, 1 April 2006

http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf

[WS-Addressing]

M. Gudgin, M. Hadley, T.Rogers:

"Web Services Addressing 1.0 - Core",

W3C Recommendation, W3C, 09 May, 2006.

http://www.w3.org/TR/2006/REC-ws-addr-core-20060509

[WS-ResourceLifetime]

L. Srinivasan, T. Banks:

"Web Services Resource Lifetime 1.2 (WS-ResourceLifetime)",

OASIS Standard, 1 April 2006
http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf

[WS-BaseFaults]
L. Liu, S. Meder:
"Web Services Base Faults 1.2 (WS-BaseFaults)",
OASIS Standard, 1 April 2006
http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf

[OGSA Profile]
T. Maguire, D. Snelling:
"OGSA Profile Definition Version 1.0",
GGF Informational Document GFD.59 , Global Grid Forum, January, 2006.
http://www.ggf.org/documents/GFD.59.pdf

[SOAP]
M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H.F. Nielsen:
"SOAP Version 1.2 Part 1: Messaging Framework",
W3C Recommendation, W3C, 24 June, 2003.
http://www.w3.org/TR/soap12-part1/.

[RFC2119]
S. Bradner (Editor):
"Key words for use in RFCs to Indicate Requirement Levels",
The Internet Engineering Task Force Best Current Practice, March, 1997.
http://www.ietf.org/rfc/rfc2119.txt

[XML-Infoset]
J. Cowan, R. Tobin:
"XML Information Set (Second Edition)",
W3C Recommendation, W3C, 4 February, 2004.
http://www.w3.org/TR/xml-infoset/

[WS-Security]
A. Nadalin, C. Kaler, P. Hallam-Baker, R. Monzillo:
"Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)",
OASIS Standard 200401, OASIS, March 2004.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[XPath]
J. Clark, S. DeRose:
"XML Path Language (XPath) Version 1.0",
W3C Recommendation, W3C, 16 November, 1999.
http://www.w3.org/TR/xpath

[XML Schema]
D. C. Fallside, P. Walmsley:
"XML Schema Part 0: Primer Second Edition",
W3C Recommendation, W3C, 28 October, 2004.
http://www.w3.org/TR/xmlschema-0/

[XML]
T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau:
"Extensible Markup Language (XML) 1.0 (Third Edition)":
W3C Recommendation, W3C, 4 February, 2004.
http://www.w3.org/TR/REC-xml

[URI]
T. Berners-Lee, R. Fielding, U.C. Irvine, 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-Resource]
S. Graham, A. Karmarkar, J. Mischkinsky, I. Robinson, I. Sedukhin:
"Web Services Resource 1.2 (WS-Resource)",
Public Review Draft 02, OASIS, 6 October, 2005.
http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-pr-02.pdf

[JSDL]
A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, A. Savva (Editor):
"Job Submission Description Language (JSDL) Specification, Version 1.0",
Grid Forum Document GFD-R-P.056, Global Grid Forum, November, 2005.
http://www.ggf.org/documents/GFD.56.pdf

Appendix 1 - XML Schema and WSDL

This appendix defines normative syntax for the XML elements and WS-Agreement specific Web service interfaces, using XML Schema Definitions and Web Service Definition Language. The definitions are authoritative for the target namespace given within the definitions themselves. The preceding specification text and pseudo-schema is also considered normative, and the authors expect that most conflicts should be resolved in favor of the main text by considering this appendix to be in error. This more accurately captures the authors' intent but requires that a new normative schema definition be published in a new namespace to correct any such errors to support conformant and interoperable implementation of WS-Agreement.

Agreement Types Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-agreement">
```

```
    <xs:import namespace="http://www.w3.org/2001/XMLSchema"
schemaLocation="http://www.w3.org/2001/XMLSchema.xsd"/>


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


    <xs:import namespace="http://docs.oasis-open.org/wsrf/bf-2"
schemaLocation="http://docs.oasis-open.org/wsrf/bf-2.xsd"/>

    <xs:element name="Template" type="wsag:AgreementTemplateType"/>
    <xs:element name="AgreementOffer" type="wsag:AgreementType"/>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="AgreementId" type="xs:string"/>
    <xs:element name="Context" type="wsag:AgreementContextType"/>
    <xs:element name="Terms" type="wsag:TermTreeType"/>
    <xs:element name="NoncriticalExtensions"
type="wsag:NoncriticalExtensionType"/>


    <xs:complexType name="TermTreeType">
        <xs:sequence>
            <xs:element minOccurs="0" ref="wsag:All"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AgreementContextType">
        <xs:sequence>
            <xs:element minOccurs="0" name="AgreementInitiator"
type="xs:anyType"/>
            <xs:element minOccurs="0" name="AgreementResponder"
type="xs:anyType"/>
            <xs:element name="ServiceProvider"
type="wsag:AgreementRoleType"/>
            <xs:element minOccurs="0" name="ExpirationTime"
type="xs:dateTime"/>
            <xs:element minOccurs="0" name="TemplateId"
type="xs:string"/>
            <xs:element minOccurs="0" name="TemplateName"
type="xs:string"/>
            <xs:any maxOccurs="unbounded" minOccurs="0"
namespace="##other" processContents="lax"/>
        </xs:sequence>
```

```
        <xs:anyAttribute namespace="##other"/>
    </xs:complexType>
    <xs:element name="All" type="wsag:TermCompositorType"/>
    <xs:complexType name="TermCompositorType">
        <xs:sequence>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="ExactlyOne"
type="wsag:TermCompositorType"/>
                <xs:element name="OneOrMore"
type="wsag:TermCompositorType"/>
                <xs:element ref="wsag:All"/>
                <xs:element name="ServiceDescriptionTerm"
type="wsag:ServiceDescriptionTermType"/>
                <xs:element name="ServiceReference"
type="wsag:ServiceReferenceType"/>
                <xs:element name="ServiceProperties"
type="wsag:ServicePropertiesType"/>
                <xs:element name="GuaranteeTerm"
type="wsag:GuaranteeTermType"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AgreementTemplateType">
        <xs:complexContent>
            <xs:extension base="wsag:AgreementType">
                <xs:sequence>
                    <xs:element name="CreationConstraints"
type="wsag:ConstraintSectionType"/>
                </xs:sequence>
                <xs:attribute name="TemplateId" type="xs:string"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="AgreementType">
        <xs:sequence>
            <xs:element minOccurs="0" ref="wsag:Name"/>
            <xs:element ref="wsag:Context"/>
            <xs:element ref="wsag:Terms"/>
        </xs:sequence>
        <xs:attribute name="AgreementId" type="xs:string"/>
    </xs:complexType>
    <xs:complexType name="AgreementInitiatorIdentifierType">
        <xs:sequence>
```

```
            <xs:element name="Reference" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AgreementResponderIdentifierType">
        <xs:sequence>
            <xs:element name="Reference" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType abstract="true" name="TermType">
        <xs:attribute name="Name" type="xs:string" use="required" />
    </xs:complexType>
    <xs:complexType name="GuaranteeTermType">
        <xs:complexContent>
            <xs:extension base="wsag:TermType">
                <xs:sequence>
                    <xs:element maxOccurs="unbounded" minOccurs="0"
name="ServiceScope" type="wsag:ServiceSelectorType"/>
                    <xs:element minOccurs="0"
ref="wsag:QualifyingCondition"/>
                    <xs:element ref="wsag:ServiceLevelObjective"/>
                    <xs:element name="BusinessValueList"
type="wsag:BusinessValueListType"/>
                </xs:sequence>
                <xs:attribute name="Obligated"
type="wsag:ServiceRoleType"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="ServiceSelectorType">
        <xs:sequence>
            <xs:any maxOccurs="unbounded" minOccurs="0"
namespace="##other" processContents="lax"/>
        </xs:sequence>
        <xs:attribute name="ServiceName" type="xs:string"/>
    </xs:complexType>
    <xs:element name="QualifyingCondition" type="xs:anyType"/>
    <xs:element name="ServiceLevelObjective"
type="wsag:ServiceLevelObjectiveType"/>
    <xs:complexType name="BusinessValueListType">
        <xs:sequence>
            <xs:element minOccurs="0" name="Importance"
type="xs:integer"/>
```

```
            <xs:element maxOccurs="unbounded" minOccurs="0"
name="Penalty" type="wsag:CompensationType"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
name="Reward" type="wsag:CompensationType"/>
            <xs:element minOccurs="0" name="Preference"
type="wsag:PreferenceType"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
name="CustomBusinessValue" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="CompensationType">
        <xs:sequence>
            <xs:element name="AssessmentInterval">
                <xs:complexType>
                    <xs:sequence>
                        <xs:choice>
                            <xs:element name="TimeInterval"
type="xs:duration"/>
                            <xs:element name="Count"
type="xs:positiveInteger"/>
                        </xs:choice>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element minOccurs="0" name="ValueUnit"
type="xs:string"/>
            <xs:element name="ValueExpression" type="xs:anyType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PreferenceType">
        <xs:sequence maxOccurs="unbounded" minOccurs="1">
            <xs:element name="ServiceTermReference" type="xs:string" />
            <xs:element name="Utility" type="xs:float"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType abstract="true" name="ServiceTermType">
        <xs:complexContent>
            <xs:extension base="wsag:TermType">
                <xs:attribute name="ServiceName" type="xs:string"
use="required" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
```

```
    <xs:complexType name="ServiceReferenceType">
        <xs:complexContent>
            <xs:extension base="wsag:ServiceTermType">
                <xs:sequence>
                    <xs:any namespace="##other"
processContents="strict"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="ServiceDescriptionTermType">
        <xs:complexContent>
            <xs:extension base="wsag:ServiceTermType">
                <xs:sequence>
                    <xs:any namespace="##other"
processContents="strict"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="ServicePropertiesType">
        <xs:complexContent>
            <xs:extension base="wsag:ServiceTermType">
                <xs:sequence>
                    <xs:element name="VariableSet"
type="wsag:VariableSetType"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="ServiceNameSet">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0"
name="ServiceName" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Location" type="xs:string"/>
    <xs:complexType name="VariableType">
        <xs:sequence>
            <xs:element ref="wsag:Location"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string"/>
        <xs:attribute name="Metric" type="xs:string"/>
```

```
    </xs:complexType>
    <xs:complexType name="VariableSetType">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" name="Variable"
type="wsag:VariableType"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ConstraintSectionType">
        <xs:sequence>
            <xs:element maxOccurs="unbounded" minOccurs="0" name="Item"
type="wsag:OfferItemType"/>
            <xs:element maxOccurs="unbounded" minOccurs="0"
ref="wsag:Constraint"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="OfferItemType">
        <xs:sequence>
            <xs:element ref="wsag:Location"/>
            <xs:element name="ItemConstraint">
                <xs:complexType>
                    <xs:choice minOccurs="0">
                        <xs:group ref="xs:simpleRestrictionModel"/>
                        <xs:group ref="xs:typeDefParticle"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
            <xs:any maxOccurs="unbounded" minOccurs="0"
processContents="lax"/>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string"/>
    </xs:complexType>
    <xs:element name="Constraint" type="xs:anyType"/>
    <xs:simpleType name="AgreementRoleType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="AgreementInitiator"/>
            <xs:enumeration value="AgreementResponder"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ServiceRoleType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="ServiceConsumer"/>
            <xs:enumeration value="ServiceProvider"/>
        </xs:restriction>
```

```
    </xs:simpleType>
    <xs:complexType name="NoncriticalExtensionType">
        <xs:sequence>
            <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="ServiceLevelObjectiveType">
        <xs:choice>
            <xs:element name="KPITarget" type="wsag:KPITargetType" />
            <xs:element name="CustomServiceLevel" type="xs:anyType" />
        </xs:choice>
    </xs:complexType>

    <xs:complexType name="KPITargetType">
        <xs:sequence>
            <xs:element name="KPIName" type="xs:string"/>
            <xs:element name="CustomServiceLevel" type="xs:anyType" />
        </xs:sequence>
    </xs:complexType>


    <!-- ////// fault section -->
    <xs:complexType name="ContinuingFaultType">
        <xs:complexContent>
            <xs:extension base="wsrf-bf:BaseFaultType"/>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="ContinuingFault"
type="wsag:ContinuingFaultType"/>


</xs:schema>
```

Agreement Factory Port Type 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:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
    xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
    xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
    xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
    xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/rpw-2"
```

```
    targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-
agreement">

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

    <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.ggf.org/graap/2007/03/ws-
agreement"
            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:include schemaLocation="agreement_types.xsd" />

            <!--Resource property element declarations-->
            <!--global elements are defined in the included schema-->
            <!--Resource property document declaration-->
            <xs:element name="AgreementFactoryProperties"
                type="wsag:AgreementFactoryPropertiesType" />
            <xs:complexType name="AgreementFactoryPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag:Template" minOccurs="0"
                        maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>

            <!-- Operational input/output type declarations -->
            <xs:element name="CreateAgreementInput"
                type="wsag:CreateAgreementInputType" />
            <xs:element name="CreateAgreementResponse"
                type="wsag:CreateAgreementOutputType" />
            <xs:complexType name="CreateAgreementInputType">
```

```
                <xs:sequence>
                    <xs:element name="InitiatorAgreementEPR"
                        type="wsa:EndpointReferenceType" minOccurs="0"
/>
                    <xs:element ref="wsag:AgreementOffer" />
                    <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:complexType name="CreateAgreementOutputType">
                <xs:sequence>
                    <xs:element name="CreatedAgreementEPR"
                        type="wsa:EndpointReferenceType" />
                    <xs:any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:complexType>

        </xs:schema>
    </wsdl:types>

    <wsdl:message name="CreateAgreementInputMessage">
        <wsdl:part name="parameters"
            element="wsag:CreateAgreementInput" />
    </wsdl:message>

    <wsdl:message name="CreateAgreementOuputMessage">
        <wsdl:part name="parameters"
            element="wsag:CreateAgreementResponse" />
    </wsdl:message>

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

    <wsdl:portType name="AgreementFactory"
        wsrf-rp:ResourceProperties="wsag:AgreementFactoryProperties">
        <wsdl:operation name="CreateAgreement">
```

```
            <wsdl:input message="wsag:CreateAgreementInputMessage" />
            <wsdl:output message="wsag:CreateAgreementOuputMessage" />
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
            <wsdl:fault name="ResourceUnavailableFault"
                message="wsrf-rw:ResourceUnavailableFault" />
            <wsdl:fault name="ContinuingFault"
                message="wsag:CreateAgreementFaultMessage" />
        </wsdl:operation>

        <!-- resource property accessor definitions from WSRF-RP -->
        <wsdl:operation name="GetResourceProperty">
            <wsdl:input name="GetResourcePropertyRequest"
                message="wsrf-rpw:GetResourcePropertyRequest" />
            <wsdl:output name="GetResourcePropertyResponse"
                message="wsrf-rpw:GetResourcePropertyResponse" />
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
            <wsdl:fault name="ResourceUnavailableFault"
                message="wsrf-rw:ResourceUnavailableFault" />
            <wsdl:fault name="InvalidResourcePropertyQNameFault"
                message="wsrf-rpw:InvalidResourcePropertyQNameFault" />
        </wsdl:operation>
    </wsdl:portType>

</wsdl:definitions>
```

Pending Agreement Factory Port Type 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:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/rpw-2"
targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-agreement">

<wsdl:import
  namespace="http://docs.oasis-open.org/wsrf/rpw-2"
  location="http://docs.oasis-open.org/wsrf/rpw-2.wsdl"/>
<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.ggf.org/graap/2007/03/ws-
agreement"
      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:include schemaLocation="agreement_types.xsd"/>

      <!--Resource property element declarations-->
      <!--global elements are defined in the included schema-->
      <!--Resource property document declaration-->

      <xs:element
        name="PendingAgreementFactoryProperties"
        type="wsag:PendingAgreementFactoryPropertiesType"/>
        <xs:complexType name="PendingAgreementFactoryPropertiesType">
          <xs:sequence>
            <xs:element ref="wsag:Template" minOccurs="0"
                        maxOccurs="unbounded"/>
           </xs:sequence>
        </xs:complexType>

      <!-- Operational input/output type declarations -->

      <xs:element name="CreatePendingAgreementInput"
                  type="wsag:CreatePendingAgreementInputType"/>
      <xs:element name="CreatePendingAgreementResponse"
                  type="wsag:CreatePendingAgreementOutputType"/>
      <xs:complexType name="CreatePendingAgreementInputType">
        <xs:sequence>
          <xs:element name="AgreementAcceptanceEPR"
                      type="wsa:EndpointReferenceType" minOccurs="0"/>
          <xs:element name="InitiatorAgreementEPR"
                      type="wsa:EndpointReferenceType" minOccurs="0"/>
```

```
        <xs:element ref="wsag:AgreementOffer"/>
        <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:complexType name="CreatePendingAgreementOutputType">
      <xs:sequence>
        <xs:element name="CreatedAgreementEPR"
                    type="wsa:EndpointReferenceType"/>
        <xs:any namespace="##other" processContents="lax"
                minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:schema>

  </wsdl:types>
    <wsdl:message name="CreatePendingAgreementInputMessage">
      <wsdl:part name="parameters"
                 element="wsag:CreatePendingAgreementInput"/>
    </wsdl:message>
    <wsdl:message name="CreatePendingAgreementOuputMessage">
          <wsdl:part name="parameters"
                     element="wsag:CreatePendingAgreementResponse"/>
    </wsdl:message>
    <wsdl:message name="CreateAgreementFaultMessage">
      <wsdl:part name="fault"
                      element="wsag:ContinuingFault"/>
    </wsdl:message>

    <wsdl:portType
      name="PendingAgreementFactory"
  wsrf-rp:ResourceProperties="wsag:PendingAgreementFactoryProperties"
    >
      <wsdl:operation name="CreatePendingAgreement">
        <wsdl:input
          message="wsag:CreatePendingAgreementInputMessage"/>
        <wsdl:output
          message="wsag:CreatePendingAgreementOuputMessage"/>
        <wsdl:fault name="ResourceUnknownFault"
          message="wsrf-rw:ResourceUnknownFault"/>
```

```
            <wsdl:fault name="ResourceUnavailableFault"
                  message="wsrf-rw:ResourceUnavailableFault" />
            <wsdl:fault name="ContinuingFault"
                  message="wsag:CreateAgreementFaultMessage" />
        </wsdl:operation>

        <!-- resource property accessor definitions from WSRF-RP -->
        <wsdl:operation name="GetResourceProperty">
          <wsdl:input name="GetResourcePropertyRequest"
            message="wsrf-rpw:GetResourcePropertyRequest"/>
          <wsdl:output name="GetResourcePropertyResponse"
            message="wsrf-rpw:GetResourcePropertyResponse"/>
          <wsdl:fault name="ResourceUnknownFault"
            message="wsrf-rw:ResourceUnknownFault"/>
          <wsdl:fault name="ResourceUnavailableFault"
            message="wsrf-rw:ResourceUnavailableFault"/>
          <wsdl:fault name="InvalidResourcePropertyQNameFault"
            message="wsrf-rpw:InvalidResourcePropertyQNameFault"/>
        </wsdl:operation>
      </wsdl:portType>

</wsdl:definitions>
```

Agreement Acceptance Port Type 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:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-agreement">

<wsdl:import
  namespace="http://docs.oasis-open.org/wsrf/rpw-2"
  location="http://docs.oasis-open.org/wsrf/rpw-2.wsdl"/>
<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.ggf.org/graap/2007/03/ws-
agreement"
      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:include schemaLocation="agreement_types.xsd"/>

      <!--Resource property element declarations-->
      <!--global elements are defined in the included schema-->
      <!--No Resource property document declaration-->

      <!-- Operational input/output type declarations -->
      <xs:element name="AcceptAgreementInput"
        type="wsag:AgreementAcceptanceInputType"/>
      <xs:element name="AcceptAgreementResponse"
        type="wsag:AgreementAcceptanceOutputType"/>
      <xs:element name="RejectAgreementInput"
        type="wsag:AgreementAcceptanceInputType"/>
      <xs:element name="RejectAgreementResponse"
        type="wsag:AgreementAcceptanceOutputType"/>
      <xs:complexType name="AgreementAcceptanceInputType">
        <xs:sequence>
          <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:complexType name="AgreementAcceptanceOutputType">
        <xs:sequence>
          <xs:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:schema>
```

```
    </wsdl:types>
      <wsdl:message name="AcceptAgreementInputMessage">
        <wsdl:part name="parameters"
                   element="wsag:AcceptAgreementInput"/>
      </wsdl:message>
      <wsdl:message name="AcceptAgreementOuputMessage">
        <wsdl:part name="parameters"
                   element="wsag:AcceptAgreementResponse"/>
      </wsdl:message>
      <wsdl:message name="RejectAgreementInputMessage">
        <wsdl:part name="parameters"
                   element="wsag:RejectAgreementInput"/>
      </wsdl:message>
      <wsdl:message name="RejectAgreementOuputMessage">
            <wsdl:part name="parameters"
                       element="wsag:RejectAgreementResponse"/>
      </wsdl:message>

      <wsdl:portType name="AgreementAcceptance">
        <wsdl:operation name="AcceptAgreement">
          <wsdl:input message="wsag:AcceptAgreementInputMessage"/>
          <wsdl:output message="wsag:AcceptAgreementOuputMessage"/>
          <wsdl:fault name="ResourceUnknownFault"
                      message="wsrf-rw:ResourceUnknownFault"/>
        </wsdl:operation>
        <wsdl:operation name="RejectAgreement">
          <wsdl:input message="wsag:RejectAgreementInputMessage"/>
          <wsdl:output message="wsag:RejectAgreementOuputMessage"/>
          <wsdl:fault name="ResourceUnknownFault"
                      message="wsrf-rw:ResourceUnknownFault"/>
        </wsdl:operation>
      </wsdl:portType>
</wsdl:definitions>
```

Agreement Port Type 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:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
    xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
    xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
    xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
    xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/rpw-2"
```

```
    targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-
agreement">

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

    <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.ggf.org/graap/2007/03/ws-
agreement"
            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:include schemaLocation="agreement_types.xsd" />
            <xs:include schemaLocation="agreement_state_types.xsd"/>

            <!--Resource property element declarations-->
            <!--global elements are defined in the included schema-->
            <!--Resource property document declaration-->
            <xs:element name="AgreementProperties"
                type="wsag:AgreementPropertiesType" />
            <xs:complexType name="AgreementPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag:Name" minOccurs="0" />
                    <xs:element ref="wsag:AgreementId" />
                    <xs:element ref="wsag:Context" />
                    <xs:element ref="wsag:Terms" />

                </xs:sequence>
            </xs:complexType>
            <!--======================-->
            <!-- Operational input/output type declarations -->
            <xs:element name="TerminateInput"
                type="wsag:TerminateInputType" />
            <xs:element name="TerminateResponse"
                type="wsag:TerminateOutputType" />
            <xs:complexType name="TerminateInputType">
```

```
                <xs:sequence>
                    <xs:any processContents="lax" namespace="##any" />
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="TerminateOutputType" />
        </xs:schema>
    </wsdl:types>

    <wsdl:message name="TerminateInputMessage">
        <wsdl:part name="parameters" element="wsag:TerminateInput" />
    </wsdl:message>
    <wsdl:message name="TerminateOuputMessage">
        <wsdl:part name="parameters" element="wsag:TerminateResponse"
/>
    </wsdl:message>


    <wsdl:portType name="Agreement"
        wsrf-rp:ResourceProperties="wsag:AgreementProperties">

        <!-- resource property accessor definitions from WSRF-RP -->
        <wsdl:operation name="GetResourceProperty">
            <wsdl:input name="GetResourcePropertyRequest"
                message="wsrf-rpw:GetResourcePropertyRequest" />
            <wsdl:output name="GetResourcePropertyResponse"
                message="wsrf-rpw:GetResourcePropertyResponse" />
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
            <wsdl:fault name="ResourceUnavailableFault"
                message="wsrf-rw:ResourceUnavailableFault" />
            <wsdl:fault name="InvalidResourcePropertyQNameFault"
                message="wsrf-rpw:InvalidResourcePropertyQNameFault" />
        </wsdl:operation>

        <wsdl:operation name="Terminate">
            <wsdl:input name="TerminateRequest"
                message="wsag:TerminateInputMessage" />
            <wsdl:output name="TerminateResponse"
                message="wsag:TerminateOuputMessage" />
            <wsdl:fault name="ResourceUnknownFault"
                message="wsrf-rw:ResourceUnknownFault" />
        </wsdl:operation>
    </wsdl:portType>
```

```
</wsdl:definitions>
```

AgreementState Port Type 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:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
      xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
      xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
      xmlns:wsrf-rw="http://docs.oasis-open.org/wsrf/rw-2"
      xmlns:wsrf-rpw="http://docs.oasis-open.org/wsrf/rpw-2"
      targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-
agreement">

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

      <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.ggf.org/graap/2007/03/ws-
agreement"
                  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:include
schemaLocation="agreement_state_types.xsd" />

                  <!--Resource property element declarations-->
                  <!--global elements are defined in the included
schema-->
                  <!--Resource property document declaration-->
                  <xs:element name="AgreementStateProperties"
                        type="wsag:AgreementStatePropertiesType" />
                  <xs:complexType name="AgreementStatePropertiesType">
                        <xs:sequence>
```

```
                              <xs:element ref="wsag:AgreementState" />
                              <xs:element ref="wsag:GuaranteeTermState"
                                    minOccurs="0" maxOccurs="unbounded"
/>
                              <xs:element ref="wsag:ServiceTermState"
                                    minOccurs="0" maxOccurs="unbounded"
/>
                        </xs:sequence>
                  </xs:complexType>
            </xs:schema>

      </wsdl:types>
      <wsdl:portType name="AgreementState"
            wsrf-rp:ResourceProperties="wsag:AgreementStateProperties">
            <!-- resource property accessor definitions from WSRF-RP --
>
            <wsdl:operation name="GetResourceProperty">
                  <wsdl:input name="GetResourcePropertyRequest"
                        message="wsrf-rpw:GetResourcePropertyRequest"
/>
                  <wsdl:output name="GetResourcePropertyResponse"
                        message="wsrf-rpw:GetResourcePropertyResponse"
/>
                  <wsdl:fault name="ResourceUnknownFault"
                        message="wsrf-rw:ResourceUnknownFault" />
                  <wsdl:fault name="ResourceUnavailableFault"
                        message="wsrf-rw:ResourceUnavailableFault" />
                  <wsdl:fault name="InvalidResourcePropertyQNameFault"
                        message="wsrf-
rpw:InvalidResourcePropertyQNameFault" />
            </wsdl:operation>
      </wsdl:portType>
</wsdl:definitions>
```

Agreement State Types Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://schemas.ggf.org/graap/2007/03/ws-agreement">

    <xs:simpleType name="AgreementStateDefinition">
        <xs:restriction base="xs:string">
            <xs:enumeration value="Pending"/>
            <xs:enumeration value="PendingAndTerminating"/>
            <xs:enumeration value="Observed"/>
            <xs:enumeration value="ObservedAndTerminating"/>
            <xs:enumeration value="Rejected"/>
            <xs:enumeration value="Complete"/>
            <xs:enumeration value="Terminated"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="GuaranteeTermStateDefinition">
        <xs:restriction base="xs:string">
            <xs:enumeration value="NotDetermined"/>
            <xs:enumeration value="Fulfilled"/>
            <xs:enumeration value="Violated"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="ServiceTermStateDefinition">
        <xs:restriction base="xs:string">
            <xs:enumeration value="NotReady"/>
            <xs:enumeration value="Ready"/>
            <xs:enumeration value="Completed"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="AgreementStateType">
      <xs:complexContent>
        <xs:extension base="wsag:InnerAgreementStateType">
          <xs:sequence>
            <xs:element name="State"
type="wsag:AgreementStateDefinition"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="GuaranteeTermStateType">
```

```
      <xs:complexContent>
        <xs:extension base="wsag:TermStateType">
          <xs:sequence>
            <xs:element name="State"
type="wsag:GuaranteeTermStateDefinition"/>
            <xs:any namespace="##other" processContents="lax"
minOccurs="0"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="ServiceTermStateType">
      <xs:complexContent>
        <xs:extension base="wsag:TermStateType">
          <xs:sequence>
            <xs:element name="State"
type="wsag:ServiceTermStateDefinition"/>
              <xs:choice minOccurs="0">
                  <xs:any namespace="##other" processContents="lax"/>
                  <!--Processing and Idle only as substates of Ready-->
                  <xs:element name="Processing"
type="wsag:InnerTermStateType"/>
                  <xs:element name="Idle"
type="wsag:InnerTermStateType"/>
              </xs:choice>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>

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

    <xs:complexType name="TermStateType">
        <xs:attribute name="termName" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="InnerTermStateType">
```

```
        <xs:sequence>
            <xs:any namespace="##other" processContents="lax"/>
        </xs:sequence>
    </xs:complexType>

    <!--global elements are defined in the imported type library-->
    <xs:element name="AgreementState" type="wsag:AgreementStateType"/>
    <xs:element name="GuaranteeTermState"
type="wsag:GuaranteeTermStateType"/>
    <xs:element name="ServiceTermState"
type="wsag:ServiceTermStateType"/>

</xs:schema>
```

Appendix 2 - Job Submission Example

The job submission scenario mentioned in Section 2 involves an Agreement that describes a single job, where the initiator of the Agreement is the one submitting the job. The templates for such a scenario will show the job description term language that is supported by the job hosting service and MAY also indicate default values or constraints on what values are supported for jobs executed by that service.

The following example template shows the use of a prototype JSDL **[JSDL]** document as a service term and specifies some default values in the prototype. Specifically, it shows:

- The use of the optional jsdl-posix:POSIXApplication sub-language
- Default 1 MiB file size limit
- Default 0 byte core dump size limit
- Default 64 open file descriptors limit
- Default "LINUX" operating system
- Default "x86" CPU type
- Default 1.6 GHz CPU speed
- Default 2 CPUs per node
- Default 100 Mb/s network connectivity for nodes
- Default 1 node per job

And the creation constraints indicate limited values for some options. Specifically, they constrain:

- Maximum 500 MiB file size limit
- Maximum 500 MiB core dump size limit
- Maximum 1024 open file descriptors limit
- Choice of "LINUX" or "FreeBSD" operating systems
- Choice of "x86", "x86_32", or "x86_64" CPU types
- Choice of 10, 100, or 1000 Mb/s network connectivity for nodes
- Choice of 1, 2, or 4 CPUs per node
- Maximum of 128 nodes per job

No complex constraints are given to show how fields are inter-related, so the consumer of the template does not know if there are limited numbers of nodes with particular OS, CPU, network, or SMP types.

```
<wsag:Template
...
  xsi:jsdl="http://schemas.ggf.org/jsdl/2005/06/jsdl"
  xsi:jsdl-posix="http://schemas.ggf.org/jsdl/2005/06/jsdl-posix"
... />
...
<wsag:ServiceDescriptionTerm
   wsag:Name="Job JSDL" wsag:ServiceName="Job">
   <jsdl:JobDefinition>
      <JobDescription>
         <Application>
            <jsdl-posix:POSIXApplication>
               <FileSizeLimit>1048576</FileSizeLimit>
             <CoreDumpLimit>0</CoreDumpLimit>
             <OpenDescriptorsLimit>64</OpenDescriptorsLimit>
          </jsdl-posix:POSIXApplication>
         </Application>
         <Resources ...>
          <OperatingSystem>
             <OperatingSystemType>
                <OperatingSystemName>LINUX</OperatingSystemName>
             </OperatingSystemType>
          </OperatingSystem>
          <CPUArchitecture>
             <CPUArchitectureName>x86</CPUArchitectureName>
          </CPUArchitecture>
          <IndividualCPUSpeed>
             <Exact>1600000</Exact>
          </IndividualCPUSpeed>
          <IndividualCPUCount>
             <Exact>2.0</Exact>
          </IndividualCPUCount>
          <IndividualNetworkBandwidth>
             <Exact>100000000</Exact>
          </IndividualNetworkBandwidth>
          <TotalResourceCount>
             <Exact>1</Exact>
          </TotalResourceCount>
```

```
         </Resources>
      </JobDescription>
   <jsdl:JobDefinition>
</wsag:ServiceDescriptionTerm>
...
<wsag:Item>
   <Location>//jsdl-posix:FileSizeLimit</Location>
   <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="524288000"/>
   </xs:restriction>
</wsag:Item>
<wsag:Item>
   <Location>//jsdl-posix:CoreDumpLimit</Location>
   <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="524288000"/>
   </xs:restriction>
</wsag:Item>
<wsag:Item>
   <Location>//jsdl-posix:OpenDescriptorsLimit</Location>
   <xs:restriction base="xs:positiveInteger">
      <xs:maxInclusive value="1024"/>
   </xs:restriction>
</wsag:Item>
<wsag:Item>
   <Location>//jsdl:CPUArchitecture/CPUArhitecturename</Location>
   <xs:restriction base="jsdl:ProcessorArchitectureEnumeration">
      <enumeration value="x86_32"/>
      <enumeration value="x86_64"/>
      <enumeration value="x86"/>
   </xs:restriction>
</wsag:Item>
<wsag:Item>

<Location>//jsdl:OperatingSystem/jsdl:OperatingSystemType/jsdl:Operatin
gSystemName</Location>
   <restriction base="jsdl:OperatingSystemTypeEnumeration">
       <enumeration value="LINUX"/>
       <enumeration value="FreeBSD"/>
   </restriction>
</wsag:Item>
<wsag:Item>
   <wsag:Location>//jsdl:IndividualNetworkBandwidth</wsag:Location>
   <xs:sequence>
```

```
      <xs:element name="Exact" minOccurs="1" maxOccurs="unbounded">
         <xs:simpleType>
            <xs:restriction base="xs:double">
               <xs:enumeration value="1000000000"/>
               <xs:enumeration value="100000000"/>
               <xs:enumeration value="10000000"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
   </xs:sequence>
</wsag:Item>
<wsag:Item>
   <wsag:Location>//jsdl:IndividualCPUCount</wsag:Location>
   <xs:sequence>
      <xs:element name="Exact" minOccurs="1" maxOccurs="unbounded">
         <xs:simpleType>
            <xs:restriction base="xs:double">
               <xs:enumeration value="1"/>
               <xs:enumeration value="2"/>
               <xs:enumeration value="4"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
   </xs:sequence>
</wsag:Item>
<wsag:Item>
   <wsag:Location>//jsdl:TotalResourceCount</wsag:Location>
   <xs:sequence>
      <xs:element name="Exact" minOccurs="1" maxOccurs="unbounded">
         <xs:simpleType>
            <xs:restriction base="xs:double">
               <xs:maxInclusive value="128"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:element>
   </xs:sequence>
</wsag:Item>
```

It is worth noting that JSDL includes a rich value constraint language for expressing limits, intervals, and enumerations that is used for many numeric fields, as in the jsdl:IndividualNetworkBandwidth and subsequent fields constrained in our example. The interval semantics of this domain-specific constraint language are clear and allow for simple intersection/conformance checks to see how two such range

expressions overlap. Therefore, the JSDL range expressions could be useful in templates to describe the numerical limitations on range expressions that may appear in agreement offers.

Rather than using standard WS-Agreement syntactic restrictions for creation constraints, as shown in the example template above, a community using JSDL and WS-Agreement to their fullest might define a specialized creation constraint element that uses the JSDL range expression type natively. With this, the template author could easily advertise complicated semantic constraints that any JSDL agreement author should understand, without worrying about the syntactic representation of the values. In the above template example, we are not only restricting the allowed numeric values but also disabling much of the rich JSDL range language through our simplistic syntactic restrictions.

Given the preceding template example, a submitting client might generate an offer such as in the following example. This offer overrides some of the default values while observing the creation constraints, and also includes additional JSDL language elements which are supported by the schema of the top-level term document even though they are not present in the template's prototype document, nor are they constrained in any way according to the schema.

Specifically, this offer:

- Provides an executable program file name
- Provides an argument list to the program
- Provides input/output/error file redirections
- Provides a working directory name for the job
- Selects a 16 MiB file size limit
- Keeps the default 0 byte core dump size limit
- Selects the maximum 1024 open file descriptors limit
- Selects the "LINUX" operating system
- Selects the "x86_32" CPU type
- Keeps the default (fixed??) 1.6 GHz CPU speed
- Sets optional CPU time limits to schedule the job to run for at least 15 minutes but no more than one hour of CPU time per CPU
- Selects dual-CPU SMP nodes
- Selects 100/1000 Mb/s network speeds (the scheduler can choose which)
- Selects 4 nodes for the job

The Agreement responder can validate that this offer meets the criteria for the template and also can check whether the job hosting service has sufficient resources to handle this job, before deciding whether to accept or reject the agreement offer. If accepted, the resulting Agreement resource MUST have the same terms as in the offer, so we do not provide an additional agreement document example.

An offer:

```
<wsag:AgreementOffer AgreementId=”JobAgreement123”
   ... >
...

<wsag:Terms>
  <wsag:All>
    <wsag:ServiceDescription wsag:ServiceName=”Job”>
      <jsdl:JobDefinition id="xs:ID"?>
```

```
        <jsdl:JobDescription>
          <jsdl:Application>
            <jsdl-posix:POSIXApplication name="xs:NCName"?>
              <Executable>/home/user1/test/program</Executable>
              <Argument>-benchmark</Argument>
              <Argument>-output</Argument>
              <Argument>-</Argument>
              <Input>input</Input>
              <Output>output</Output>
              <Error>error</Error>
              <WorkingDirectory>/home/user1/test/1</WorkingDirectory>
              <FileSizeLimit>16777216</FileSizeLimit>
              <CoreDumpLimit>0</CoreDumpLimit>
              <OpenDescriptorsLimit>1024</OpenDescriptorsLimit>
            </jsdl-posix:POSIXApplication>
          </jdsl:Application>
          <Resources ...>
            <OperatingSystem>
              <OperatingSystemType>
                <OperatingSystemName>LINUX</OperatingSystemName>
              </OperatingSystemType>
            </OperatingSystem>
            <CPUArchitecture>
              <CPUArchitectureName>x86_32</CPUArchitectureName>
            </CPUArchitecture>
            <IndividualCPUSpeed>
              <jsdl:Exact>1600000</jsdl:Exact>
            </IndividualCPUSpeed>
            <IndividualCPUTime>
              <jsdl:Range>
                <jsdl:LowerBound>900</jsdl:LowerBound>
                <jsdl:UpperBound>3600</jsdl:UpperBound>
              </jsdl:Range>
            </IndividualCPUTime>
            <IndividualCPUCount>
              <jsdl:Exact>2.0</jsdl:Exact>
            </IndividualCPUCount>
            <IndividualNetworkBandwidth>
              <jsdl:Exact>1000000000</jsdl:Exact>
              <jsdl:Exact>100000000</jsdl:Exact>
            </IndividualNetworkBandwidth>
            <TotalResourceCount>
              <jsdl:Exact>4</jsdl:Exact>
```

```
            </TotalResourceCount>
          </Resources>
        </jsdl:JobDescription>
      <jsdl:JobDefinition>
    </wsag:ServiceDescription>
  </wsag:All>
</wsag:Terms>
</wsag:AgreementOffer>
```

Appendix 3 - Preference Example

Preference business values in guarantee terms can be used to guide which the choice of, for example, system configurations for jobs. The following example illustrates this, using a hypothetical job submission language.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:AgreementOffer
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:job="http://www.gridforum.org/namespaces/job"
  xmlns:sdtc="http://foo.org/sdtc"
  xsi:schemaLocation="http http://schemas.ggf.org/graap/2007/03/ws-
agreement agreement_types.xsd http://www.w3.org/2001/XMLSchema
XMLSchema.xsd http://www.gridforum.org/namespaces/job job_terms.xsd
http://foo.org/sdtc SDTCondition.xsd"
  wsag:AgreementId="PreferenceExample">

  <wsag:Name>Offer2</wsag:Name>
  <wsag:Context/>
  <wsag:Terms>
    <wsag:All>

      <!-- job submission example based on a hypothetical
           job submission language for service description and
           and expressions -->
      <wsag:ExactlyOne>
        <wsag:All>
          <wsag:ServiceDescriptionTerm
            wsag:Name="numberOfCPUsHigh"
            wsag:ServiceName="ComputeJob1">
```

```
          <job:numberOfCPUs>32</job:numberOfCPUs>
        </wsag:ServiceDescriptionTerm>
        <wsag:ServiceDescriptionTerm
          wsag:Name="memoryPerCPUHigh"
          wsag:ServiceName="ComputeJob1">
          <job:realMemorySize>200</job:realMemorySize>
        </wsag:ServiceDescriptionTerm>
      </wsag:All>
       <wsag:All>
        <wsag:ServiceDescriptionTerm
          wsag:Name="numberOfCPUsLow"
          wsag:ServiceName="ComputeJob1">
          <job:numberOfCPUs>8</job:numberOfCPUs>
        </wsag:ServiceDescriptionTerm>
        <wsag:ServiceDescriptionTerm
          wsag:Name="memoryPerCPULow"
          wsag:ServiceName="ComputeJob1">
          <job:realMemorySize>1000</job:realMemorySize>
        </wsag:ServiceDescriptionTerm>
      </wsag:All>
      </wsag:ExactlyOne>
        <wsag:GuaranteeTerm wsag:Name="ConfigurationPreference"
                            wsag:Obligated=”ServiceProvider”>
          <wsag:ServiceScope>
            <wsag:ServiceName>ComputeJob1</wsag:ServiceName>
          </wsag:ServiceScope>
          <wsag:ServiceLevelObjective xsi:type="sdtc:OpType">
            <Or>
              <SDT>numberOfCPUsHigh</SDT>
              <SDT>numberOfCPUsLow</SDT>
            </Or>
           </wsag:ServiceLevelObjective>
           <wsag:BusinessValueList>
             <wsag:Preference>
               <wsag:ServiceTermReference>numberOfCPUsHigh
               </wsag:ServiceTermReference>
               <wsag:Utility>0.8</wsag:Utility>
               <wsag:ServiceTermReference>numberOfCPUsLow
               </wsag:ServiceTermReference>
               <wsag:Utility>0.5</wsag:Utility>
             </wsag:Preference>
           </wsag:BusinessValueList>
         </wsag:GuaranteeTerm>
```

```
         </wsag:All>
      </wsag:Terms>
</wsag:AgreementOffer>
```

In this example, the following simple condition language is used:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://foo.org/sdtc" xmlns:sdtc="http://foo.org/sdtc">
      <complexType name="OpType">
            <sequence>
                  <choice minOccurs="1" maxOccurs="2">
                        <element name="SDT" type="string"></element>
                        <element name="And"
                                 type="sdtc:OpType"></element>
                        <element name="Or"
                                 type="sdtc:OpType"></element>
                  </choice>
            </sequence>
      </complexType>
</schema>
```

Guarantee terms are used where the particular performance of an aspect of a service is subject to a business value. In the example, a high number of CPUs is associated with a utility of 0.8 while a low number of CPUs has a utility of 0.5, expressing the preference of a high number of CPUs available for the job to a low number.

Appendix 4 - Reference Type Examples

The example shows a service that defines a Web as well as a Web service interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Agreement
  xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
  xmlns:sdtc="http://foo.org/sdtc"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.ggf.org/graap/2007/03/ws-agreement
agreement_types.xsd http://foo.org/sdtc SDTCondition.xsd
http://www.w3.org/2001/XMLSchema XMLSchema.xsd"
  AgreementId=”Agreement3”>

<wsag:Name>Offer3</wsag:Name>
```

```
      <wsag:Context/>
      <wsag:Terms>
            <wsag:All>
                  <wsag:ServiceReference
                        wsag:Name="WSDLInterface"
                        wsag:ServiceName="BankingService">
                    <sdtc:WSDLReference>
                         http://www.foo.org/interfaces/bank.wsdl
                    </sdtc:WSDLReference>
                  </wsag:ServiceReference>
                  <wsag:ServiceReference
                        wsag:Name="WebAccess"
                        wsag:ServiceName="BankingService">
                      <sdtc:URLPrefixDefinition>
                          http://www.foo.org/bank
                      </sdtc:URLPrefixDefinition>
                  </wsag:ServiceReference>
            </wsag:All>

            <!-- More Terms -->

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

The following – domain-specific – simple schema is used for the references:

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://foo.org/sdtc" xmlns:sdtc="http://foo.org/sdtc">

      <element name="WSDLReference" type="anyURI"></element>
      <element name="URLPrefixDefinition" type="anyURI"></element>

</schema>
```