Interoperability and Reuse with WS-Agreement
Problems for Discussion as of January 20, 2004

Alain Andrieux   Karl Czajkowski

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292 U.S.A.

Abstract

The GRAAP working group of the Global Grid Forum is drafting a specification for the management of resources and services using negotiated service level agreements in a Web services environment (WS-Agreement). This memo discusses ongoing design considerations for this activity, focusing on the desire to strike a balance between goals for flexibility, reusability, and interoperability of systems utilizing the WS-Agreement interface. If WS-Agreement services are to be discovered and utilized by clients in a large-scale environment, their extended negotiation capabilities, policies, and offered services must be available for search, inspection, and comparison. The GRAAP working group faces difficult design challenges to achieve these goals while utilizing Web services technologies for term and constraint languages*, negotiation messages, negotiator characterization, and negotiator discovery.*

1 Introduction

The GRAAP working group of the Global Grid Forum is drafting a specification for the management of resources and services using negotiated service level agreements in a Web services environment (WS-Agreement). The WS-Agreement architecture identifies roles of *initiator* and *provider* in the negotiation system, meaning to support a wide range of management scenarios in the Web [3]. This memo discusses ongoing design considerations for this activity, focusing on the desire to strike a balance between goals for flexibility, reusability, and interoperability of systems utilizing the WS-Agreement interface.

Reusability and interoperability in a large-scale environment, such as the global Web, require that components follow well defined interface conventions. However, reusability and flexibility for us also means a way for components to offer management (allocation and control) of new and novel services or resources not envisioned in the base WS-Agreement mechanisms. Balancing these goals suggests a two-level semantics, with stable and reusable mechanisms for performing negotiation and management while composing these concepts with domain-specific behavioral concepts of the underlying managed resources [3, 1].

1.1 Integration Concerns

These goals are further complicated by the desire to integrate WS-Agreement into the larger Web services environment to compose functionality. If WS-Agreement providers are to be discovered and utilized by initiators in a large-scale environment, their extended negotiation capabilities, policies, and offered services must be available for search, inspection, and comparison. The presence of extensions to negotiate domain-specific service behavior in one WS-Agreement provider, eg. space-sharing computational jobs with guaranteed processor time and memory capacity, distinguish that provider from others. Therefore, registries and other discovery mechanisms must assist clients in efficiently distinguishing providers based on such extensions.

One approach to discovery is to explicitly name each extension and augment the "type" of the provider with this name and reuse existing type-based discovery mechanisms. However, such an approach does not help distinguish individual providers who offer the same extensions with different policies as to how the extensions can be negotiated, eg. up to 128 processors for 12 hours. Supporting such policy data requires the discovery system to encode extension-specific parameters in the search space in addition to type names. Nor does it easily admit highly combinable extensions, where a negotiation would likely mix in many different types. Type-based discovery of combinations would require either impractical population of the discovery system with different combined types (in some canonical form) or an algebraic type model in the discovery system's type-based query mechanism.

A different approach, as exemplified in the Condor Matchmaker [6], is to use a generic syntactic model to structurally compare provider capabilities with a client-provided description. The drawback of this approach is that registries with typical query languages cannot be reused—the Matchmaker capable of doing the structural match is in effect a specialized registry using the matchmaking algorithm as the query processor.

Ideally, we believe that WS-Agreement should adopt a hybrid solution between these extremes. Some discovery processes will undoubtedly be subsumed by matchmakers, brokers, and other negotiating intermediaries [3, 5]. However, abstracted characterization of providers should be possible in order to use existing registry models such as OGSI's ServiceGroup [8]. Therefore, we believe that the WS-Agreement term model must support both rough characterization of provider capability and policy, eg. flattened representations of available terms and possibly value ranges, as well as exhaustive validation, eg. matckmaking or *structural unification* of initiator-desired terms and provider capabilities.

1.2 Implementation Concerns

The GRAAP working group faces difficult design challenges to achieve these goals while utilizing Web services technologies for *term and constraint languages*, negotiation messages, and negotiator characterization.

Some concerns arise from the limitations of existing tools for binding the XML Schema language to programming languages used for implementing Web services. These issues are discussed in the term language section of this document.

The chosen design will influence the nature of WS-Agreement entity implementations, depending on how many advanced features for message structuring and validation are assumed in the specification.

Beyond the practical issue of generic program code for reuse in WS-Agreement implementations, some use cases that we envision for WS-Agreement require generic behavior in deployed implementations. While intermediating negotiators may be specialized for a given problem domain [5], eg. the recently announced Platform CSF for computational job meta-scheduling, our experience with co-allocators [4] suggests that a number of intermediary behaviors are generic—in fact, much of the complexity of intermediaries comes from fault-handling behavior during the asynchronous negotiation in addition to any domain-specific planning mechanisms. We believe that an important category of WS-Agreement implementation will provide generic brokering or aggregate-scheduling capability "out of the box," while admitting run-time extension of the domain-specific negotiating details. The approach chosen for the WS-Agreement model will have profound effects on the viability of such products.

1.3 Namespaces

The following namespace prefixes and namespace values are used in this document:

Prefix	Specification	Namespace
wsp	WS-Policy	http://schemas.xmlsoap.org/ws/2002/12/policy
wsa	WS-Agreement	http://www.gridforum.org/namespaces/agreement (temporary)

1.4 Overview

Given these overall concerns, we wish to reevaluate all ongoing design activities for WS-Agreement. Experimentation during the design process informs some of our current position, but other concerns below are speculative, based only on reflection with respect to the above-stated principles. Specifically, in this memo we focus on the following areas where critical WS-Agreement design decisions must still be made:

1. *Symmetric negotiation.* Our intuitive models for offer exchange involve a symmetric (or reciprocating) dialog of offers and counter-offers. However, the realities of a Web service environment are often less symmetric. We propose mechanisms for synchronous and asynchronous offer exchange and *publication* that clarifies the negotiation model without significantly changing the semantics.

2. *Term language.* We examine our standing term language proposal [1] and report on experimental results and refinement of our view as it pertains to the core term language of WS-Agreement.

3. *Constraint language.* There are unresolved issues with mixing of terms and constraints in an extensible and interoperable way, both from a technical perspective with respect to contemporary XML tools and a speculative perspective with respect to matchmaking algorithms and soundness.

4. *Offer structure.* A recurring discussion in GRAAP-WG indicates that people have a generic understanding of "offers" exchanged between parties in a negotiation. We wish to exploit this generality and provide a normalized semantics for WS-Agreement in terms of sequences of offer exchanges.

5. *Agreement templates.* Agreement templates are the means by which WS-Agreement providers can advertise their supported terms, constraints, and policies for interested initiators.

6. *Negotiation protocol.* Given mechanisms for symmetric and asymmetric exchange of offer documents, we need to define a rigorous semantics for WS-Agreement. Such a semantics defines the meaning of sequenced exchanges and conversely isolates meaningless exchanges that are not used with WS-Agreement. We propose informally a commitment protocol based on *advisory*, *soliciting*, and *committing* offers.

Each of these design decisions will affect the practical ability of WS-Agreement to support interoperable resource management in the Grid. They are explored in more detail in the remainder of this paper.

2 Symmetric Negotiation

To enable WS-Agreement to serve a wide range of scenarios, we wish the mechanisms to be rich enough for symmetric negotiation between Web service-enabled parties while also serving asymmetric negotiation between a Web service-enabled provider and a client initiator. Due to cost of implementation or user-driven constraints, a client may not be willing or able to provide full Web service functionality, i.e. to provide a Web service endpoint. Nonetheless, we wish the two modes of operation to share as much mechanism and capability as is possible without compromising either scenario.

By exploiting the notion of a generic offer structure, as suggested in the Introduction, we can factor the WS-Agreement mechanisms to provide a Web service interface for *delivering offers* to a negotiating party. This interface can be implemented by both parties in symmetric negotiation, or, by the (Web service-enabled) provider in asymmetric negotiation.

2.1 Changes to AgreementFactory Interface

This proposal requires modest change to the AgreementFactory interface. The factory pattern is retained, and the result is still an Agreement (which now is also a AgreementParty). However, to support AgreementParty-enabled intitiators, the creation parameters to the AgreementFactory need to include a locator identifying the initiator's AgreementParty. Abstractly, the combined pattern for Agreement creation looks much like OGSI notification subscription. The factory call creates a new pair-wise negotiating context represented by the Agreement, including the endpoint to which the initiator wishes subsequent offers to be directed.

2.2 New AgreementParty Interface

We propose that WS-Agreement be re-factored to include a new interface herein referred to as AgreementParty. The AgreementParty interface will be composed with other WS-Agreement mechanisms to form the Agreement interface. It may also be implemented in a standalone fashion for use by the Web service-enabled initiator in symmetric negotiation. Thus, AgreementParty subsumes the existing renegotiation mechanism in WS-Agreement while also (we hope) clarifying the asynchronous message pattern implied in the original WS-Agreement draft.

To support the widest range of scenarios, we include in the AgreementParty interface not only the offer delivery operation to send offers to the AgreementParty, but also a complementary *offer publication* mechanism to expose pending offers as service data. In the symmetric scenario, this offer publication is redundant for the party receiving the offer, unless that publication is exploited to enhance the reliability or robustness of the offer exchange. However, for the asymmetric scenario, this offer publication becomes the only mechanism by which the provider can deliver asynchronous offers to the client. As discussed in Section 6, synchronous provider counter-offers may be overlaid on the return message in response to offers delivered to the provider.

2.2.1 Service datum offer

The offer service datum is the mechanism through which the AgreementParty can publish its current offer.

Issue 1: Should offer present only a single offer, multiple offers (cardinality greater than 1), or multiple offers in a WS-Policy composition (cardinality 1)?

2.2.2 Operation deliverOffer

The only operation in the AgreementParty interface is deliverOffer. This operation subsumes renegotiate. Both operations are under-specified at the moment, but we believe that deliverOffer should remain a simple delivery envelope for the offer syntax that must be designed. This strategy supports reuse by using the same offer syntax in both the createService and deliverOffer invocations.

2.3 Impact

The main disadvantage of offer publication—as in this proposal or in the existing WS-Agreement draft—is that the one sending the offer does not know whether the offer was delivered or not. While this does in effect extend the "period of uncertainty" for the publisher, it does not fundamentally change the risks taken by negotiating parties in an faulty, asynchronous, distributed system. Our discussion of the negotiating protocol in Section 6 further examines this issue.

Advantages of offer publication for both symmetric and asymmetric scenarios include the buffered delivery of offers and third-party monitoring of WS-Agreement systems. As mentioned before, the buffering of offers may allow a negotiating party to recover from transient or localized faults by reviewing the status of his counterpart in the negotiation. For asymmetric negotiation, this buffering enables the initiator to poll the provider for asynchronous offers. Third-party monitoring exploits the subscription and polling mechanisms supported for service data to allow other clients, such as resource and community administrators or technicians, to view the offer publications (if the access control policies are such as to allow this monitoring).

2.4 Alternatives

There are a few variants that should be considered for WS-Agreement instead of the proposed AgreementParty refactoring.

2.4.1 Omit offer publication

If offer publication is too complex or has other unseen problems, WS-Agreement could simply disallow asynchronous provider offers in asymmetric negotiation. However, we argue that this feature does not significantly complicate the AgreementParty interface once the other features are in place. Furthermore, any AgreementParty has the right to decide which offers to publish versus deliver by more direct means. If future complications are found, this feature could be selectively disabled where it is inconvenient or disruptive to the AgreementParty.

2.4.2 Use anonymous endpoints

There are emerging standards at the Web service binding level to permit anonymous Web service-endpoints, eg. WS-Addressing [2]. Such standards could be exploited as an alternate means to deliver asynchronous offers to an initiator who is unable to implement a normal Web service endpoint, eg. because of an intervening firewall. However, such exploitation is necessarily binding-specific and as such we do not recommend making WS-Agreement *reliant* on it.

3 Term Language

3.1 Meta-Language Requirements and Abstract Model

The term meta-language should provide a framework for defining domain-specific term languages used to express agreements or offers. The following features are required:

1. *Negotiability/Criticality Flags.* Terms can be marked with domain-independent meta-data for negotiation. In fact a term can be flagged as:

   (a) mandatory or optional

   (b) negotiable or not negotiable in terms of its contained value(s)

   See the constraint language section for a discussion of those flags.

2. *Tree-like structure.* An agreement or an offer is not just a flat list of name-value pairs. Terms can be simple (containing one simple value) or complex (containing fields/sub-terms which themselves can be complex or simple). Also, terms can be grouped. The term document(s) should therefore have a tree-like structure, where a term can have sub-terms automatically qualified by their parent, which provides context.

3. *Extensibility.* Terms already modeled in a domain-specific language should be ”extensible”

   (a) *at the modeling level:* it should be possible for a provider to extend terms with provider-specific sub-terms, in order for instance to further specify the domain-level definition of the service it is able to offer.

   (b) *at the negotiation level:* negotiating parties should be able to edit offers by inserting sub-terms defined at the modeling level. Sub-terms could be domain-specific data or behavioral constraints on the parent term (see the constraint language section for a discussion on how to specify constraints).

4. *Boundaries and Constraints.* Terms are not supposed to only be fixed values. It should be possible to express constraints such as boundaries and ranges of values for negotiation purposes or even to specify flexible service behavior. The term meta-language should therefore provide ways to mark terms with constraints. See the constraint language section in this document.

5. *Term Grouping.* It is useful to group terms so as to apply negotiability attributes, criticality and other metadata to the entire group as opposed to individual terms. For instance the group of term can be marked as a required or optional term, negotiable or not, etc. The term meta-language must offer a way to group terms logically (using AND, OR, XOR operators). In this way it is possible to express for instance that all terms in a set are required, or only one of them.

3.2 Issues in Schema Encoding

We are facing several issues when attempting to model terms in XML and create an XML encoding of the meta-language. A lot of issues arise from the fact that there is not one way to express terms in XML.

1. *Schema Typing Approach*

   In order for the meta-language to express syntactical rules that the XML representation of terms should follow, it is useful to record those rules in a Schema document (that can be used at run-time for validation, and as input to development tools). The question is raised of how to model those rules into XML Schema types.

   There are different approaches to modeling the terms in Schema:

   (a) *A strongly-typed model* where every domain-specific term structure corresponds to a statically defined Schema type. For example a fileTransfer term corresponds to a FileTransferType Schema type which can declare elements localPath and remoteURL.

   (b) *A loosely-typed model* where the content of every term is typed as a multiple element and attribute wildcards. For example a fileTransfer term corresponds to a TermType Schema type which contains

   ```
   <xsd:any maxOccur="unbounded" ...>
   ```

   (c) *Hybrid approach* where complex terms are typed with complex Schema types containing defined fields but also extensibility points via wildcards. This is the approach we chose in [1].
   For example:

   ```
   <xsd:complexType name="FileStageInTermType">
       <xsd:sequence>
           <xsd:element name="remoteSource" type="wsa:TermType"/>
           <xsd:element name="localDestination" type="wsa:TermType"/>
           <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
       </xsd:sequence>
   ```

```
    <xsd:anyAttribute processContents="lax"/>
</xsd:complexType>
```

2. *Tooling Issues*

   We noticed limitations of the existing Schema compilation and/or marshalling tooling which does not correctly support important Schema features used in the WS-Policy schema and our current WS-Agreement term language schema (as explained in [JSDL]). For instance the Axis 1.1 WSDL2Java code generator currently (January 2004) lacks support for or fails when encountering Schema constructs such as:

   (a) *unions* (used in wsp:OpenUsageType to allow other values than wsp:Required, wsp:Optional...that are not predefined),

   (b) *derivation by restrictions* (used to specialize wsp:PolicyExpression into wsa:AgreementType),

   (c) *enumerations of QName values* (used in wsa:UnboundedOccurenceType).

3. *Issues with WS-Policy*

   Using the WS-Policy specification to define the meta-language for terms raises questions of usage and potential limitations.

   We are wondering how WS-Policy compositors should be used. WS-Policy compositors can be used to group terms logically (AND, OR, XOR). Shoud we specialize grouping operators (compositors) such as wsp:All, wsp:ExactlyOne and sp:OneOrMore? Or reuse the already defined compositor instances?

   The WS-Policy schema prevents using type derivations of WS-Policy types polymorphically. In fact, the WS-Policy schema allows for derivation of the types it defines, which includes wsp:Compositor. However it forbids using derived types as the types of instances (elements) defined in the WS-Policy schema.

   So if one must derive compositors, it must also declare all elements that will be of those derived types. It is not possible to rely on the exisiting compositor instance declaration in the WS-Policy schema. For instance, it is not possible to derive wsp:PolicyExpression by restriction to define wsa:AgreementType and rely on the fact that the base tpe contains references to the WS-Policy compositors. References to new elements must be made [7].

   Consequence: this can complicate the task of creating specialized compositors since elements declaration also have to be made. These restrictions and the WS-Agreement solution will also affect generic WS-Agreement implementations.

   Should we discontinue using WS-Policy?

Our JSDL proposal [1] gives an example of XML encoding for the meta-language, which unfortunately suffers from the preceding problems.

4 Constraint Language

Following our proposal for JSDL [1], the composite term framework, reusable metric types, and domain-specific terms can be qualified by constraints. Constraints on metrics in a term can be characterize one of two major aspects of that term's use in WS-Agreement:

1. *Partial behavior* may be specified through limits on a variable service metric. The **numberOfCPUs** term of a multi-threaded time-sharing application, for which a device-counting metric could be modeled as a simple positive integer, might have an upper bound **numberOfCPUs maximum limit** during the lifetime of a job where the job system may dynamically change CPU allocations. Compare this to a simple fixed form of the term, eg. the static allocation of processors to a space-sharing parallel computation. There are several variations of partial behavioral constraints:

   (a) *Limits* set upper or lower bounds for an ordered metric (such as a natural or real numbered metric).

   (b) *Enumerations* define finite sets of discrete values.

   A degenerate case of enumeration is where only a single fixed value is acceptable. In this case, no special constraint syntax is needed if the term follows our extensibility proposal to permit direct use of the reusable metric types.

2. *Negotiability* may be specified through limits imposed by a negotiating party on term metrics. These constraints serve to inform the other party of offered values that are *likely* to be accepted by the other party during negotiation—they are not guarantees, as only completion of the WS-Agreement negotiation protocol described in Section 6 can decide this outcome. There are four conceptual variations of negotiability constraint:

   (a) *Fixed term negotiability* expresses a subset of meaningful (service) values that are acceptable to either party, eg. minimum or maximum selectable **numberOfCPUs** of a space-sharing job.

   (b) *Behavioral constraint negotiability* expresses a subset of meaningful (service limit) values by either party, eg. minimum or maximum selectable **numberOfCPUs maximum limit** of a time-sharing job.

   (c) *Cardinality rules* may be applied to domain-specific terms that can be meaningfully repeated, eg. an **inputFile** term for a job might be replicated to indicate that the job takes multiple input files. A particular job system might have restrictions on how many input files it is willing or able to handle for the user.

   (d) *Arbitrary term properties* may be defined by either negotiating party, eg. whether **numberOfCPUs** is required or optional from the point of view of that party.

   The fixed term and behavioral constraint negotiability can be modeled through the same reusable negotiability constraint language because they operate on the same reusable metric values wherever they might be embedded.

4.1 Representation of Constraints

In our JSDL proposal, we offered a preliminary XML syntax for the term and constraint languages. However, we are not certain yet where constraints should be specified in a complex Agreement state document, so we revisit the issues here. The two main approaches to constraint representation are embedded markup and external constraint annotations:

1. *Embedded constraints* use some aspect of term extensibility in order to reuse constraint syntax with domain-specific terms. A partial behavioral term would contain a value constraint as a "sub-term" of the domain-specific term. As a nested term, the behavioral constraint would also benefit from all of the capabilities afforded other terms, eg. negotiability constraints and logical compositors.

   Embedding negotiability constraints in this way might be awkward because, while they might benefit from logical compositors, it is not meaningful to apply negotiability constraints recursively.

2. *External constraints* annotate a closed term document by addressing or referencing the term(s) they qualify and providing the additional qualifying information. As an external annotation, these constraints could qualify terms that do not support extensible markup. However, because these constraints affect the semantics of the Agreement state, this approach obfuscates the service behavioral description and complicates agreement processing. The term compositor tree becomes a forest of base terms and constraint annotations, cross-linked by term address and yet still subject logical composition.

   Annotating the term tree with external negotiability constraints might be beneficial for these same reasons—we maintain a clear syntactic distinction between the negotiated term state and the unilateral negotiability policies of each party, and we avoid opening the negotiability constraints to nonsensical recursive nesting. There is still the same drawback of a more complicated processing in moving from a simple tree to a directed graph structure.

We believe a hybrid approach may strike the right balance: embedded behavioral constraints and external negotiability constraint annotations exploit the asymmetric benefits and drawbacks described above. Our initial JSDL proposal [1] took a different approach of always embedding both forms of constraint.

Our revised opinion is that while behavioral constraints should remain inside the offer, it makes sense to separate the negotiability constraints part of the offer from the semantic part of the offer (the terms and their behavioral constraints, which together are the actual terms of the offer). This allows for a clear syntactic separation of these different concepts. In addition, keeping the negotiability constraints out of the term tree means that we can share a single term document in the Agreement denotational state model while carrying separate annotation lists for the negotiability constraints imposed by each party. Thus, the Agreement state model should have three parts:

1. A tree of terms and compositors with their embedded behavioral constraints.

2. The negotiability constraints of the provider as an external annotations list which addresses elements in the term tree.

3. The negotiability constraints of the initiator which also addresses the term tree.

In terms of service interface, this could translate into a complex service datum in the Agreement service. While the protocol proposed later in Section 6 does not require service data for its basic function, such data could be used by a priveleged third party in order to inspect the state of the negotiation at any point in time and thus monitor the progress of the negotiation.

4.2 Other Term Qualifiers

The state of the commitment between the two parties is neither part of the negotiability constraints nor part of the semantics of the service. However, it is part of the state of the agreement negotiation, and as such is common to both parties. We therefore propose to represent it embedded inside the term tree just as we embed simple fixed metrics. Each term can be flagged with a commit state. Our currently proposed commitment state values are described in Section 6.

Another type of qualifier envisioned for a term relates to violation or penalty clauses on terms. Part of the semantics of a term is the severity of failures to provide service according to the domain-specific meaning of the term and its embedded metrics, whether partially or fully constrained. We are not sure what importance such clauses will have in the near-term application of WS-Agreement to real use cases, and we are not aware of any special language features needed to support this sort of term qualifier. Our lack of proposals in this area is not intended to discourage other groups from defining useful violation, cost, or penalty clauses.

5 Offer Structure

Offers are the abstract messages that are exchanged in the negotiation protocol. The meaningful sequences of such messages are discussed later in Section 6. An important decision in the WS-Agreement design is whether offers are incremental *change* messages against an implied Agreement state or, rather, independent and fully denoted Agreement state descriptions.

We propose that the general offer structure should be change based and capture negotiation against an implied Agreement state. In doing so, we advocate having a denotable Agreement state model (based on the full term and constraint language) and a separate offer syntax for describing updates to this state.

Issue 2: Should the offer syntax be based on some existing XML change-set standard or a WS-Agreement-specific solution?

5.1 Change Modes

There are, of course, idiomatic abstract modes of operation against an implied Agreement state:

1. *Add term(s) or term detail(s).* If supported, a party may wish to augment the Agreement state with additional terms or add additional details to existing terms.

2. *Delete term(s) or term detail(s).* If supported, a party may wish to remove non-critical terms or details in existing terms.

3. *Replace term(s) or term detail(s).* Replacement is just a composition of add and delete and does not need further consideration except where atomicity might be at stake. We mention it because idiomatically, modification of the details of a term, eg. to adjust a numeric parameter, is done by replacement of the term or some of its details. Using add/delete would result in an intermediate step where the term document might have non-validating terms according to the term schema.

Depending on how terms and constraints are distinguished in WS-Agreement, modification of terms may be syntactically represented as adding and removing constraints. The effects of these design decisions must be considered together. Assuming a hierarchical XML representation of Agreement state based on WS-Policy and WS-Agreement term languages, these operations will be parameterized with hierarchical context addresses, eg. XPATH or XPtr expressions.

5.2 Offer Parts

The offer consists of change directives. While the term document of the Agreement state carries a term document and negotiability constraints, the offer carries changes to this state.

5.2.1 Changes to terms

The term document changes in the offer document denote specific textual modifications to the implied Agreement term state. The validity and effect of these offers on the state is subject to the policies and interpretation of the party receiving the offer. For example, the offer can express changes which are not syntactically possible, or which violate the commitment protocol described in Section 6. However, we want to distinguish this sort of pathological offer from valid but *unacceptable* offers that the other party might reject due to local policies.

An offer contains one or more change elements, for instance to add new terms and delete and/or replace others. It also contains a potentially nillable element with changes to the asserted negotiability constraints of the sender.

Each change-element specifies if the operation applies to an element or an attribute. The syntactical specifics of each command are contained in the element that the command is specified with. In particualar, the location parameters vary depending on the command:

1. *add.* An issue with adding one or more new term element(s) to a term which already contains children elements (sub-terms) is to decide of the final position of the term(s) to add relatively to the rest of the children elements of the target term. Being able to specify the position is useful for instance in order to respect a sequencing of sub-terms mandated by the modeling of the target parent term. We therefore propose a syntax of the add command with optional paths 'before' and 'after' giving indication as to where to insert the new term(s). The term(s) will be inserted at the same nesting level as the elements pointed to by the paths, but right before the 'before' element and after the 'after' element.

2. *delete.* the path represents the term element or attribute to delete.

3. *replace.* the path represents the term element or attribute to replace. The command specify a whole replacement for that element or attribute.

There are several possible ways to encode the operations in XML. Many questions can be raised, such as: should the negotiabilityChanges element be nillable or optional? Before we can make the right decisions as to how to represent the data, we need to spend time analyzing the needs and modeling the meta-language in the abstract. Only when we agree on what needs to be represented and with what exact meaning will we be able to make decisions concerning the encoding into XML.

Therefore, the mapping from the meta-language concepts to XML representation suggested by the examples presented below is not prescriptive. The examples are here only to illustrate the ideas presented in this section.

For example, the following change element adds two terms to the top-level WS-Policy wsp:All compositor of the agreement. The location of the target parent element is specified in the path attribute. The terms will be simply appended to the content of the target parent element.

```
<wsa:add path="//wsp:All">
    <elements>
        <job:fileStageIn>
            <remoteSource wsa:fileURL="someFileURL"/>
            <localDestination wsa:filePath="someFilePath"/>
        </job:fileStageIn>
        <job:numberOfCPUs wsa:count="16"/>
    <elements/>
</wsa:add>
```

The example assumes the template syntax (included in appendix) of our JSDL proposal[1].

Issue 3: should we mandate only one term per add operation and rely on aggregation of add operations at the term level or is it so useful to be able to factor the location to add to into one single add operation?

The following change element is the same as above but the two new terms are inserted before the job:arguments term.

```
<wsa:add path="//wsp:All" after"job:arguments">
    <elements>
        <job:fileStageIn>
            <remoteSource wsa:fileURL="someFileURL"/>
            <localDestination wsa:filePath="someFilePath"/>
        </job:fileStageIn>
        <job:numberOfCPUs wsa:count="16"/>
    <elements/>
</wsa:add>
```

In the following example, the change element modifies the job:numberOfCPUs term in the agreement being negotiated. More precisely it seeks to modify the attribute wsa:count (as pointed to by the XPath value in path) by setting it to the value "20":

```
<wsa:replace path="//wsp:All/job:numberOfCPUs/wsa:minimum/@wsa:count">
    <attribute wsa:count="20"/>
</wsa:replace>
```

The example assumes the agreement syntax (included in appendix) of our JSDL proposal[1].

A complete offer could look like:

```
<wsa:offer>
   <termChanges>
      <wsa:add path="//wsp:All">
          <elements>
              <tns1:term1 .../>
              <tns1:term2 .../>
              ...
          <elements/>
      </wsa:add>
      <wsa:replace path="//wsp:All/job:numberOfCPUs/wsa:minimum/@wsa:count">
         <attribute wsa:count="20"/>
      </wsa:replace>
      <wsa:replace path="//wsp:All/job:numberOfCPUs/wsa:maximum">
         <element>
            <wsa:maximum wsa:count="40"/>
         </element>
      </wsa:replace>
      <wsa:delete path="//wsp:All/wsp:OneOrMore"/>
      ...
   </termChanges>
   <negotiabilityChanges nil=true />
</wsa:offer>
```

Note: These examples show modification requests to service terms but leave out the commitment state for the sake of simplicity. It is explained in the corresponding sections of this document.

5.2.2 Changes to negotiability constraints

Because negotiability constraints are unilateral statements made by one party to the other, there is no ability to address changes against the other party's negotiability constraints. The offer contains one negotiability-changes section that affects the textual content of the constraints asserted by the sending party to the recipient party.

Here is an example of negotiability constraint change section. The actual change syntax of the change commands is the same as for term change. The path field refers to the term to constraint.

This example adds a maximum of negotiability to a fixed count term (job:numberOfCPUs) and replaces a minimum of negotiability (already existing in the list of negotiability constraints of this negotiating party) with a different value.

```
<negotiabilityChanges>
      <wsa:add path="//wsp:All/job:numberOfCPUs">
         <wsa:maximum wsa:count="32"/>
         [...other constraints...]
      </wsa:add>
      <wsa:replace path="//wsp:All/job:numberOfCPUs/wsa:minimum/@wsa:count">
         <attribute wsa:count="20"/>
      </wsa:replace>
<negotiabilityChanges>
```

5.3 Agreement Templates

Agreement *templates* are simply negotiability constraints published by the AgreementFactory as service data. This may be surprising to the reader who might assume that templates (advertisements by a provider) would be expressed in the denotational Agreement state model. In Section 6, we explain how templates are considered as just one more (prefixing) message in the offer exchange sequence. An alternative viewpoint is that templates are offers with an empty change-set (empty because there is no well-defined Agreement state to operate on until after the Agreement is created). However, context regarding identities of parties, etc. , may be meaningful. A pending decision is whether to advocate such context through some variant of our constraint language or some other policy annotation mechanism.

Issue 4: Is there an immediate need, and simple method, to annotate such a template to distinguish the case where a provider will only accept initiator pre-committing or initiator soliciting offers, i.e. to signal that the provider can only handle a strict subset of the proposed negotiation protocol such as GRAM batch submission? Is this required in general for negotiability constraints? If so, perhaps solicitation should be removed as a term characteristic and formulated only as advisory terms with negotiability constraints requiring a pre-commitment response!

5.4 Sequence-defining Template

A more advanced form of template was suggested during a previous face-to-face meeting of GRAAP. **We believe this approach is orthogonal to the proposals in this memo, and do not recommend considering that (much more complex) model for the first version of WS-Agreement.**

However, to achieve consensus on this point, we can quickly describe this approach in terms of the proposal here: wrapping around the base model described above, we might define a negotiability constraint syntax that describes a (partial) order of the simpler change-set offers (each with a potentially different set of negotiability constraints). Several new negotiation patterns could be defined to exploit this “offer-flow” language:

1. A two-phase negotiation where first the change-set sequence is committed by the parties (protocol negotiation) and then the terms are negotiated by the parties in a process consistent with the mutually-agreed sequence.

2. A reduction model where each step in the offer exchange *executes* one of the simple offers (in partial order) until the system arrives at the final Agreement state implied by that workflow. The execution of simple offers would use the change-sets from the offer within the range of acceptable parameters characterized by the negotiability constraints in that simple offer.

3. An advisory model where the change-set sequences are used to augment the unilateral negotiability constraints sent by either party. The other party can use these hints to decide which of its negotiation strategies to pursue (or whether to pursue negotiation at all).

To reiterate, we think these might be interesting areas for future investigation but not in the time-frame desired for an initial WS-Agreement specification.

6 Negotiation Protocol

The WS-Agreement model proposed in this memo consists of abstract offer messages bearing the offer content as described in Section 5. These abstract messages must be overlaid on concrete Web service messages in order to define the

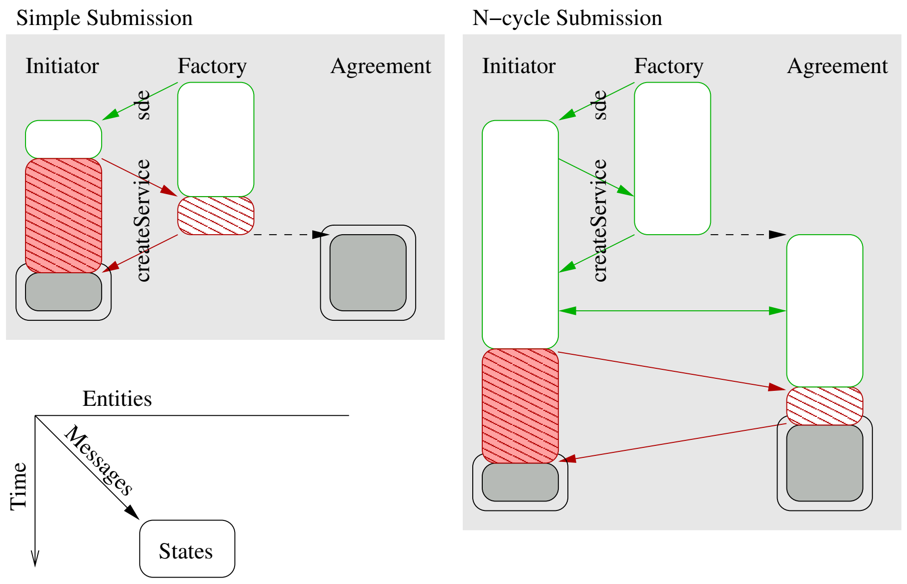

Figure 1. Simple and N-cycle submission. A round-trip of committing offers (in red), started by the initiator, lead both scenarios to an observed agreement state. In the N-cycle variant, a number of advisory offers are exchanged before the initiator finds terms he wants to observe. In this and subsequent protocol figures, fault transitions are omitted; faults can abort negotiation and return parties to the start state at the top of the time-event diagrams.

WS-Agreement interface. We have two operations to work with: the createService operation of the AgreementFactory and the deliverOffer operation of the AgreementParty. The input document to each operation will contain an offer, with the createService input also containing other context-initializing values. The output from these operations will be either a reciprocal offer or a fault signaled by the invoked party, i.e. the first party makes an offer in an input message and the second party makes a counter-offer in the output message.

In this section we examine the different kinds of offer and ordering requirements on these offers. The valid sequences of offers lead to an observed agreement between the two parties. Thus, we informally define the negotiation and commitment protocol for WS-Agreement.

6.1 Offer Types

The proposed negotiation and commitment protocol uses three types of offer to change the state of the negotiation:

1. *Committing* offers indicate that the sender of the offer is committed to the Agreement state implied by the offer in its context. The first party to send such commitment is *pre-committed* or “out on a limb” waiting for the other party to reciprocate commitment or reject the offer. The second party to send such commitment knows the agreement is observed.

2. *Soliciting* offers indicate that the sender of the offer requires a committing counter-offer. The party who sends a soliciting offer is able to precipitate a conclusion to negotiation without himself pre-committing.

3. *Advisory* offers describe intent of the sending party but do not commit either party or otherwise constrain subsequent offers. WS-Agreement does not require counter-offers to an advisory offer to satisfy constraints, converge, or otherwise follow any structured conventions, though a community of WS-Agreement providers and initiators may wish to follow such conventions.

These simple offer types lead to a very rich set of negotiation strategies which all terminate with the same simple commitment handshake. This handshake is designed to minimize ambiguity in the face of failures and delays, while clarifying the inherent risk in negotiation in a wide-area, non-deterministic network such as the Web.

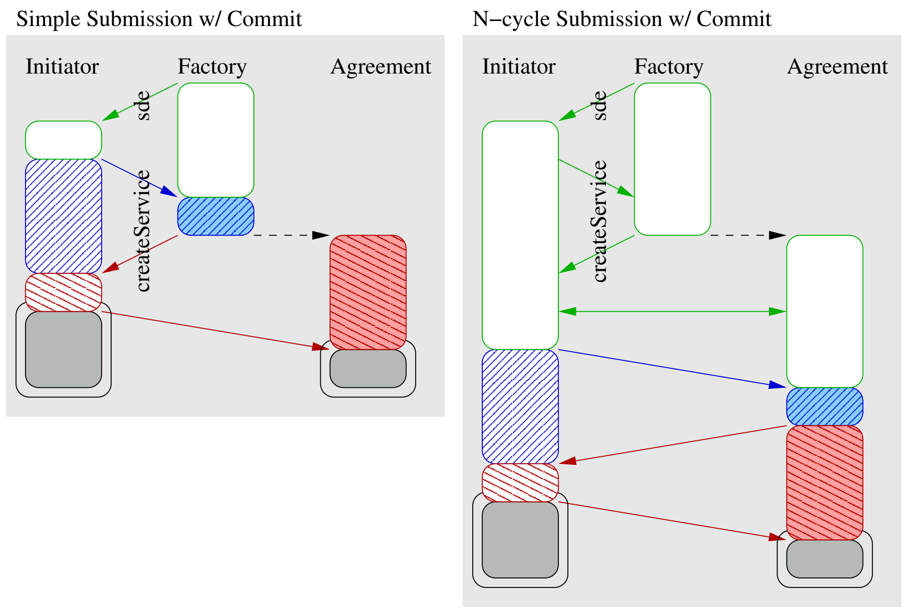

Figure 2. Simple and N-cycle submission with delayed commitment. A round-trip of committing offers (in red), started by the provider, lead both scenarios to an observed agreement state. Instead of the initiator pre-committing, he solicits the provider to pre-commit and reserves the final commitment decision for himself.

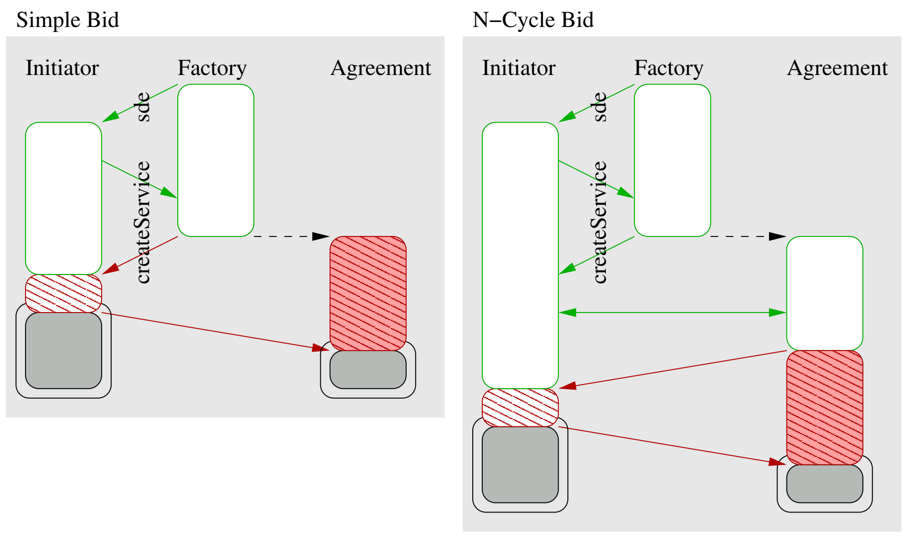

Figure 3. Simple and N-cycle bid. A round-trip of committing offers (in red), started by the provider, lead both scenarios to an observed agreement state.

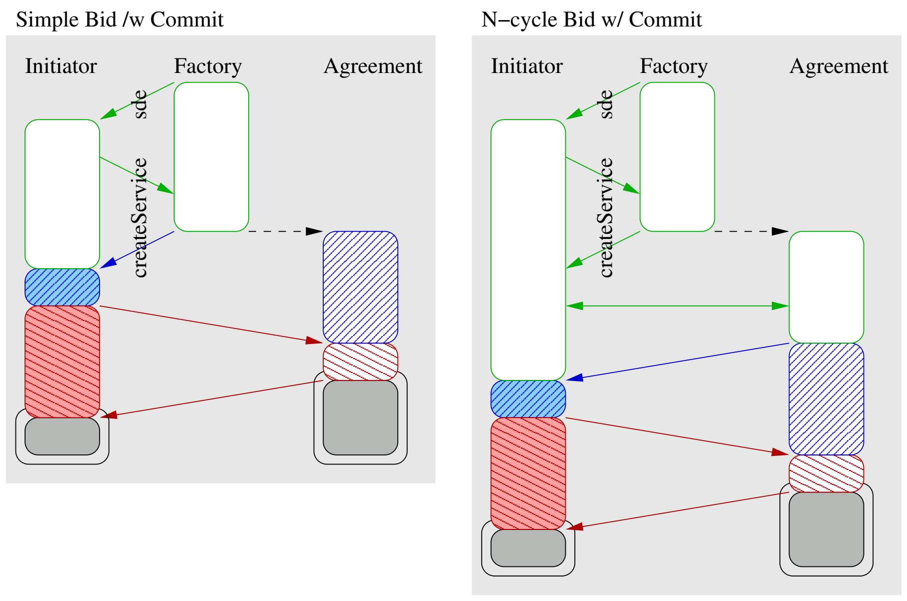

Figure 4. Simple and N-cycle bid with delayed commitment. A round-trip of committing offers (in red), started by the initiator, lead both scenarios to an observed agreement state. Instead of the provider pre-committing, he solicits the initiator to pre-commit and reserves the final commitment decision for himself.

6.2 Negotiation State

The term document in the implied Agreement state may be decorated with commitment state. Due to distributed system asynchrony, the initiator and provider may have instantaneously distinct beliefs in the implied Agreement state. The existing WS-Agreement specification draft includes attributes to represent commitment state, but we think a different set must be considered.

The term document as represented in the AgreementParty interface represents the view Agreement state held by the negotiator implementing that interface. Specifically, the term document exposed in the interface of the provider has terms in these states:

1. *Advisory* terms are probably satisfactory to both parties but have no effect on agreement semantics.
2. *Initiator-solicited* terms have been solicited by the initiator for the provider to respond with an offer. This state only exists until the provider issues an committed counter-offer or a fault.
3. *Provider-solicited* terms have been solicited by the provider but an initiator committed counter-offer has not been received.
4. *Initiator-committed* terms have been pre-committed by the initiator but not by the provider. This state only exists until the provider issues a counter-offer, eg. a reciprocal commitment or a fault.
5. *Provider-committed* terms have been pre-committed by the provider but an initiator counter-offer has not been received.
6. *Observed* terms have been committed by both parties and the provider knows this.

This state machine may be executed independently on different terms in the term document. Terms without explicit decoration inherit the commitment status of their ancestors. The root element in the term document MUST be decorated with a commitment state. Because our proposal shifts some of the previously-considered "states" into negotiability constraints, the protocol state machine should encode only the above states.

While the semantics of different commitment states on sibling terms in the tree is easily understood in the abstract, it is beyond the scope of this memo to explore the meaning of all nested combinations of commitment states. For consideration:

we do not think it is meaningful for a wsp:All term to be committed if any of its children are not yet committed; but it seems reasonable for some children to be committed while the wsp:All is still advisory.

Issue 5: Should the initiator-committed/provider-committed and initiator-solicited/provider-solicited states be named explicitly or implicitly through asymmetric designations that are understood in relation to the party presenting the term document, eg. self-committed and other-committed?

Issue 6: Does the working group have consensus on the need for the solicited state? It could be omitted at the expense of having a simpler but more opaque commitment process—a party has no way to signal that he wishes the other party to pre-commit.

6.3 Commitment

Given the preceding offer types, WS-Agreement will support an arbitrary length dialog using advisory offers. When either party is ready to conclude the dialog, they decide to send a committing or soliciting offer. The choice depends on which party will be *pre-committed*, and this choice depends on both the natural asymmetry of *risk* in a given service or resource domain and the policies of the negotiators. By risk, we mean that the pre-committing party is waiting to hear the other party's decision. If the second party commits, the agreement is observed. During the period between when the first party issues the pre-committing offer and receives the reciprocal commitment, he is uncertain whether the agreement will be observed.

If the agreement terms include a schedule for service performance, the observed agreement is in force for the duration of that schedule, *even if the pre-committed party does not learn of the second party's commitment until after the schedule has begun or completed*. Due to the faulty, asynchronous nature of the global Web and the quasi-realtime nature of such schedule-based agreements, the pre-committed party has to gamble during the uncertainty period. If he speculates that the other party will commit, he may commence honoring the agreement terms even before he knows. If he speculates that the other party will reject the offer, he may not have enough time to honor the terms properly if the party actually accepts. This is particularly troublesome when the activities needed to honor the agreement cannot be performed speculatively, i.e. with rollback capabilities. There is potentially no "safe" choice where the pre-committed party can wait for further information. In the context of scheduled delivery, waiting is itself a choice.

This pre-commitment risk is a well known result of distributed systems theory. Since there is no way to resolve the conflict between asynchronous, arbitrarily-delayed messages and quasi-realtime agreement semantics, we can only distinguish the at-risk and safe roles in WS-Agreement and leave it to applications to mitigate costs. As suggested above, a first step is to analyze a particular service domain and put the risk on the party that can safely speculate and rollback as necessary. Secondly, if one can expect a certain statistical distribution of outcomes, the speculation can be biased to reduce average cost for successful and failed offers.

Issue 7: Should a fault on committing or soliciting offers conclude the negotiation, or just return it to its previous state?

6.4 Offer Granularity

The offer syntax must support both the initiating messages of the negotiation (template and createService) as well as further incremental progress in the negotiation. We believe that the template is supported by publishing negotiability constraints, while the createService offers are supported by expressing *add* offers against an *empty* implied WS-Agreement state. These messages may tend to be rather large, with a nearly complete term and constraint expression. On the other hand, subsequent deliverOffer offers may be much smaller *delete* or *modify* offers against localized parts of the (now populated) WS-Agreement state.

Issue 8: Should the createService offer be able to refer by name to a template as a non-empty implied WS-Agreement state, or should it have to duplicate the desired template content and always start with an empty state?

6.5 Negotiation State Granularity

The negotiation states implied by the offer types above can be applied at two levels of granularity: across the whole agreement and for specific terms or sets of terms. Again, we assume a hierarchical XML model for the Agreement state, and we mix the negotiation state into the attribute markup on that representation. The granularity of an offer is controlled through the hierarchical context addressing of the offer with regard to the implied Agreement state. A whole-agreement committing offer, for example, addresses and changes the negotiation state at the root of the Agreement state.

6.6 Complete Scenarios

The time-event diagrams in this section illustrate the combined WS-Agreement system state for negotiations between two parties. These diagrams show all possible successful negotiation sequences (leading to observed agreements), as an alternative to concurrent automata.

All of the figures show a template as the first message path. This template could be delivered directly through the provider's AgreementFactory service data or indirectly through a registry or other intermediary.

6.6.1 Submission

Figure 1 represents the simple and generalized form of negotiation where the initiator decides to send a pre-committing offer. The only difference between the simple and generalized form is the number of advisory offers exchanged before the initiator pre-commits. In effect, the initiator is able to discover more “personalized” information from the provider before formulating the pre-committing offer.

The submission pattern supports common use cases such as batch job submission.

6.6.2 Submission with Commit

Figure 2 represents the simple and generalized form of negotiation where the initiator decides to send a soliciting offer. As before, the simple and generalized form differ only in the number of advisory offers.

The submission with commitment pattern supports use cases such as batch job submission with delayed commitment, eg. as used by Condor to reliably “transact” jobs with GRAM.

6.6.3 Bidding

Figure 3 represents the simple and generalized form of negotiation where the provider decides to send a pre-committing offer. As before, the simple and generalized form differ only in the number of advisory offers.

The bidding pattern supports use cases such as batch job bidding, where a resource might “pull” jobs from a broker or central queue.

6.6.4 Bidding with Commit

Figure 4 represents the simple and generalized form of negotiation where the provider decides to send a soliciting offer. As before, the simple and generalized form differ only in the number of advisory offers.

We do not have specific use cases for this pattern yet, but provide it for symmetry in the negotiation protocol. We see no reason to think this pattern will not be useful in novel applications of WS-Agreement.

References

[1] A. Andrieux, K. Czajkowski, J. Lam, C. Smith, and M. Xu. Standard terms for specifying computational jobs. Technical report, Global Grid Forum, September 2003.

[2] D. Box and F. Curbera. Web services addressing (WS-Addressing). Technical report, BEA Systems Inc., International Business Machines Corporation, Microsoft Corporation, March 2003. http://www.ibm.com/developerworks/library/ws-add/.

[3] K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, and M. Xu. Agreement-based grid service management (draft 0). Technical report, Global Grid Forum, July 2003. http://forge.gridforum.org/projects/graap-wg.

[4] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

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

[6] M. Livny. Matchmaking: Distributed resource management for high throughput computing. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, 1998.

[7] H. S. Thompson. E-mail correspondance with Alain Andrieux regarding xsi:type usage. W3C XML Schema Developer mailing list, August 2003. http://lists.w3.org/Archives/Public/xmlschema-dev/2003Aug/0049.html.

[8] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open grid services infrastructure (OGSI) v. 1.0. Global Grid Forum GFD.15, June 2003. http://forge.gridforum.org/projects/ggf-editor/-document/GFD-R-P.15/en/1.