**Editors:**
K. Czajkowski, USC/ISI
A. Dan, IBM
J. Rofrano, IBM
S. Tuecke, ANL
M. Xu, Platform

September 25, 2003

# Agreement-based Service Management (WS-Agreement)

# Version 0

## Status of this Memo

This document provides information to the community regarding the specification of the Agreement-Based Service Management (WS-Agreement) model. Distribution of this document is unlimited.

## Abstract

The *Open Grid Services Architecture* (OGSA) integrates Grid technologies with *Web services* mechanisms to create a distributed computing framework based around the *Open Grid Services Infrastructure* (OGSI). The scalable deployment of OGSA environments will require support for *Service Management:* the ability to create Grid services and adjust their policies and behaviors based on organizational goals and application requirements. In practice, the Grid environment will include legacy applications and resources that are not Grid services, yet would benefit from the same management mechanisms. This specification proposes the *Agreement-based Service Management* (WS-Agreement) model, defining a set of OGSI-compatible portTypes through which management applications and services can negotiate with management services for the purpose of managing Grid services and other applications or resources. These negotiations dynamically mediate between users and service providers within *virtual organizations*, related by potentially complex community relationships. The WS-Agreement negotiation model allows management in these environments where centralized control is impossible. The fundamental mechanism of WS-Agreement is the creation of OGSI-compliant *Agreement service*s each of which represents an ongoing relationship between an *agreement provider* and a customer (management client) known in WS-Agreement as the *agreement initiator*. This agreement defines behavior of a *delivered service* with respect to a *service consumer*. These roles may be served by distinct entities or merged, depending on the scenario. Many different service control and monitoring strategies are possible for an agreement provider to satisfy its agreements, and WS-Agreement does not require any specific approach.

The WS-Agreement service interfaces are proposed as OGSI portTypes in a document-extensible style, to support richly expressive extensions. Companion specifications are expected which will provide domain-specific agreement terms and explain common usage scenarios for those terms. In anticipation of such specifications, we also define optional negotiation terms suitable for use as metadata in domain-specific term languages. These metadata terms are too abstract to form stand-alone agreements, but they can express important concepts that the domain-specific terms will otherwise have to recapture.

**GLOBAL GRID FORUM**

**office@gridforum.org**
**www.ggf.org**

## Full Copyright Notice



## Intellectual Property Statement

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

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

Explicit statements about IPR should not be included in the document, because including a specific claim implies that the claim is valid and that the listed claims are exhaustive. Once a document has been published as a final GFD there is no mechanism to effectively update the IPR

information. Authors should instead provide the GGF secretariat with any explicit statements or potentially relevant claims.

## Contents

# 1 Introduction

The *Open Grid Services Architecture* (OGSA) integrates Grid technologies with *Web services* mechanisms to create a distributed computing framework based around the *Open Grid Services Infrastructure* (OGSI). Grid environments based on OGSA concepts will include many dynamic and stateful *Grid services*, each subject to a mixture of distributed and localized policies and resource capabilities. The scalable deployment of such environments will require support for *Service Management:* the ability to create Grid services and adjust their policies and behaviors based on organizational goals and application requirements. In practice, the Grid environment will include legacy applications and resources that are not Grid services yet would benefit from the same management mechanisms. This specification proposes the *Agreement-based Service Management* (WS-Agreement) model, defining a set of OGSI-compatible portTypes through which management applications and services can negotiate with management services for the purpose of managing Grid services or legacy resources and applications. These negotiations dynamically mediate between users and service providers within *virtual organizations*, related by potentially complex community relationships.

The fundamental mechanism of WS-Agreement is the creation of OGSI-compliant *Agreement services*, each of which represents an ongoing relationship between an *agreement provider* and a customer known in WS-Agreement as the *agreement initiator*. This agreement defines behavior of a *delivered service* with respect to a *service consumer*. The Agreement service captures the service behavior in potentially domain-specific *agreement terms*. The service and agreement providers may be different entities or may simply be different roles of a single entity. The Agreement service lifetime corresponds to the lifetime of the agreement terms as they affect the delivered service behavior. Even after the termination of the Agreement service and its corresponding service-provider behaviors, the agreement initiator and agreement provider may have other secondary obligations due to the terms of the negotiated agreement, e.g. costs or penalties associated with the services rendered. Furthermore, the agreement may include terms about its own finalization, such as a requirement to pass termination status information to an external auditing service.

This specification defines WS-Agreement mechanisms in terms of OGSI portType definitions, including generic interfaces through which extensible agreement terms can be added to support particular service (or application) domains. Each Agreement service is capable of supporting OGSI monitoring and life-cycle mechanisms shared by all Grid services. This specification describes in detail how a generic WS-Agreement negotiation may proceed, identifying base functionality for all management components and highlighting the intended use of extensibility features.

The two main applications envisioned for WS-Agreement services are:

1. Managed *operation* of application or domain-specific services whether first-class Grid services or second-class legacy components, for example to control the policies and "service-level agreements" which parameterize or affect service-provider behavior.
2. Managed *creation* of such services, for example through the orchestrated deployment and instantiation of supporting Grid services or other service-providing entities.

The WS-Agreement service interfaces are written in a document-extensible style, to support richly expressive extensions. Companion specifications are expected which will provide domain-specific agreement terms and explain common usage scenarios for those terms. In anticipation of such specifications, we also define optional negotiation terms suitable for use as metadata in

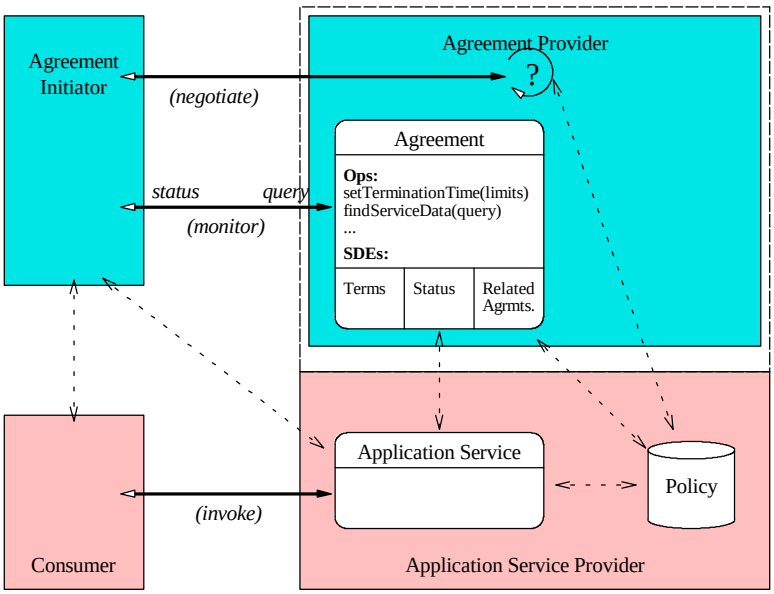


**Figure 1: In WS-Agreement, agreements are represented as Grid services. Two kinds of operation are provided in the interface of an *agreement provider* for use by the (client) *agreement initiator*: anegotiation operations establishe agreements and monitoring operations follows the status of agreements. In the figure, solid arrows point with solid arrowhead towards the invoked service, and with hollow arrowheads towards the client who receives results. Agreement *terms* are an abstract description of behavior, capturing requirements on the behavior of the *application service* with respect to a (client) *user*. The many possible monitoring and effects relationships between components are represented in the figure with dashed, bidirectional arrows; not all such links are necessarily present in every scenario.**

domain-specific term languages, and optional usage terms using the Web Services Policy Framework [WS-Policy] for adding meta-data to agreement terms expressed as policy assertions. These metadata terms are too abstract to form stand-alone agreements, but they can express important concepts that the domain-specific terms will otherwise have to recapture.

## 2 Notational Conventions

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

This specification uses namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant.

**Table 1: Prefixes and Namespaces used in this specification.**

| Prefix | Namespace |
|---|---|
| wsa | Some URI |
| ogsi | "http://www.gridforum.org/namespaces/2003/03/OGSI" |
| wsp | “http://schemas.xmlsoap.org/ws/2002/12/policy” |
| xsd | "http://www.w3.org/2001/XMLSchema" |
| xsi | "http://www.w3.org/2001/XMLSchema-instance" |

Namespace names of the general form "http://example.org/..." and "http://example.com/..." represent application or context-dependent URIs [RFC 2396].

The following abbreviations and terms are used in this document:

- The terms *GSH*, *GSR*, and *SDE* are as defined in [Cite OGSI].
- The terms *Web services*, *XML*, *SOAP*, and *WSDL* are as defined in [Grid Physiology].

# 3 Setting the Context

Elsewhere we provide the overall motivation for the Open Grid Services Architecture (OGSA) [Grid Physiology], and a detailed description of the architecture [OGSI]. We describe here an approach to manage OGSA service environments using the abstraction of agreement negotiation. We call this approach the Agreement-Based Service Management model (WS-Agreement). In this section, we describe the scope and flavor of WS-Agreement by relating it to other OGSA and OGSI concepts as well as to scenarios not specific to OGSA.

## *3.1 Service and Resource Management through Agreement*

In the Grid environment, users or their agents make application-oriented requests which are affected by resource-oriented policies expressed by resource owners. Many, possibly conflicting user requests are reconciled against these resource policies. For many high-performance or adaptive applications, predictability is required in the resource environment to permit proper application function. The WS-Agreement agreement-based model helps provide such predictability, where different application managers (agreement initiators) are often simultaneously attempting to manage, by sharing, the same services or underlying primitive resources (maintained by agreement providers). The WS-Agreement model uses agreement negotiation to capture the notion of dynamically adjusting policies that affect the service environment without necessarily exposing the details necessary to enact or enforce the policies. Each agreement represents a well-understood policy, capturing a mutual understanding of (future) service provider behavior held by the agreement initiator and agreement provider.

Ideally, the agreement model abstracts the impact of other users, agreement initiators, and service implementation details so that the bilateral agreement between the initiator and provider is understood without requiring initiator knowledge of existing workloads, the provider's proprietary operating policies, or the resource infrastructure "backing" the provider's promises. However, we intend for the agreement model of WS-Agreement to be applicable even in more concrete scenarios where the initiator does become aware of some of these details behind the agreement. WS-Agreement management scenarios include a wide range of lifecycle possibilities, from long-term management interactions that control the entire lifecycle of the provided service to short-term management interactions that temporarily modify the behavior of existing services. Examples of management effects include, but are not limited to, deploying new services and adjusting access policies, resource consumption, or performance goals (i.e. provisioning of resources to services). OGSI provides basic mechanisms for monitoring services, but lacks specific solutions for management effects as proposed here with WS-Agreement.

## *3.2 The Granularity of Agreement: Agreement Services*

Each agreement represents a stable, named representation of promised service behavior where many volatile details are abstracted out to simplify the presentation of predictable or expected behaviors in the terms of the agreement negotiation. The underlying enactment of these agreements, in turn, may effect the creation of new domain-specific processes and/or services and may also adjust the behavior of newly created (or preexisting) processes or services.

First-class, OGSI-compliant agreement services permit us to exploit basic OGSI capabilities in order to monitor and manage the lifecycle of agreement. This not only simplifies the mechanisms needed in order to participate in agreement negotiation, but also makes more explicit a stateful semantics for every agreement. Agreement formation is represented as creation of an agreement service, and agreement cancellation is represented by the destruction of such a service. By representing each separately-negotiated bilateral agreement as a service, this approach faithfully decomposes the accumulative changes in agreed-upon policy held by an entity capable of forming agreements with multiple parties. Agreements that are no longer capable of affecting present or future resource behavior can be safely terminated (and reported to logging or audit services), though obligations may still be in place regarding cost or penalties for past service delivery.

### *3.3 Negotiation is a Stateful Dialogue*

Negotiation does not always necessitate complex multi-trip communications, but rather may be as simple as a single request message being allowed (or not) by policy. Of course, one can imagine much more complicated negotiation scenarios where the policies and intermediate commitments of the two parties are revealed piece by piece over a long sequence of message exchanges, resulting in an agreement capturing an intersection in their policies. We define an Agreement service *creation* primitive which can directly yield effective negotiated policies, either from the base policy environment of the parties or in explicit reference to other negotiated Agreements. We also define an optional, fine-grained *renegotiation* mechanism to operate on individual terms of an existing Agreement service, where appropriate for the term semantics and policies of the negotiating parties.

This approach acknowledges the asynchronous consistency problems inherent in large-scale Service Management: the frequency and importance of different term (re)negotiations are affected both by term semantics and application models. Different management scenarios will have to weigh the benefits and costs of various client- and server-side fault-handling strategies. Negotiation of dependent Agreements allows flexible soft-state management models with full introspection and monitoring capability on important aspects of a negotiated policy environment. Term-level renegotiation allows light-weight negotiation at the cost of reduced naming, lifecycle and monitoring capability.

Using abstracted terms in a preliminary agreement (e.g. computer reservation terms in anticipation of an upcoming job) the initiator may gain initial commitment (the reservation) which are then utilized in support of further dependent agreements (the job submission). Renegotiation is appropriate to refine an existing Agreement (e.g. to change the amount of physical memory available to a running job). Both of these negotiation mechanisms establish a commitment modeled as terms in an Agreement service. But not all negotiation attempts will result in commitments being made. Fault responses support zero-commitment patterns in which the initiator learns that its request terms are unacceptable.

In WS-Agreement, all commitments are modeled as terms in Agreement services. Other information or terms outside of an Agreement service are informational, as in the fault case or in advertisements that an agreement provider might publish.

## 4 Agreement-based Service Management Model

An agreement provider is instantiated as an OGSI-compliant AgreementFactory service. An Agreement service represents the result of a successful negotiation between an initiator and the agreement provider AgreementFactory. Agreement services MAY also represent policies formed either out of band from WS-Agreement mechanisms or unilaterally to represent progress or status of other already negotiated agreements. In such cases, no AgreementFactory is involved.

An Agreement service, through its extensible content, SHOULD always relate to a *delivered service* behavior which MAY involve a Grid service. The Agreement service MAY relate to an existing service known by the agreement provider. In this case, the Agreement represents an aspect of policy affecting the behavior of that service. Alternatively, the Agreement service MAY relate to a new service which will be created due to the agreement. In this case, the Agreement represents, on the part of the agreement provider, both a commitment to create the new service and policy affecting the behavior of the new service.

## *4.1 Agreement Meaning and States*

With respect to its terms, a created Agreement service can be in one of two states: *satisfied* (when all its terms have been met or are being met), or *violated* (when at least some of its terms are not being met). Extended, domain-specific terms of agreement MUST define a semantics for satisfaction [SNAP]. Agreement terms SHOULD denote unambiguously what is necessary to satisfy an agreement. Even in proper, well-defined circumstances, differences in observation of the service provider state MAY lead the parties to draw differing conclusions. However, with sufficient trusted auditing mechanisms, such differences in observation SHOULD converge to a bilateral understanding of satisfaction, adequate for resolving conflicts by compensation or remuneration. An agreement MAY be considered to be satisfied or violated according to the agreement provider, and this status SHOULD be reported as understood through SDEs in the Agreement or AgreementFactory services. This reporting of status is not sufficient for monitoring or enforcement unless the initiator trusts the reporting of the agreement provider. However, it is a useful model for framing provider communication to the initiator, for example to indicate *planned violation*, where the provider can signal violation before the service behavior deviates from the requirements of the terms, or to support diagnosis where provider and initiator views of status might be compared.

A special form of satisfaction is *completion*, a final state in which the agreement provider has no more obligations to fulfill as part of the agreement terms. For example, an agreement to execute a job or transfer a data file may be completed. This status is useful to model because it defines an important event or state-change boundary around which other services or processes might be synchronized.

With respect to provider activity, some agreements may have periods of time where they are *inactive*, which means that the agreement provider does not have to do anything to satisfy the terms, and other times where they are *active* and the provider is doing something to meaningfully satisfy terms. This notion of "doing something" is rather informal and MAY be delineated differently in each extended agreement term language. Some agreement term languages MAY avoid the notion of inactivity altogether.

With respect to negotiation status, a created agreement can be in one of two states: an agreement can be *observed*, which means that all the agreement terms are being observed (see section 4.3.1), and the agreement is ready to enter an active period, or an agreement can be *considered*, which means that the terms of the agreement are still under negotiation and the agreement as a whole is not ready to be observed. A considered agreement may not become active.

### 4.1.1 Directionality of Agreements

The terms in an Agreement represent commitments made by the agreement provider. These commitments MAY include aspects such as service cost which are commonly thought to be bidirectional commitments. In the WS-Agreement model, such terms are considered to be the "provider's half" of the commitment. If the parties are not mutually trusting, the provider

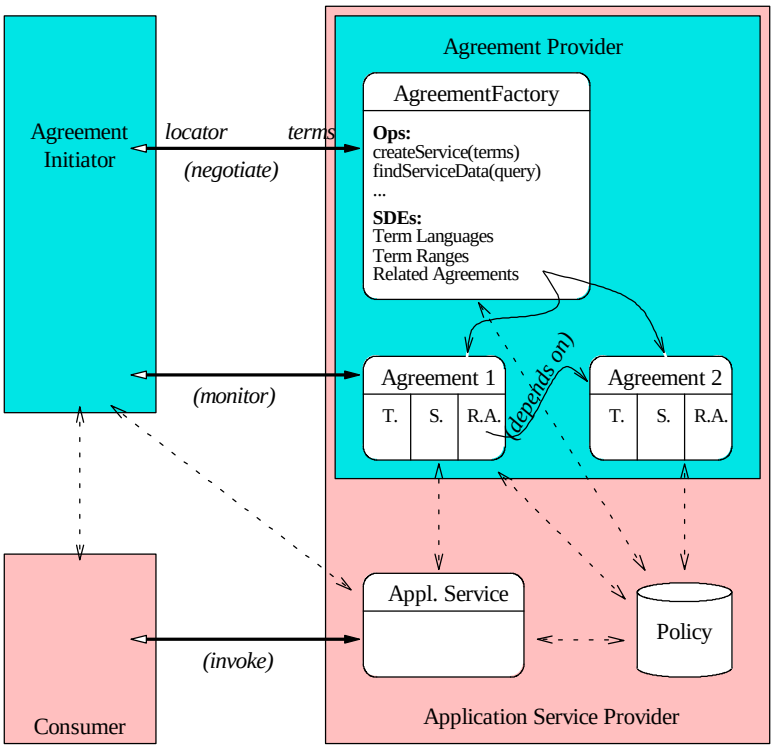


**Figure 2: The WS-Agreement model uses OGSI service creation as the negotiation primitive, by providing an AgreementFactory interface to agreement providers, inheriting from the ogsi:Factory portType. To support more complex negotiations, WS-Agreement allows agreements to be linked by extensible relationships. In this example, Agreement 1 *depends* on Agreement 2; a common scenario where this might occur is with advance reservation. Agreement 2 represents an advance reservation (future promise) of service capability, and Agreement 1 is an agreement for service exploiting that advance reservation. This scenario illustrates an implementation strategy (for the choice introduced in Figure 1) of tightly integrating the agreement and application service provider functions, where both behaviors share the same policy store.**

SHOULD, when necessary due to local policy or a naturally bidirectional commitment semantics, obtain commitment from the initiator by some other (unspecified) means.

Such means MAY include reversing the roles and using full WS-Agreement facilities to initiate a complementary Agreement with an agreement provider representing the interests of the original client. In practice for many traditional commitments such as payment transactions, existing non-WS-Agreement mechanisms SHOULD be used, such as debiting a resource-specific accounting system or otherwise forming a domain-specific agreement outside the WS-Agreement model. For example, if an WS-Agreement Agreement has a term indicating a charge to be applied to the initiator's credit-card, the agreement provider might interact with the credit-card company or some other online billing system to get commitment or proof of payment before enacting the service behaviors embodied in the Agreement service terms.

## *4.2 Agreement Lifecycle*

The Agreement service has a lifecycle from creation through termination. At creation, any non-negotiable terms are in effect for (or *observed by*) the agreement provider and additional renegotiable terms MAY be in place or new ones added. Terms MAY bear annotations which restrict their relevance to certain periods of time inside or outside the Agreement lifetime, but terms MUST NOT be in effect beyond the termination of the Agreement.

### 4.2.1 Agreement Creation

The WS-Agreement model defines two essential portTypes: the wsa:Agreement service and the wsa:AgreementFactory service. The wsa:AgreementFactory inherits the ogsi:Factory interface and extends it to support the creation of wsa:Agreement services. An agreement initiator is a client of the wsa:AgreementFactory who negotiates agreements by invoking the ogsi:createService operation with appropriate argument content. This operation can yield a fault if some of the required terms are rejected, or can result in the creation of a considered or observed agreement. The parties may require further negotiation to bring the agreement into an observed state after which it can be acted upon.

### 4.2.2 Agreement Lifetime

During its lifetime an agreement may take on states defined in section 4.1. Specifically, further negotiation or renegotiation can be used to change the state of a considered agreement to that of an observed agreement. Either considered or observed agreement can be renegotiated within the framework determined by the properties of agreement terms (see section 4.3.1).

Agreements can be monitored during their lifetime. As described above, the agreement terms SHOULD describe measurable goals or requirements on service provider behavior. Either the agreement initiator or agreement provider MAY be able to observe the service provider behavior and compare it to these terms. It some circumstances, it may be useful for the agreement provider to communicate its observations of service provider behavior to the agreement initiator or other clients. Extended agreement term languages MAY provide a means for the agreement initiator to request that certain measurable behavior be reported back as service data values in the Agreement service, for example using the monitoring criterium concept from Section 4.3.5. In addition, the agreement provider MAY export measured service behaviors even when not explicitly required by the agreement terms. The preceding violation and activity status concepts are generally useful instances of this sort of extensible observation reporting.

The Agreement SHOULD expose service data providing introspection on the Agreement itself, e.g. visibility of negotiated terms and ongoing renegotiation. By monitoring the state of the Agreement, an initiator MAY be informed of renegotiable term status changes and be able to adapt to changing circumstances before any change in delivered service behavior is actually observed.

### 4.2.3 Agreement Termination

The initiator SHOULD utilize service termination to conveniently cancel all terms of commitment the provider is observing as part of the Agreement. The provider MAY use service termination to signal an unwillingness or inability to satisfy its commitments or support renegotiation of its terms. However, the agreement provider SHOULD continue to host the Agreement service and represent a violation status on fatal violations in preference to terminating the service, as this provides a more capable monitoring interface for clients. Similarly, specializations of the wsa:Agreement portType MAY define additional operations with which the client MAY deactivate the agreement or individual terms without terminating the service. Such augmentation of the agreement lifecycle benefits complex negotiation scenarios where retained agreement state may be useful to avoid “from scratch” negotiation after violation or deactivation periods. Termination of the Agreement service allows the provider to reclaim any resources which might have been reserved or applied in support of the agreement. It does not necessarily release either party from any costs and/or penalties implied by the Agreement terms. Depending on the semantics of the extensible agreement term language and negotiated terms, agreement initiators and/or agreement providers MAY incur costs for premature agreement termination.

By analogy to business scenarios, either party could be in breach of contract at any time. This means that the provider does not have to provide the agreed upon service, but the party in breach likely will be penalized in some way. Continuing with this analogy, any attempts to mitigate the effects of being in breach, such as delivering some service, but at less than agreed to level, or delivering the service late, MAY result in less severe penalties and lower overall "cost," depending on the extensible term semantics in use. Termination of an Agreement service by the agreement provider is the most extreme signaling mechanism for agreement violation in WS-Agreement and SHOULD be an action of last resort. Conversely, termination of an Agreement service by the agreement initiator is a companion operation to agreement creation and SHOULD be used to signal a loss of interest in the services negotiated in the agreement creation.

### 4.2.4 Agreement Audit and Finalization

The monitoring terms suggested in Section **Error! Reference source not found.** control what observed service behaviors are reported by the agreement provider as part of the Agreement service. There is also a need to negotiate behavior of the Agreement service monitoring with respect to termination. Term extensions MAY be introduced to control how Agreement service state is delivered to audit systems, including possible archiving or accounting of Agreement service lifetime, terms, provided service behaviors, and reason for termination.

## *4.3 Extensible Agreement Language*

An invocation to an AgreementFactory for creation of an Agreement service carries an agreement document containing a set of extensible terms, resulting in the formation of an agreement between the requester and the service provider, and embodiment of this agreement as an Agreement service. We define an agreement document type and a set of agreement term types that MUST be used both in creation of an agreement service and in expressing this agreement in an agreement document. Externally defined agreement languages MAY be used to extend WS-Agreement behaviors.

### 4.3.1 wsa:AgreementType

An agreement document MUST be of the type wsa:AgreementType, which is extension of the WS-Policy [WS-Policy] wsp:PolicyExpression type.

```
<xsd:complexType name=”AgreementType”>
   <xsd:complexContent>
      <xsd:extension base=”wsp:PolicyExpression”>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name=”agremeent” type=”AgreementType” />
```

The term elements within an agreement document SHOULD be either one of the standard WS-Policy Compositors (wsp:OneOrMore, wsp:All, wsp:ExactlyOnce, wsp:Reference), or an element that extends the wsa:TermType.

### 4.3.2 wsa:TermType

Agreement terms are an extensible and interoperable means to author and convey the nature of an agreement. These terms can be used to convey meaning during a negotiation exchange or afterwards to characterize an existing agreement. Not all agreement terms have the same obligations to be met. There are some agreement terms that are *negotiable*, and others that are

*non-negotiable*,meaning that the expressing party is unable or unwilling to consider alternatives for the term. Some agreement terms may be *required* while others may be *optional*. Once agreed to by both parties, terms are considered to be *observed* with the exception of optional terms which can be *ignored* by the provider. These status concepts are common to all agreement terms and if expressed in a common way, can facilitate the understanding of the agreement and what is required to satisfy it.

The base type of all agreement terms in WS-Agreement MUST be wsa:TermType, which is an extension of the WS-Policy wsp:Compositor type. This base type uses the wsp:Usage attribute for specifying if a term is Required, Optional, Observed or Ignored. It also uses the <wsp:All>, <wsp:OneOrMore>, and <wsp:ExactlyOne> operators to group terms. Finally it uses the wsp:Preference attribute to specify preferences when groups have one or more selectable terms. A well defined agreement MAY include terms on the service guarantees, such as service level goals to be supported by the provider [Ludwig SLA], importance of this term and/or business impact for violating these terms, as well as terms on the manageability of this agreement, such as termination time, monitoring criteria, and termination criteria.

The wsp:Usage and negotiability attributes of an agreement term define the status of this agreement term. The wsp:Usage status specifies whether a term is agreed by both the client and the provider (i.e., wsp:Observed), whether the term is required by the client or provider (i.e., wsp:Required), or whether the term is optional to the client or ignored by the provider (i.e., wsp:Optional or wsp:Ignored). The negotiability attribute specifies whether the term is fixed (i.e., required by client or provider, or agreed by both), or negotiable.

#### 4.3.2.1 Term State Definitions for wsp:Usage

**wsp:Required** The term is required but no agreement has been reached on it yet. The following actions MUST be taken based on who received the required term:

**Provider:** If the term is received by a provider as a creation parameter and cannot be met, the agreement provider MUST return a fault. If the agreement term is acceptable to the provider then the provider MUST change the wsp:Usage value to wsp:Observed to indicate its acceptance. It may also need to select a value for the term if the term was negotiable.

**Initiator:** The term can be received by an initiator in on of two ways:

As the result of a fault: This specifies terms that the provider could not accept and indicates the new values the provider can accept. The initiator could use this information to formulate a new creation request if they feel they can accept the new terms.

As additional terms after an agreement has been created: The initiator MUST change the required terms to wsp:Observed before the agreement can be used. If the initiator cannot agree to the terms they SHOULD call ogsi:Destroy on the Agreement service.

**wsp:Observed** The term has been accepted by both parties. Terms can transition their state to wsp:Observered from wsp:Required or wsp:Optional. This is true for terms specified by either the initiator or the provider.

Terms that have wsp:Usage=wsp:Observed and

wsa:Negotiability=wsa:Negotiable can still be renegotiated even though they are accepted.

**wsp:Optional** The Term is desired but no agreement has been reached on it yet. An optional term implies the agreement term could be ignored. The following actions MUST be taken based on who received the optional term:

**Provider:** If the term is received by a provider as a creation parameter and cannot be met, the agreement provider MUST change the wsp:Usage value to wsp:Ignored to indicate it is not part of the agreement. If the agreement term is acceptable to the provider then the provider MAY change the wsp:Usage value to wsp:Observed to indicate its acceptance. Alternately the provider MAY keep the state as wsp:Optional to indicate it wants to keep it as an option.

**Initiator:** The term can be received by an initiator in on of two ways:

As the result of a fault: This specifies terms that the provider considers optional and the initiator MAY choose to accept by passing it back it as wsp:Required as part of the creation parameters on the next request.

As additional terms after an agreement has been created: The initiator MAY change the required terms to wsp:Observed indicating they want to accept the optional term or they MAY change it to wsp:Ignored to not include it in the agreement. If the term is left as wsp:Optional, the provider has the option to honor that term or not.

**wsp:Ignored** The term is not being used. Only terms that were wsp:Optional can transition to the wsp:Ignored state.

#### 4.3.2.2 Term State Definitions for wsa:Negotiability

The table below illustrates how the wsp:Usage states relate to the wsa:Negotiability states:

| **wsa:Negotiability State Combinations** | | |
|---|---|---|
| | **wsa:Fixed** | **wsa:Negotiable** |
| **wsp:Required** | The term is required but no agreement has been reached on it yet. The contents of the term cannot be changed and MUST be agreed to (i.e., changed to wsp:Observed) for the agreement to be valid. | The term is required but no agreement has been reached on it yet. The contents of the term can be changed within the range of specified negotiable parameters and MUST be agreed to (i.e., changed to wsp:Observed) for the agreement to be valid. |
| **wsp:Observed** | There is an agreement to provide the term. The contents of the term cannot be changed. | There is an agreement to provide the term. The contents of the term can still be renegotiated. |

| | | |
|---|---|---|
| **wsp:Optional** | The term is desired but no agreement has been reached on it yet. The contents of the term cannot be changed but the term MAY be ignored. If the term is not ignored it MUST be agreed to (i.e., changed to wsp:Observed) for the agreement to be valid.. | The term is desired but no agreement has been reached on it yet. The contents of the term can be changed within the range of specified negotiable parameters but the term MAY be ignored. If the term is not ignored it MUST be agreed to (i.e., changed to wsp:Observed) for the agreement to be valid.. |
| **wsp:Ignored** | The term is not being used. | The term is not being used. |

```
<xsd:complexType name=”TermType” abstract=”true” >
   <xsd:attributeGroup ref="wsp:CompositorAndAssertionAttributes" />
   <xsd:attribute name=”Name” type=”xsd:NCName” />
   <xsd:attribute name=”Negotiability” type=”wsa:NegotiabilityType”/>
</xsd:complexType>

<xsd:simpleType name="NegotiabilityType">
   <xsd:restriction base="xsd:QName">
      <xsd:enumeration value="wsa:Fixed"/>
      <xsd:enumeration value="wsa:Negotiable"/>
   </xsd:restriction>
</xsd:simpleType>
```

By modeling agreement terms as WS-Policy assertions, more complex term associations can be expressed. The use of wsp:Preference in conjunction with <wsp:All>, <wsp:ExactlyOne>, and <wsp:OneOrMore> can express groups of terms which represent choices to select from.

The following example illustrates the use of these WS-Policy artifacts:

```
<wsp:Policy>
   <wsp:ExactlyOne wsp:Usage="wsp:Required">
      <ns:term1 wsp:Preference="1" wsa:Negotiability="wsa:Fixed">
         ...
      </ns:term1>
      <ns:term2 wsp:Preference="100" wsa:Negotiability="wsa:Fixed">
         ...
      </n2:term2>
   <wsp:ExactlyOne>
</wsp:Policy>
```

The example above, when specified by a requestor, specifies two term which are not negotiable (wsa:Negotiability="wsa:Fixed") but are a disjunction in that the provider only has to satisfy one of them (<wsp:ExactlyOne>) and the requestor prefers that the provider satisfy the second one (wsp:Preference="100") if possible. Combinations of these WS-Policy assertion tags, can be used to express agreements containing complex conjunctions and disjunctions of terms.

### 4.3.3 wsa:ContextType

An agreement is scoped by its associated context that SHOULD include parties to an agreement, and additionally, SHOULD include reference to the service provided in support of the agreement. The context MAY also include other prior and/or related agreements. The new agreement thus augments prior related agreements, between the client and the service provider. The agreement terms SHOULD be associated with the context via the WS-PolicyAttachment mechanisms [WS-PolicyAttachment], unless the appropriate association for a term is implied by semantics.

The base type wsa:ContextType defines parties to this agreement, i.e. the agreement initiator and agreement provider. The URI for an agreement provider MAY be a Grid Service Handle (GSH) [OGSI] to an existing service. However, the requester and provider URIs MAY instead identify the parties by more abstract naming, e.g. by security identity of the owner or operator. The context also defines references to the provided service for which the agreement terms are defined. Additionally, the context defines any number of related agreements that define existing agreement terms which are being augmented via this agreement. The related agreements are represented in the agreement service as related agreement services (see Section 4.6).

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

```
<xsd:complexType name="ContextType">
 <xsd:complexcontent>
  <xsd:extension  base="wsa:TermType"
   <xsd:sequence>
      <xsd:element name="agreementInitiator" type="xsd:anyURI" />
      <xsd:element name="agreementProvider" type="xsd:anyURI" />
      <xsd:element name="service" type="xsd:anyURI"/>
      <xsd:element name="relatedAgreements"
                   type="ogsi:LocatorType"
                   minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
   <xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="context" type="wsa:ContextType" />
```

### 4.3.4 Agreement Lifetime

Negotiation of the lifetime of Agreement services is important to the WS-Agreement model. However, OGSI defines a negotiation model for Grid services and we are able to use this without change. The ogsi:Factory portType defines an input message to initialize the lifetime of new services, and the ogsi:GridService portType defines an SDE and negotiation operations to represent and update the scheduled lifetime of the service. Agreement initiators MAY use this mechanism to negotiate Agreement service lifetime. Extended negotiation languages MAY define other mechanisms to negotiate lifetime integrated with other negotiation terms. The resulting negotiated lifetime MUST be exposed as ogsi:TerminationTime and further negotiation MUST be possible through the basic OGSI mechanisms.

### 4.3.5 wsa:MonitoringCriteriumType

The base type wsa:MonitoringCriteriumType can be extended to define criteria for monitoring agreement terms. A set of well defined agreement monitoring criteria MAY include both the set of terms to be monitored by the client, as well as monitoring details, such as how often the monitoring data can be received by the client (without causing undue burden on the provider) or the details on receiving audit data after the agreement service becomes unavailable. The set of values to be monitored MAY include status with respect to agreement terms (say, violation of a response time goal) as well as values over which the guarantees are expressed (say, response time itself). The values to be monitored dynamically becomes available as monitoredValue service data (see section 6.2.7).

```
<xsd:complexType name="MonitoringCriteriumType"
                      abstract="true" >
   <xsd:complexcontent>
      <xsd:extension base="wsa:TermType"
      <xsd:extension>
   </xsd:complexContent
</xsd:complexType>
```

### 4.3.6 wsa:TerminationCriteriumType

The base type TerminationCriteriumType is an abstract type that can be extended to define a criterium for termination of agreement and details of finalization process. A well defined agreement termination criterium may include who can terminate this agreement and under what conditions the agreement may be terminated (say, multiple violations within a time window, client exceeding an agreed upon request rate, presence of a higher valued agreement, etc ). Note that both the client as well as provider MAY initiate an early termination. The termination criteria MAY also include penalty assessment on the party responsible for this termination.

```
<xsd:complexType name="TerminationCriteriumType"
                      abstract="true" >
   <xsd:complexcontent>
      <xsd:extension  base="wsa:TermType"
      <xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

## *4.4 Agreement Negotiation*

The WS-Agreement model supports complex negotiation scenarios through two mechanisms: (1) creation of dependent agreements by repeated application of the extensible ogsi:createService operation of the AgreementFactory, and (2) a Renegotiation operation to manipulate terms of an Agreement marked with the wsp:Negotiable property. Fault responses from either operation signal agreement provider rejection of initiator terms. If createService faults, there is no new Agreement. If Renegotiate fails, the existing Agreement terms are unchanged. Successful responses signal acceptance of initiator terms as described below. By repeatedly invoking the createService or Renegotiate operations, a stateful client may discover acceptable terms. However, the agreement provider should publish, as SDEs, information about acceptable terms whenever possible.

### 4.4.1 Creation Parameters

The model of agreement places the following requirements on the behavior of an ogsi:createService invocation, beyond the basic requirements from the ogsi:Factory definition:

The input CreationParameters MUST bear an element of type wsp:AgreementType, and which SHOULD match one of the supported agreements advertised by the AgreementFactory service in the supportedAgreement SDE.

The following applies to terms within this agreement:

- For each term not bearing the wsp:Usage attribute, a default of wsp:Usage=”wsp:Required” MUST be assumed.
- For each term not bearing the wsa:Negotiability attribute, a default of wsa:Negotiability= "wsa:Fixed” MUST be assumed.

### 4.4.2 Creation Faults

Extensible faults can communicate reasons behind failures to negotiate an agreement. With a fault response to the AgreementFactory createService invocation, the agreement initiator knows that no new agreement has been created. Faults MAY expose information not easily generalized (or computed) without detailed negotiation terms, and faults MAY expose information not generally accessible through SDEs due to complex provider policies. Faults MAY include elements with wsp:Usage=”wsp:Required” values. Elements MUST NOT appear in faults with the wsp:Usage=”wsp:Observed” value.

The expected response of a client to faults is to: retry the negotiation after transient failures, reformulate the negotiation in response to fault information, abandon the agreement provider for an alternate choice, etc.

### 4.4.3 Created Agreement

An ogsi:createService invocation on an AgreementFactory MAY yield the locator of an Agreement service whose terms satisfy the agreement terms specified in the CreationParameters supplied by the client. These terms MAY represent either a final committed agreement between the parties or intermediate negotiable terms, and the overall status is expressed in the status SDE as well as returned via extensible output from the createService invocation. Extended languages SHOULD use the wsp:Usage and wsa:Negotiability attributes defined in Section 4.3.1 to distinguish these situations wherever possible.

### 4.4.4 Renegotiation

In the created Agreement service, any input terms which were marked wsp:Required MUST appear as wsp:Observed. However, the provider MAY introduce additional terms with wsp:Required for which additional negotiation is necessary. Until all terms are marked wsp:Observed, the Agreement itself is not observable and its status SDE MUST NOT bear the wsp:Observed status.

Additionally, terms marked wsp:Negotiable MAY continue to be adjusted even after the Agreement is observed. The extensible content to the wsa:Renegotiate operation selects a negotiation semantics for how the Agreement terms change. During renegotiation, the Agreement MAY cease to be observed and the Agreement MAY atomically change from one observed state to another, depending on the negotiation semantics at work.

## *4.5 Service Relationships*

There are many interesting relationships that can be formed between services, of which *agreement composition* (described below) is a primary concept. Some of the agreement relationships discussed in this specification represent important behavioral links between

Agreement services, while other relationships represent more abstract conceptual models, e.g. a relationship to capture that an AgreementFactory service *provides* an Agreement service.

The relationships between Agreement services may be dynamically changed during the agreement lifecycle. Therefore, the WS-Agreement model does not distinguish different structural relationships at the portType level. Instead, every WS-Agreement service has the capability to expose rich, dynamic relationships to other services. We achieve this flexibility by inheriting the ogsi:ServiceGroup portType and defining ServiceGroup *entry content* to characterize the relationship of member services to the service presenting the member in a ServiceGroup entry.

The general interpretation of a member agreement service with a RelatedAgreementType content, is that the member service is a "related agreement of the containing service." For example, if the content is ProvidedAgreementType, then the member is a "provided agreement of the containing (factory) service."

The member service in a ServiceGroup entry containing content of type RelatedAgreementType MUST implement the Agreement portType (this affects all relationships presented below).

Due to the nature of the complex, extensible agreement terms, agreements may be semantically related in ways not exposed as relationships.

```
TBD: RelatedAgreementType here as an abstract type
```

### 4.5.1 Agreement Dependencies

Dependent Agreement services are asymmetrically related to another Agreement service (the dependency). If the dependency Agreement service is terminated or goes into a violation, the dependent Agreement services might be expected to terminate or go into violation unless compensating actions are taken. Directed (and complementary) DependentAgreementType and DependencyAgreementType relationships allow both services to express their relationship to the other in their ServiceGroup entry contents. The dependency Agreement service MAY hold an entry with the dependent Agreement instance as the member and the DependentAgreementType relationship as content. The dependent Agreement service MAY hold an entry with the dependency Agreement service as the member and the DependencyAgreementType relationship as content.

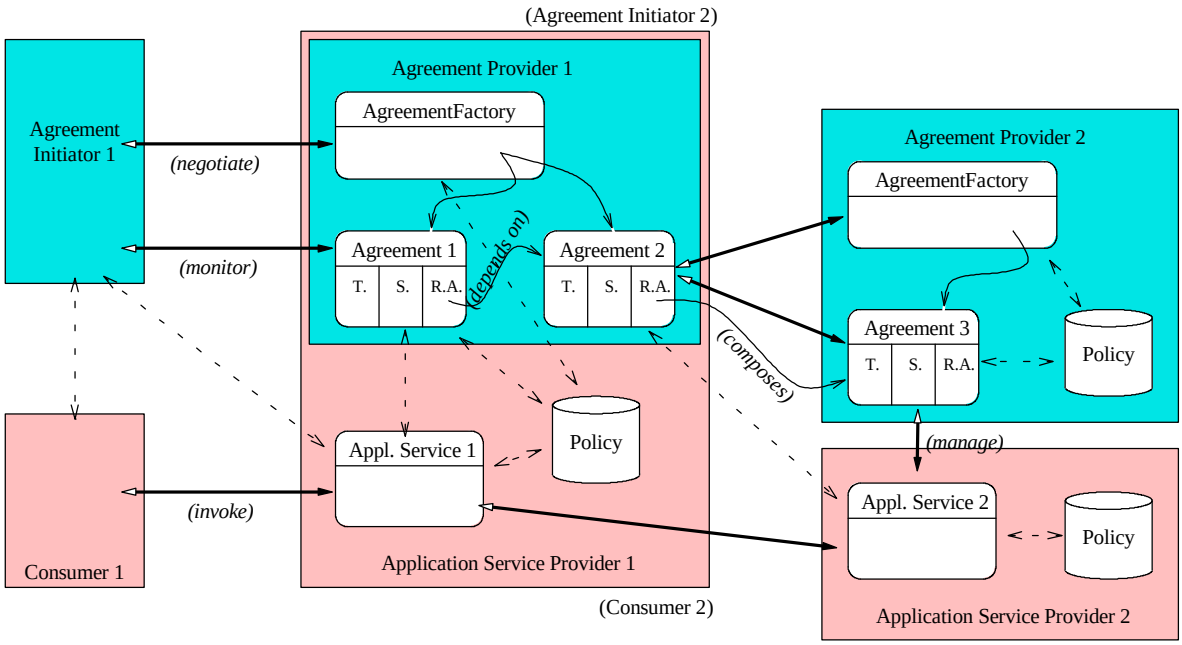


**Figure 3: Agreement virtualization supports composed application scenarios. This figure extends the scenario from Figure 2 such that the Application Service 1 implementation depends on Application Service 2 at runtime. To implement the promise for service represented by Agreement 2, Agreement Provider 1 negotiates with Agreement Provider 2 for required capacity of Application Service 2. This figure also illustrates another implementation strategy in Agreement Provider 2 versus that of Agreement Provider 1. Agreement Provider 2 is not integrated with Application Service Provider 2 but instead must *adapt* to an underlying management interface.**

```
TBD: DependentAgreementType here derived from RelatedAgreementType.

TBD: DependencyAgreementType here derived from RelatedAgreementType.
```

### 4.5.2 Agreement Composition

Agreement composition provides a coordinated interface to multiple Agreement services. The *composite* Agreement service groups together a set of *component* Agreement services, such that the meaning of the composite agreement is derived from the meaning of the components and vice versa. For example, a job-execution agreement might compose a computer allocation agreement and a software-license allocation agreement. In such situations, agreement status and lifecycle are closely linked:

- Termination of a composite agreement MAY imply termination of component agreements.
- Status changes in component agreements SHOULD be reflected through status changes in the composite agreement when that is implied by the meaning of the composite. Violation or termination of a component agreement MAY imply violation of the composite agreement.

The composition relationship is not constructed by any single operation, but may result from a number of different operational scenarios:

- *Decomposition for enactment*: the client could negotiate complex agreement terms that the provider satisfies by coordinating multiple *enacting* agreements. This process can be

opened to introspection by expressing the client-initiated Agreement service as a composite agreement, with each provider-initiated agreement appearing as a component.

- *Decomposition by restriction*: an Agreement could decompose into one or more *restricted* Agreement services, e.g. a computational resource agreement could be decomposed into space-sharing or time-sharing components targeted for particular purposes. Such restriction MAY define component agreements capturing only a subset of the original agreement meaning, with a complementary or residual commitment implied in the original (newly composite) agreement.
- *Aggregation*: a lightweight composite Agreement could be constructed to gather several Agreement services as components for convenient monitoring or management.

In the primary, negotiated form of these scenarios, the composite relationship occurs as a result of explicit terms specified by the agreement initiator. For example, a client might negotiate a complex workflow agreement, including fine-grained monitoring requirements that require elements of the workflow to be exposed as component agreements. Similarly, the client might request an aggregate job-set interface to a set of simple job agreements in order to simplify monitoring or lifecycle management. More interestingly, the client might explicitly negotiate both composite workflow and component job agreements to intervene in the planning process and refine the agreements with specialized intelligence, in which case the agreement provider performs the role of an *enactment service*.

In the secondary, introspective form of these scenarios, the agreement provider may unilaterally expose compositional relationships to support observation of implementation choices it has made in satisfaction of a negotiated agreement. For example, the provider might instantiate job agreements to expose its progress in delivering a negotiated workflow. This also might occur for complex workflows where the provider is publicizing process information beyond the negotiated monitoring requirements. Extensive, hybrid forms can be imagined in which a combination of negotiated and introspective compositions are constructed.

```
TBD: CompositeAgreementType here derived from RelatedAgreementType.

TBD: ComponentAgreementType here derived from RelatedAgreementType.
```

### 4.5.3 Provided Agreement

An AgreementFactory service can express its relationship to Agreement services that it *provides* (that it created in the role of the agreement provider). The complementary provided-by relationship is already captured in the base ogsi:GridService portType through the ogsi:factoryLocator SDE, so we do not replicate that information as a relationship.

- The service expressing ServiceGroup entry content of type ProvidedAgreementType MUST implement the AgreementFactory portType.
- The member service of a ServiceGroup entry with content of type ProvidedAgreementType SHOULD have its ogsi:factoryLocator SDE referring to the AgreementFactory service containing the entry.

```
TBD: ProvidedAgreementType here derived from RelatedAgreementType.
```

# 5 AgreementFactory PortType

The AgreementFactory portType represents a manageability interface for negotiating with an agreement provider. The AgreementFactory provides specialized Agreement service creation

capabilities by extending the ogsi:Factory behavior. Additionally, the AgreementFactory provides a discovery or aggregate management capability by specializing the ogsi:ServiceGroup behavior.

The AgreementFactory portType inherits ogsi:Factory and ogsi:ServiceGroup portTypes.

## *5.1 AgreementFactory: Meaning of Inherited Service Data Declarations*

Some service data are inherited from ogsi:Factory.

### 5.1.1 ogsi:createServiceExtensibility

The ogsi:createServiceExtensibility SDE is used to advertise the createService parameter language(s) supported by the Factory. This SDE MUST be the QName of an element of type wsa:AgreementType.

The AgreementFactory portType does not define an ogsi:createServiceExtensibility SDE. Instead, it is expected that portTypes that extend AgreementFactory will define this SDE.

Additional information about the supported agreement languages, such as the terms supported by the agreement languages, and dynamically supported agreement languages, can be discovered via the supportedAgreements SDE.

## *5.2 AgreementFactory: Service Data Declarations*

### 5.2.1 AgreementFactory:supportedAgreements

```
<sd:serviceData name="supportedAgreement"
     type="wsa:SupportedAgreementType"
     minOccurs="0" maxOccurs="unbounded"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>

<xsd:complexType name="SupportedTermType">
   <xsd:attribute name="qname" type="xsd:QName" />
   <xsd:sequence>
      <xsd:any namespace="##any"
               minOccurs="0" maxOccurs="unbounded" />
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SupportedAgreementType">
   <xsd:attribute name="qname" type="xsd:QName" />
   <xsd:sequence>
      <xsd:element name="term" type="SupportedTermType"
                   minOccurs="0" maxOccurs="unbounded">
      <xsd:any namespace="##any"
               minOccurs="0" maxOccurs="unbounded" />
   </xsd:sequence>
</xsd:complexType>
```

This SDE describes an agreement language that is supported by the AgreementFactory service. The supportedAgreement/qname attribute contains the QName of an XSD element of type wsa:AgreementType. The supportedAgreement/term element describes one terms supported by this agreement language, where the supportedAgreement/term/qname attribute contains the

QName of an XSD element of type wsa:TermType. Additional information about an agreement language or term MAY be carried via the extensibility elements of this SDE.

And example of this SDE is:

```
<wsa:supportedAgreement qname="job:jobSubmission">
   <wsa:term qname="job:executable"/>
   <wsa:term qname="job:arguments/>
</wsa:supportedAgreement>
```

## 5.3 AgreementFactory: Service Data Values

All AgreementFactory services MUST include the following among its static SDEs.

```
   <ogsi:membershipContentRule>
      <ogsi:memberInterface>wsa:Agreement</ogsi:memberInterface>
      <ogsi:content>wsa:RelatedAgreementType</ogsi:content>
   </ogsi:membershipContentRule>
```

Therefore, any member service (listed in an ogsi:entry SDE) that implement the AgreementFactory portType MUST have a corrseponding wsa:RelatedAgreementType entry content describing the relationship of the member agreement to the AgreementFactory service containing the ogsi:entry SDE.

## 5.4 AgreementFactory: Meaning of Inherited Operations

The use of the Factory::createService operation is so fundamental to the WS-Agreement negotiation model that it is worth reviewing here. The use of the operation in this specification is consistent with its definition in the OGSI specification, and WS-Agreement places one further normative restriction on the CreationParameters input element, as described below. This restriction MUST be adopted by any portType inheriting from AgreementFactory.

### 5.4.1 Factory::createService

**Input**

- *TerminationTime* (optional): As per the OGSI specification.
- *CreationParameters*: This parameter bears the agreement request in the extensible language. The client MUST provide a CreationParameter element containing one element conforming to the wsa:AgreementType type and this element SHOULD match one of the element declarations denoted by the supportedAgreement SDE values of the invoked AgreementFactory.

**Output**

- *Locator*: A locator to a newly created Agreement service.
- *CurrentTerminationTime*: As per the OGSI specification.
- *ExtensibilityOutput* (optional): As per the OGSI specification. If the provider is adding terms to the agreement, those terms MUST be specified in this extensibility output. A copy of the status SDE SHOULD be returned so the initiator can easily determine if the status of the returned agreement.

**Fault(s)**

- Some AgreementFactory specific faults? These faults SHOULD contain the terms which the provider could not agree to with suggested values that they could agree to.

- *Fault*: As per the OGSI specification.

### 5.5 AgreementFactory: Operations

None.

# 6 Agreement PortType

The Agreement portType represents a manageability interface for individually established policies. Through extensible content, the Agreement portType exposes the *terms* of agreement and the *status* of the agreement in domain-specific language. In addition, the Agreement portType exposes structural relationships between multiple Agreement services via serviceData and specialized use of the inherited ogsi:ServiceGroup portType from OGSI. Finally, basic monitoring and lifecycle management are supported through the inherited ogsi:GridService and optional ogsi:SubscriptionSource portTypes.

### 6.1 Agreement: Meaning of Inherited Service Data Declarations

Fundamental Agreement service capabilities are exposed through the ogsi:GridService portType:

- *Agreement lifetime management*, through ogsi:terminationTime
- *Agreement lifecycle monitoring*, through CMM
- *Agreement provider factory*, through ogsi:factoryLocator

Additionally, structured discovery is supported through the ogsi:ServiceGroup portType:

- *Related agreements*, through ogsi:entry

These important features are discussed below.

#### 6.1.1 GridService: terminationTime

The ogsi:terminationTime SDE, inherited from ogsi:GridService, defines the lifetime of any Grid service. For Agreement services, this lifetime bounds the relationship between the client and the agreement provider expressed in the agreement terms. Regardless of any temporal metadata captured in the extensible agreement terms, both the client and the agreement provider MUST assume the agreement relationship is terminated when the Agreement service terminates. In general, the Agreement service SHOULD support service termination requested by the agreement initiator, and MAY unilaterally initiate termination without client consent. Restriction of these capabilities MAY be captured in the semantics of domain-specific agreement term extensions.

#### 6.1.2 GridService: factoryLocator

The ogsi:factoryLocator SDE, inherited from ogsi:GridService, identifies the ogsi:Factory which created the Grid service. For Agreement services, this ogsi:Factory locator MUST identify an AgreementFactory service if it is not nil. A nil ogsi:factoryLocator signifies an Agreement service which was created outside the scope of the WS-Agreement AgreementFactory mechanisms; for such an Agreement, the agreement provider MAY not be able to support service termination or any other side-effect.

#### 6.1.3 ServiceGroup: entry

The ogsi:entry SDE, inherited from ogsi:ServiceGroup, represents relationships between this and other Agreement services. Each entry characterizes the relationship between this and one other Agreement service, with the following properties:

- The entry's ogsi:serviceGroupEntryLocator element SHOULD be nil, and management of the agreement relationships SHOULD occur through portTypes in this WS-Agreement specification.
- Entries MAY appear with additional ogsi:content elements not matching the agreementRelationship type defined in this specification. Clients SHOULD ignore such entries if they do not understand the meaning based on extensions defined in an auxiliary specification.
- Entries MAY appear with ogsi:memberServiceLocator elements referring to services not supporting the Agreement portType. Clients SHOULD ignore such entries if they do not understand the meaning based on extensions defined in an auxiliary specification.

## *6.2 Agreement: Service Data Declarations*

The following concepts are captured as service data of the Agreement portType:

- *Creation parameters*: The content specified by the agreement initiator.
- *Agreement terms*: The mutual understanding between the client (agreement initiator) and agreement provider are captured in extensible language. See the AgreementFactory portType for more information.
    - *MonitoringCriteria*: A subset of the terms describing provided service behavioral properties which should be made observable through the Agreement service.
- *Agreement status*: During the lifetime of the Agreement service, state changes may occur that are of interest to monitoring clients. The status of the agreement captures both an abstract state machine and extensible content related to the agreement terms.
    - *MonitoredValue*: Extensible content exposing observable service behavior as defined by the monitoring criteria.

### 6.2.1 Agreement:renegotiableTerm

```
<sd:serviceData name="renegotiableTerm"
     type="wsa:SupportedTermType"
     minOccurs="0" maxOccurs="unbounded"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

This SDE contains information about agreement terms that MAY be renegotiable within this agreement.

### 6.2.2 Agreement: agreement

The agreement SDE captures the current agreement terms between the two parties, whether observed, required, fixed, or negotiable.

```
<sd:serviceData name="agreement"
     type="wsa:AgreementType"
     minOccurs="1" maxOccurs="1"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

### 6.2.3 Agreement: status

What about state-machine stuff from GRAM, CRM or WSLA? Such content could go in the body of the status element, and additional SDEs may describe the states or state transition model.

The extensible state of the agreement.

```
<sd:serviceData name="status"
     type="wsa:statusType"
     minOccurs="1" maxOccurs="1"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

The type wsa:statusType provides for an extensible state model. The base type captures observability of the Agreement. If the status SDE indicates the wsp:Usage="wsp:Observed" value, the terms SDE document MUST be observable according to the WS-Policy observation model.

```
statusType base type
      wsp:Usage attribute
      xsd:anyElement body?
```

### 6.2.4 Agreement: monitoredValueName

The semantics of the terms SDE document MAY suggest or require the presence of a specialized *monitored value* SDE. Monitored values allow the dynamic description of the provided service as measured by the agreement provider. Additionally, the agreement provider MAY expose monitored values not specifically identified in the terms SDE. One can find the names of all such monitored value SDEs through the use of the monitoredValueName SDE.

```
<sd:serviceData name="monitoredValueName"
     type="xsd:QName"
     minOccurs="0" maxOccurs="unbounded"
     mutability="mutable"
     modifiable="false"
     nillable="false"/>
```

## *6.3 Agreement: Service Data Values*

All Agreement services MUST include the following among its constant SDEs.

```
<ogsi:membershipContentRule>
   <ogsi:memberInterface>wsa:Agreement</ogsi:memberInterface>
   <ogsi:content>wsa:RelatedAgreementType</ogsi:content>
</ogsi:membershipContentRule>
```

Therefore, any member service (listed in an ogsi:entry SDE) that implement the Agreement portType MUST have a corrseponding wsa:agreementRelationshipType entry content describing the relationship of the member agreement to the Agreement service containing the ogsi:entry SDE.

## *6.4 Agreement: Meaning of Inherited Operations*

There are many important and valuable operations inherited from the GridService portType. However, WS-Agreement does not apply any novel restrictions or interpretation to these operations except as described above. The OGSI specification should be consulted for information on these operations.

### *6.5 Agreement: Operations*

For more complex negotiation scenarios, the Agreement portType supports an operation to *renegotiate* terms of an Agreement.

#### 6.5.1 Agreement: renegotiate

TBD: An operation should be defined to support multi-phase negotiation of negotiable terms.

## 7 Security Considerations

This specification defines the abstract interaction between a Grid service agreement provider and clients of that service. While it is assumed that such interactions must be secured, the details of security are out of scope of this specification. Instead, security should be addressed in related specifications that define how the abstract interactions are bound to specific communication protocols, how service behaviors are specialized via policy-management interfaces, and how security features are delivered in specific programming environments.

## 8 Editor Information

Karl Czajkowski
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
Email: karlcz@isi.edu

Asit Dan
IBM
19 Skyline Drive
Hawthorne, NY 10532
Phone: 914-784-6677
Email: asit@us.ibm.com

John Rofrano
IBM
2455 South Road
Poughkeepsie, NY 12601
Email: rofrano@us.ibm.com

Steven Tuecke
Distributed Systems Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Phone: 630-252-8711
Email: tuecke@mcs.anl.gov

Ming Xu
Platform Computing
Email: ming@platform.com

## 9 Contributors

We gratefully acknowledge the contributions made to this specification by the following people: William Allcock, Kate Keahey, and TBD.

# 10 Acknowledgements

We are grateful to numerous colleagues for discussions on the topics covered in this document, in particular (in alphabetical order, with apologies to anybody we have missed) Ian Foster, Carl Kesselman, Robert Keraney, Miron Livny, Heiko Ludwig, Jeff Nick, Volker Sander, Ellen Stokes, and TBD.

# 11 References

## *11.1 Normative References*

[OGSI]

*Open Grid Services Infrastructure (OGSI) Version 1.0*, S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt, Editors. Global Grid Forum. draft-ggf-ogsi-gridservice-29.

[RFC 2119]

*Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, Author. Internet Engineering Task Force, RFC 2119, March 1997. Available at http://www.ietf.org/rfc/rfc2119.txt

[WS-Policy]

"Web Services Policy Framework" (WS-Policy), BEA, IBM, Microsoft, and SAP, November 2002 http://msdn.microsoft.com/ws/2002/12/Policy/

[WS-PolicyAttachment]

Web Services Policy Attachment (WS-PolicyAttachement), http://www.ibm.com/developerworks/library/ws-polatt/

## *11.2 Informative References*

[Globus Overview]

*Globus: A Toolkit-Based Grid Architecture*, I. Foster, C. Kesselman, Authors. In [Grid Book], 259-278.

[Grid Anatomy}

*The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, I. Foster, C. Kesselman, S. Tuecke, Authors. International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001. Available at http://www.globus.org/research/papers/anatomy.pdf

[Grid Physiology]

*The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, I. Foster, C. Kesselman, J. Nick, S. Tuecke, Authors. Globus Project, 2002. Available at http://www.globus.org/research/papers/ogsa.pdf

[Ludwig SLA]

*A Service Level Agreement Language for Dynamic Electronic Services*, H. Ludwig, A. Keller, A. Dan, R.P. King, R. Franck, Authors. Journal of Electronic Commerce Research, Volume 3, Issue 1&2, Kluwer Academic Publishers, March, 2003.

[SNAP]

*SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems.* K. Czajkowski, I. Foster, C. Kesselman, V. Sander, S. Tuecke, Authors.Lecture Notes in Computer Science, 2537:153-183, 2002.