GWD-I

Category: Informational
JSDL-WG

Editors:
A. Andrieux, USC/ISI
K. Czajkowski, USC/ISI
J. Lam, Platform
C. Smith, Platform
M. Xu, Platform

9/22/2003

GGF DOCUMENT SUBMISSION CHECKLIST (include as front page of submission)	
	COMPLETED (X) - Date
1. Author name(s), institution(s), and contact information	(X) – September 19, 2003
2. Date (original and, where applicable, latest revision date)	(X) - September 19, 2003
3. Title, table of contents, clearly numbered sections	(X) - September 19, 2003
4. Security Considerations	(X) - September 19, 2003
5. GGF Copyright statement inserted (See below)	(X) - September 19, 2003
6. GGF Intellectual Property statement inserted.	(X) - September 19, 2003
7. Document format	(X) - September 19, 2003

alain@isi.edu

Editors:
A. Andrieux, USC/ISI
K. Czajkowski, USC/ISI
J. Lam, Platform
C. Smith, Platform
M. Xu, Platform

Category: Informational
JSDL-WG

9/22/2003

Standard Terms for Specifying Computational Jobs (Proposal)

Status of This Memo

This document does not define any standards or technical recommendations. However this is a proposal for the job terms that the Job Submission Description Language Working Group (JSDL-WG) [JSDL] seeks to specify.

Distribution of this document is unlimited.

Copyright Notice

Abstract

It is desirable to define a standard set of terms (otherwise known as characteristics, criteria or attributes) which compose the specifications of computational jobs so as to enhance interoperability between distributed systems dealing with resource management. Defining such a list of terms is the main purpose of the JSDL working group. This document is a proposed list of such terms as well as their static and dynamic representations in XML using the WS-Agreement term language.

Contents

Editors:
A. Andrieux, USC/ISI
K. Czajkowski, USC/ISI
J. Lam, Platform
C. Smith, Platform
M. Xu, Platform

1. 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 proposal 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 proposal.

Prefix	**Namespace**
Prefix	**Namespace**
wsa	“http://www.gridforum.org/namespaces/agreement” (temporary)
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”
jsdl	“http://www.gridforum.org/namespaces/job” (example)

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 [OGSI].
- The terms *initiator* and *provider* refer to the corresponding roles used in [WS-Agreement]: an OGSI-compliant Agreement service represents an ongoing relationship between an *agreement provider* and a customer (management client) known in WS-Agreement as the *agreement initiator*.
    - the *initiator* is the party making the request for a job (submission)
    - the *provider* is the party consuming the job request while advertising job characteristics it can satisfy within certain ranges (what is referred to in this document as *constraints*).

  The initiator and the provider can negotiate a job agreement until it is observed i.e. agreed upon by both parties. Only then can the job be executed.

2. Selection of the Terms

This list of terms was put together by:

1. Comparing the job terms supported by a set of distributed resource management systems and determine:
    a. the lowest common denominator of terms
    b. valuable terms that should be added to the list
    c. terms that should be considered for addition but need remodeling
2. Renaming terms to reflect more generic concepts than features specific to a particular scheduler.
3. Using the conceptual framework of WS-Agreement to
    a. remove from the list of terms those which should be expressed using higher-level WS-Agreement constructs.
    b. generalize constraints such as minimum and maximum so as to reduce the number of terms (for instance minStartTime and maxStartTime are merged into beginTime which can be constrained)
4. Using the WS-Agreement principles and building blocks for writing terms in order to define XML Schema [XSD Structures] definition of the job terms.

2.1 Compared Job Languages

Job languages from different resource management systems were compared so as to identify common terms and suggest terms that were potentially useful to include in a standard job definition language.
The following commercial scheduling systems were compared:

- Platform LSF
- Sun Grid Engine
- Altair PBS (Portable Batch System)

In addition, the Globus RSL (Resource Specification Language), which, in the Globus toolkit, acts a façade language that shields users from different scheduling systems, was also used in the comparison.
Most of the additionally suggested terms as well as the term names come from LSF.

2.2 On using WS-Agreement

The problem of job definition, submission, control and monitoring is common in the resource management space. The GRAAP working group, whose focus is to determine standards for resource management, proposed the WS-Agreement specification as a set of concepts and an abstract term language that can be used to define domain-specific service interfaces and standard languages for the management of resources. WS-Agreement can therefore be used to specify standard interfaces and language(s) for job definition and associated tasks. This document suggests a standard set of terms for a job specification and definition language (JSDL). Because they fit in the WS-Agreement framework, these terms exploit features such as individual negotiation and monitoring.

An agreement is written in XML and therefore is a tree of XML elements. Each term is represented as an element in the tree, and each element in the tree is considered a term and as such can bear attributes from the WS-Policy [WS-Policy] and WS-Agreement [WS-Agreement] frameworks so as to indicate if the element is required/optional, negotiable, observed (agreed upon by both the initiator and the provider of the agreement), etc… (See [WS-Agreement] for more information).

3. The WS-Agreement Term Language

Terms can be simple (containing one simple value) or complex (containing fields/sub-terms which themselves can be complex or simple). Schema types for domain-specific terms SHOULD be defined with open extensibility (using XSD construct such as xsd:anyAttribute and xsd:any, see [XSD Datatypes]) so as to

- allow the addition of extra content to the terms listed in a given standard specification (e.g. JSDL) for convenient profiling of the specification. For instance a jsdl:fileStageIn term features a localPath field and a remoteURL field, each typically bearing a string value to represent a file location. However, in certain communities of users, the remoteURL field could be further defined with a more meaningful or useful syntax.
- allow for dynamic markup (via elements or attributes) such as constraints. See the relevant section in this document.

Because of the extensibility points thus defined, specialized content can be specified statically for a community and if needed via derivation by restriction (see [XSD Structures]) or dynamically via XML documents showing what term combinations are possible (agreement templates).

3.1 Constraints

The WS-Agreement term language is designed so that a term can bear constraints, which can qualify either:

- behavior (a *numberOfCPUs* count could have a minimum and/or a maximum during the lifetime of a job), or
- negotiability (a *numberOfCPUs* count could be negotiable within a certain range).

Each constraint is represented as a sub-term of the domain-specific term so as to potentially be individually negotiable, required/optional…etc.

3.1.1 Limits

WS-Agreement defines generic limit-like constraints which can be applied to any number-like term (with an ordered domain):

- wsa:minimum
- wsa:maximum

It is possible to mark a limit as inclusive or exclusive for integer-like terms (with a discrete, ordered domain) by decorating the element with the wsa:inclusion attribute (possible values: wsa:inclusive and wsa:exclusive). The default is wsa:inclusive if the wsa:inclusion attribute is not asserted, to better match the intuitive meaning of a minimum or maximum acceptable value.

See the section of this document about the numberOfCPUs term section for several examples of mixing-and-matching behavioral and negotiability limits.

3.1.2 Other types of Constraints

WS-Agreement also defines:

- the wsa:enumeration constraint to restrict the value space of a term to an enumerated list of possibilities (see the section of this document about the runQueueLength term for an example).
- the wsa:useAttribute constraint to specify that the constrained term requires a particular attribute as the holder of its value (see jsdl:executable in the template in Appendix for an example).

3.1.3 Constraint Violations

It is possible to specify what happens when a behavioral constraint is violated by the negotiated service (executing job in the JSDL domain space). WS-Agreement defines generic actions that can be reused in job specifications. Those particles are not attributes but elements so as to provide dynamic extensibility (useful when, for instance, there is a need to markup an action with provider-specific details).

- wsa:terminateOnViolation (agreement is terminated if the service behavior exceeds the constraint; example: job is terminated if it runs too long)
- wsa: faultPreventsViolation (a fault is thrown for the operation or action taken by the service which would have exceeded a behavioral constraint; example: file write fails if a jsdl:fileSize limit is reached)

3.1.4 Types and Units for Simple Values

Some standard XSD types provide unit information embedded in them. WS-Agreement uses these standard types to define global particles which can be mixed in the terms and constraints so as to represent values such as:

- Intervals and periods: mix in a wsa:duration attribute (of type xsd:duration)
- Time, dates: mix in a wsa:dateTime attribute (of type ogsi:ExtendedDateTimeType, which is based on xsd:dateTime)

For the kinds of values for which no standard type exist, units should be normalized to standard units for consistency, i.e. for one given domain-specific kind of value; all values will be expressed in the same unit:

- Memory space and file sizes: provide a value expressed in *bytes* inside an attribute such as
  - wsa:amount (of type xsd:nonNegativeInteger)
  - wsa:real (of type xsd:decimal)

Counts (positive integers without unit) can be expressed by mixing in the attribute wsa:count (of type xsd:nonNegativeInteger).

4. Suggested Job Terms

4.1 executable

The path to the executable program which will be executed.

For instance, the initiator can make the following request:

```
<jsdl:executable wsp:Usage="wsp:Required" wsa:Negotiability="wsa:Fixed"
      wsa:filePath="/bin/processData"/>
```

The wsp:Usage value of “wsp:Required” indicates the initiator requires the term **executable** to be satisfied by the provider. The wsa:Negotiability value of “wsa:Fixed” indicates the provider cannot counter-offer another value for wsa:filePath instead of the one requested by the initiator.

4.2 arguments

An ordered list of strings which are the arguments provided to the program specified in **executable**.

```
<jsdl:arguments wsp:Usage="wsp:Required" wsa:Negotiability="wsa:Fixed">

      <wsa:stringElement      wsp:Usage="wsp:Required"
                              wsa:Negotiability="wsa:Fixed"
            wsa:string="-d"/>

      <wsa:stringElement      wsp:Usage="wsp:Required"
                              wsa:Negotiability="wsa:Fixed"
            wsa:string="-c"/>

      <wsa:stringElement      wsp:Usage="wsp:Required"
                              wsa:Negotiability="wsa:Fixed"
            wsa:string="myFile"/>

</jsdl:arguments>
```

The wsp:Usage and wsa:Negotiability attributes in each argument term can be omitted (their values being then inherited from the parent element jsdl:arguments).

Note: we should think about adding an executionModel concept to capture variations like posix fork vs. system shell interpretation of the command line. Then we can define more clearly how executable and arguments are used for specific execution models.

4.3 fileStageIn

A file copy operation *from* a remote machine *to* the execution host. This file copy happens *before* the execution of the job.

It is a complex term which two fields specify the paths/URLs of the source and destination files:

- remoteSource: the URL to copy from. The source file could for instance be located on the submission machine.
- localDestination: the file path on the execution host to copy to.

These two fields are defined in Schema as elements so as to be openly extensible so more sophisticated ways of specifying the file locations can be used when needed, instead of the simple wsa:filePath and wsa:fileURL attributes used in the examples below.

Examples of request:

```
<jsdl:fileStageIn wsp:Usage="wsp:Required"
                  wsa:Negotiability="wsa:Fixed">

      <remoteSource wsa:fileURL="gass://submachine:3456/data/file1"/>

      <localDestination wsa:filePath="/jobs/input/type1_data"/>

</jsdl:fileStageIn>
```

A usage pattern: specifying several file transfers using WS-Policy compositional operators. Example of a request:

```
<wsp:All wsp:Usage="wsp:Required">

      <jsdl:fileStageIn wsp:Usage="wsp:Required"
                        wsa:Negotiability="wsa:Fixed">

            <remoteSource
                  wsa:fileURL="gass://submachine:3456/data/file1"/>
            <localDestination wsa:filePath="/jobs/input/type1_data"/>

      </jsdl:fileStageIn>

      <jsdl:fileStageIn wsp:Usage="wsp:Required"
                        wsa:Negotiability="wsa:Fixed">

            . . .

      </jsdl:fileStageIn>


      <jsdl:fileStageIn wsp:Usage="wsp:Optional"
                        wsa:Negotiability="wsa:Fixed">

            <remoteSource wsa:fileURL="http::/myserver/data/file2"/>
            <localDestination wsa:filePath="/jobs/input/type2_data"/>

      </jsdl:fileStageIn>

</wsp:All>
```

In this example one of the file transfers is marked as optional: the initiator is stating that the provider can ignore that particular term when negotiating the job agreement. The wsp:All compositor mandates that all other terms be satisfied.

4.4 fileStageOut

A file copy operation *from* the execution host *to* a remote machine. This file copy happens *after* the execution of the job.

It is a complex term which two fields specify the paths/URLs of the source and destination files:

- localSource: the file path on the execution host to copy from.
- remoteDestination: the URL to copy to. The source file could for instance be located on the submission machine.

Example of request:

```
<jsdl:fileStageOut      wsp:Usage="wsp:Required"
                        wsa:Negotiability="wsa:Fixed">

      <localSource wsa:filePath="/jobs/results/data_file"/>
      <remoteDestination
            wsa:fileURL="gass://submachine:3456/data/job_results"/>

</jsdl:fileStageOut>
```

4.5 checkpointable

The presence of this term specifies that the state of the job is to be persisted. This is a complex term, with the following fields:

- **localDirectory**: indicates the directory on the end execution node where the checkpoint files will be stored.
- **period**: specifies how often automatic checkpoints of the job should be taken. It is suggested to mix in a wsa:duration attribute (of Schema type xsd:duration).

Example:

```
<jsdl:checkpointable wsp:Usage="wsp:Required"
                     wsa:Negotiability="wsa:Fixed">

      <localDirectory wsa:filePath="/jobs/checkpointing/"/>

      <period wsa:duration="1H 20M30S"/>

</jsdl:checkpointable>
```

Through open content extensibility, the content of the term as well as the content of each of its fields can be extended with a description of the checkpointing method. The method may be OS specific (e.g. cpr on IRIX), or different checkpointing mechanisms may exist on one host platform/software environment.

4.6 posixStandardInput

A file location from where the standard input of the job is redirected.

Example:

```
<jsdl: posixStandardInput wsp:Usage="wsp:Required"
                          wsa:Negotiability="wsa:Fixed"
        wsa:filePath="/jobs/input/type1_data"/>
```

4.7 posixStandardOutput

A file location where the standard output stream of the job will be placed.

4.8 posixStandardError

A file location where the standard error stream of the job will be placed.

If this term is not specified, the standard error goes to the same file as the standard output.

4.9 environment

A list of name-value pairs specifying the environment variable bindings that will be defined in the runtime environment of the job.

Example:

```
<jsdl:environment wsp:Usage="wsp:Required"
                  wsa:Negotiability="wsa:Fixed">

      <bindings>

            <jsdl:binding name="DATA_ROOT">
                  <value wsa:string="/jobs/input"/>
            </jsdl:binding>

            <jsdl:binding name="RESULTS_DIR">
                  <value wsa:string="/jobs/results/"/>
            </jsdl:binding>

            <jsdl:binding name="FOO">
                  <value wsa:string="bar"/>
            </jsdl:binding>

      </bindings>

</jsdl:environment>
```

4.10 beginTime

The time at which the job becomes eligible for dispatch.
The term framework allows for adding constraints so a time window can be specified for the beginning of the execution, e.g. an earliest beginTime or a latest (deadline) beginTime.

4.11 endTime

The time at which the job SHOULD be finished.

The term framework allows for adding constraints so a time window can be specified for the ending of the execution.

Example of an initiator-provided constrained end time:

```
<jsdl:endTime wsp:Usage="wsp:Required" wsa:Negotiability="wsa:Fixed">

      <wsa:maximum wsa:dateTime="2003-09-18T13:20:00.000-05:00"/>

</jsdl:endTime>
```

The initiator specifies the job execution will/should finish by 12/25/2003 1:20pm EST.

This is useful for example in combination with the wsa:terminateOnViolation term to abort the job if it runs too long (in LSF if the job is still running after this time, the resource manager can terminate the job).

Example of a committed term (i.e. by both the initiator and the provider):

```
<jsdl:endTime wsp:Usage="wsp:Observed">

      <wsa:maximum wsa:dateTime="2003-09-18T13:20:00.000-05:00">

            <wsa:terminationOnViolation/>

      </wsa:maximum>

</jsdl:endTime>
```

Because the provider has no control over application-initiated exits, it can not prevent a job from exiting early. But a minimum limit on the job endTime could be used to detect job failures and is therefore still a useful concept. Using the concept of monitoring criteria from WS-Agreement it could be then possible to monitor such violations. Example:

```
<jsdl:endTime wsp:Usage="wsp:Observed">

      <wsa:minimum wsa:dateTime="2003-09-18T13:00:00.000-05:00">

            <wsa:monitorViolation/>

      </wsa:minimum>

</jsdl:endTime>
```

In this example, the provider guarantees that if the job exits before the specified date/time, the WS-Agreement monitoring mechanisms will enable the initiator to know about such violation.

4.12 totalCPUConsumption

A term that specifies the amount of CPU time allowed to all the processes of the job. The attribute wsa:duration should be mixed in to express the duration value (of type xsd:duration). It may be desirable to specify CPU information so as to normalize the value.

For example the CPUReference term can be used:

```
<jsdl:totalCPUConsumption wsp:Usage="wsp:Required"
                          wsa:Negotiability="wsa:Negotiable">

      <wsa:maximum wsa:duration="20M30S"/>

      <jsdl:CPUReference>
          . . .
      </jsdl:CPUReference>

</jsdl:totalCPUConsumption>
```

A minimum constraint indicates a promised allocation for the job while a maximum constraint can limit consumption (and therefore charges) in a host with sufficient accounting mechanisms.

4.13 wallClockDuration

The wall clock run time of the batch job. Proposed xsd type for the value: xsd:duration. It can be useful to apply a minimum and/or a maximum to this value, and to specify behavior when a limit is crossed. For example, in LSF, jobs crossing this limit will be terminated.
See the endTime term for ideas of use.

Extensible content of the term should be used to specify how the duration is measured i.e.

- does it include time when the job is suspended, e.g. is strictly the difference between endTime and beginTime? Or,
- is it a timer for any time the application is ready to run?
- Is it normalized by some other criteria of the host?

4.14 runQueueLength

For time-sharing hosts, this complex term indicates the number of jobs in the operating system's run queue averaged over the duration specified by the **averagingDuration** field (for example, the number of jobs averaged over the last 15 minutes).

Not all providers will support all possible averaging durations and therefore should use constraints to express possible values. For instance, a provider acting as a façade to LSF will use the wsa:enumeration constraint to express the fact that only the durations 15 seconds, 1 minute and 15 minutes can be supported. Such provider could advertise:

```
<jsdl:runQueueLength wsp:Usage="wsp:Optional"
                     wsa:Negotiability="wsa:Negotiable">
      <length>
            <wsa:maximum wsp:Usage="wsp:Required">

                  <wsa:negotiabilityConstraints>
                        <wsa:minimum wsa:count="50"/>
                  </wsa:negotiabilityConstraints>

            </wsa:maximum>
      </length>

      <averagingDuration wsp:Usage="wsp:Required"
                         wsa:Negotiability="wsa:Negotiable">

            <wsa:negotiabilityConstraints>
                  <wsa:enumeration>
                        <wsa:duration>15S</wsa:duration>
                  </wsa:enumeration>
                  <wsa:enumeration>
                        <wsa:duration>1M</wsa:duration>
                  </wsa:enumeration>
                  <wsa:enumeration>
                        <wsa:duration>15M</wsa:duration>
                  </wsa:enumeration>
            </wsa:negotiabilityConstraints>

      </averagingDuration>

</jsdl:runQueueLength>
```

In the example above, the provider does not require the runQueueLength term but if it is specified by the initiator, then:

- an upper bound for the number of jobs in the queue must be specified. This upper bound is negotiable but must be equal to or greater than 50.
- an averaging duration must be specified. It is negotiable but must be one of the three values listed by the series of wsa:enumeration elements.

Also, some providers may want to markup the length term with additional, provider-specific content so as to specify how they represent the length (normalized or not over the number of CPUs for instance). Or, it may be better to have different terms with names such as jobRunQueueLength, processRunQueueLength, and perCPUProcessRunQueuelength terms.

Typically the **length** field sub-term should be constrained by one more min/max limits.

4.15 temporaryDiskSpace

This complex term expresses the amount of free disk space required in the given temporary **directory** (on the execution host). The unit is bytes as for all other terms related to size.

This term should be expressed as one or more limits using min/max constraints (typically a minimum limit requested by the initiator, a maximum limit offered by the provider).

Example of request:

```
<jsdl:temporaryDiskSpace wsp:Usage="wsp:Required"
                         wsa:Negotiability="wsa:Negotiable">

      <directory wsa:filePath="/scratch/job1"/>

      <size>
            <wsa:minimum wsa:amount="20000000"/>
      </size>

</jsdl:temporaryDiskSpace>
```

4.16 pagingRate

The rate of page ins and page outs on the execution host, averaged over the last minute. The logical unit is pages per second, the XML Schema type of the value being xsd:float. Use the wsa:real attribute.

This term should be expressed as one or more limits using min/max constraints.

4.17 IORate

The block rate of the execution host, averaged over the last minute. The logical unit is bytes per second, a positive integer.

This term should be expressed as one or more limits using min/max constraints.

4.18 CPUUtilization

A percentage representing how much CPU time is spent by the execution host in system and user states, versus being idle.

This term should be expressed as one or more limits using min/max constraints.

4.19 CPUDescription

Specifies a type of CPU.

It is useful when mixed-in:
- inside a **totalCPUConsumption** term in order to normalize the duration
- inside a **host** term in order to select a host according to its CPU model.

Note: This term may not be used as is but wrapped in another element providing context to the actual CPU description, so that the description string designates a CPU model unambiguously.

4.20 host

A complex term that can be used to specify an execution host. The content is left completely open-ended in order not to mandate any selection criterion for the host. Criteria for host selection might be:
- the host name. Use the jsdl:hostName attribute
- the host type, generally the operating system which is running on the execution host. For example, the jsdl:OSName attribute can be used.
- the host hardware model, generally a qualifier to the host type. It is used to distinguish different hardware models for a particular OS. It is possible to use the jsdl:CPUDescription term.

Example: preference of execution host(s) based on the OS name:

```
<wsp:OneOrMore wsp:Usage="wsp:Required">

      <jsdl:host jsdl:OSName="jsdl:Linux" wsp:Preference="1"/>

      <jsdl:host jsdl:OSName="jsdl:Windows" wsp:Preference="3"/>

      <jsdl:host jsdl:OSName="jsdl:AIX" wsp:Preference="2"/>

</wsp:OneOrMore>
```

4.21 perFileSize

A term used to set the per file size for each file created by the job. The unit is bytes as for all other terms related to size.

Only a maximum constraint sub-term is meaningful with this term. .

4.22 realMemorySize

Specifies the available memory for the job on each host (the job may run on a single host or many hosts depending on the other job terms).

Limits bound a promise of availability. Different actions on violation can select desired behavior when limits are reached, e.g. job migration, job termination, penalty assessment, etc.

4.23 virtualMemorySize

Specifies the available virtual memory for the job on each host.

The same limit semantics apply as for realMemorySize.

4.24 dataSize

A positive integer which is used as the per process (soft) data segment size for the processes belonging to the job.

Typically, a maximum constraint sub-term is applied to this term. It is sometimes desirable to specify the behavior when reaching this limit, e.g. whether page allocation fails, the job is killed, or the allocation is allowed but a violation is signaled.

Example:

```
<jsdl:dataSize wsp:Usage="wsp:Required"
               wsa:Negotiability="wsa:Negotiable">

      <wsa:maximum wsa:amount="2000000">

            <wsa:faultPreventsViolation />

      </wsa:maximum>

</jsdl:dataSize>
```

4.25 coreDumpSize

A positive integer which is used as the per process (soft) core file size for the processes belonging to this job.

Typically, a maximum constraint sub-term is applied to this term. It is probably desirable to specify the behavior when reaching this limit.

4.26 numberOfCPUs

A term used to specify the number of CPUs for execution of the job.
Example of a negotiable fixed count as advertised by a provider:

```
<jsdl:numberOfCPUs wsp:Usage="wsp:Required"
                   wsa:Negotiability="wsa:Negotiable">

      <wsa:negotiabilityConstraints>

            <wsa:minimum wsa:count="4"/>

            <wsa:maximum wsa:count="128"/>

      </wsa:negotiabilityConstraints>

</jsdl:numberOfCPUs>
```

The provider requires the initiator to specify information about the number of CPUs, which can be negotiated between 4 and 128.

After negotiation, a committed CPU count between the initiator and the provider could be:

```
<jsdl:numberOfCPUs wsp:Usage="wsp:Required"
                   wsa:Negotiability="wsa:Observed"
        wsa:count="32"/>
```

Behavioral constraints can be applied to this term to specify the minimum and/or maximum number of CPUs desired as opposed to a fixed count. In fact, certain providers might not guarantee a fixed count but only a min-max range. Similarly, a fixed count is not necessarily of interest to all initiators.
Example of a negotiable range as advertised by the provider:

```
<jsdl:numberOfCPUs wsp:Usage="wsp:Required"
                   wsa:Negotiability="wsa:Negotiable">

      <wsa:minimum>
            <wsa:negotiabilityConstraints>
                  <wsa:minimum wsa:count="4"/>
                  <wsa:maximum wsa:count="32"/>
            </wsa:negotiabilityConstraints>
      </wsa:minimum>

      <wsa:maximum>
            <wsa:negotiabilityConstraints>
                  <wsa:minimum wsa:count="64"/>
                  <wsa:maximum wsa:count="128"/>
            </wsa:negotiabilityConstraints>
      </wsa:maximum>

</jsdl:numberOfCPUs>
```

Here the provider states than it can negotiate a lower bound (as well as an upper bound) between specified values for the number of CPUs.

The initiator could then make the following matching request:

```
<jsdl:numberOfCPUs wsp:Usage="wsp:Required"
                   wsa:Negotiability="wsa:Negotiable">

      <wsa:minimum wsa:count="16"/>

      <wsa:maximum wsa:count="72"/>

</jsdl:numberOfCPUs>
```

4.27 hold

This is a flag which, if present, indicates that the job should stay in the pending state (i.e. the scheduler should not consider it for dispatch), until it is manually released.

Example:

```
<jsdl:holdForRelease wsp:Usage="wsp:Required"
                     wsa:Negotiability="wsa:Fixed"/>
```

4.28 rerunOnFailure

This complex term turns on automatic requeuing for the job in order to retry the execution in the event of job failure. It is possible to list exit codes which will cause the job to be queued again (i.e. the exit code might indicate a transient failure) and/or exit codes which will cause the resource manager to specifically exclude the execution host on which the job failed from running the job again.

- If the term is absent, the job will never be requeued on failure
- If the term is present, the job will be marked as eligible for requeueing, and:
    - With a child element jsdl:rerunOnExitCodes specifying certain exit codes, if the job exited with one of these exit codes, it will be run again (i.e. the exit code might indicate a transient failure). This element SHOULD contain the wsa:listOfIntegers attribute.
    - With a child element jsdl:rerunExceptExitCodes specifying certain exit codes, if the job exited with one of these exit codes, the execution host on which the job failed will be specifically excluded from running the job again. This element SHOULD contain the wsa:listOfIntegers attribute.

4.29 priority

This is a positive integer which helps the scheduler rank this job amongst the jobs of a given user.

4.30 priorityEscalation

This is a complex term with a **priorityOffset** field and a **wsa:interval** field. With this set, the job's priority will increase by *priority* every *interval*; where *priority* is the value of the **priorityOffset** term (a positive integer) and *interval* is the value of **wsa:interval** (an xsd:duration).

4.31 project

Assigns the job to the specified project. On some OSes (e.g. IRIX) this project must be defined in a file like /etc/project, and the user must be part of this project.

4.32 userGroup

Associates the job with the specified group. This is useful for fair share scheduling where compute shares might be assigned at a group level.

4.33 userName

This string represents the user identity under which this job should run, subject to proper authorization.

4.34 userCredentials

This string represents any credentials which should be provided to the job during its run time. Examples of credentials include Kerberos TGTs and GSI proxy certificates.

5. Considered Job Terms

5.1 jobArrayIndexList

Original idea: a string which specifies how many elements there are in a *job array*. A job array is a mechanism by which one job definition is used to submit more than one job instance at a given time. The resource management system will instantiate as many jobs in the system as there are elements in the job array. The environment of the job needs to be used to indicate to each job instance which element it is.

The syntax of the IndexList is a comma separated list of elements of the form [start[-end[:step]], where start, end and step are positive integers. If step is omitted, a step of 1 is assumed.

Comment: Ideally this term would be removed and the equivalent could be expressed as a special kind of compound job (WS-Agreement enables composition of agreements, and therefore, jobs). The array parameters would not be terms in a normal job but fields in this special compound job.

5.2 stackSize

A positive integer which is used as the per process (soft) stack segment size for each process of the batch job.
Typically, a maximum constraint sub-term is applied to this term.

Comment: We need more discussion about this term.

5.3 numberOfProcesses

Sets the number of processes for the whole job. Typically, a maximum constraint sub-term is applied to this term. It is sometimes desirable to specify the behavior when crossing this limit (example: crossing the limit will cause the job to terminate).

Comment: We need more discussion about this term.

5.4 numberOfThreads

Sets the number of concurrent threads which can be part of a job. Typically, a maximum constraint sub-term is applied to this term. It is sometimes desirable to specify the behavior when crossing this limit (example: crossing the limit will cause the job to terminate).

Comment: same as for numberOfProcesses.

5.5 suspendAction

Original idea: a string which provides a client a way to override the resource manager's default mechanism for suspending a job. The syntax is in the form "*signal* | *command* | `CHKPNT`". *Signal* is the name of any POSIX signal (e.g. SIGHUP, SIGUSR1, etc). *command* is a string which will be fed to "/bin/sh –c" (allowing you to use shell features in the command-line). `CHKPNT` is a

special token which tells the resource manager to first checkpoint the job (if it is checkpointable) before sending it a SIGSTOP signal.

Comment: needless to say, this needs to be remodeled (decomposing the string into several fields, defining constant terms or QNames for the different signals, …etc).

5.6 suspendCommand

Original idea: This string provides a client a way to override the resource manager's default mechanism for suspending a job. The syntax is in the form "*signal* | *command* | `CHKPNT`". *Signal* is the name of any POSIX signal (e.g. SIGHUP, SIGUSR1, etc). *command* is a string which will be fed to "/bin/sh –c" (allowing you to use shell features in the command-line). `CHKPNT` is a special token which tells the resource manager to first checkpoint the job (if it is checkpointable) before sending it the series of SIGINT, SIGTERM, SIGKILL signals.

Comment: same as for suspendAction.

5.7 resumeAction

Original idea: This string provides a client a way to override the resource manager's default mechanism for continuing a suspended job. The syntax is in the form "*signal* | *command*". *Signal* is the name of any POSIX signal (e.g. SIGHUP, SIGUSR1, etc). *command* is a string which will be fed to "/bin/sh –c" (allowing you to use shell features in the command-line).

Comment: same as for suspendAction.

6. Security Considerations

6.1 Job Submission

Securing the job specification and submission is outside the scope of this document which deals with the domain-specific content of the job specification.

6.2 Job Execution

Some of the job terms relate to the issue of authorization of the job execution:

- **jsdl:username**
- **jsdl:userCredentials**.

7. Author Information

Alain Andrieux
Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
Email: alain@isi.edu

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

Jason Lam
Platform Computing
Email: jlam@platform.com

Chris Smith
Platform Computing
Email: csmith@platform.com

Ming Xu
Platform Computing
Email: ming@platform.com

8. Contributors

We gratefully acknowledge the contributions made to this proposal by the following people:
Steve Tuecke (Argonne National Laboratory).

9. Intellectual Property Statement

10. Full Copyright Notice

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

[XSD Structures]
XML Schema Part 1: Structures, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, Editors. W3C Recommendation 2 May 2001.
Available at http://www.w3c.org/TR/xmlschema-1/

[XSD Datatypes]
XML Schema Part 2: Datatypes, Paul V. Biron, Ashok Malhotra, Editors. W3C Recommendation 2 May 2001.
Available at http://www.w3c.org/TR/xmlschema-1/

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

[WS-Agreement]
Note: This document still bears the old OGSI-Agreement name. A new specification will be posted soon.
Agreement-based Service Management, K. Czajkowski, A. Dan, J. Rofrano, S. Tuecke, Authors. Global Grid Forum (2003).
https://forge.gridforum.org/projects/ogsa-wg/document/OGSI-Agreement/en/1/OGSI-Agreement.pdf

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

[JSDL]
Job Submission Description Language Working Group (JSDL-WG) http://www.epcc.ed.ac.uk/~ali/WORK/GGF/JSDL-WG/

12. Appendix

12.1 Examples of Job Agreements

The structure of a job agreement itself is not within the scope of this document, which deals with the actual terms contained in a job agreement, therefore the following XML snippets all have a root wsp:All element containing job terms, but the outer element if any (maybe a jsdl:job element?) is not represented.

Terms are building blocks that can be picked and chosen freely

- by providers in order to create agreement templates
- by initiators in order to create tentative job requests matching those templates

At the end of a negotiation process between an initiator and a provider, a resulting *observed* agreement binds both parties.

12.1.1 Agreement Template (Advertised by a Provider)

```
<wsp:All>

      <jsdl:executable wsp:Usage="wsp:Required"
                       wsa:Negotiability="wsa:Negotiable">

            <wsa:useAttribute jsdl:attributeName="wsa:filePath"/>

      </jsdl:executable>

      <jsdl:arguments wsp:Usage="wsp:Optional"
                      wsa:Negotiability="wsa:Negotiable">

            <wsa:stringElement wsp:Usage="wsp:Optional"
                               wsa:Negotiability="wsa:Negotiable">

                  <wsa:useAttribute jsdl:attributeName="wsa:string"/>

            </wsa:stringElement>

      </jsdl:arguments>

      <jsdl:fileStageIn wsp:Usage="wsp:Optional"
                  wsa:Negotiability="wsa:Negotiable">

            <remoteSource>
                  <wsa:useAttribute jsdl:attributeName="wsa:fileURL"/>
            </remoteSource>

            <localDestination>
                  <wsa:useAttribute jsdl:attributeName="wsa:filePath"/>
            </localDestination>

      </jsdl:fileStageIn>

      <jsdl:numberOfCPUs wsp:Usage="wsp:Required"
                   wsa:Negotiability="wsa:Negotiable">
            <wsa:minimum>
                  <wsa:negotiabilityConstraints>
                        <wsa:minimum wsa:count="4"/>
                        <wsa:maximum wsa:count="32"/>
                  </wsa:negotiabilityConstraints>
            </wsa:minimum>

            <wsa:maximum>
                  <wsa:negotiabilityConstraints>
                        <wsa:minimum wsa:count="64"/>
                        <wsa:maximum wsa:count="128"/>
                  </wsa:negotiabilityConstraints>
            </wsa:maximum>
      </jsdl:numberOfCPUs>

</wsp:All>
```

12.1.2 Job Terms (Requested by an Initiator)

```
<wsp:All>
      <jsdl:executable wsp:Usage="wsp:Required"
                       wsa:Negotiability="wsa:Fixed"
            wsa:filePath="/bin/processData"/>

      <jsdl:arguments wsp:Usage="wsp:Required"
                      wsa:Negotiability="wsa:Fixed">
            <wsa:stringElement wsp:Usage="wsp:Required"
                               wsa:Negotiability="wsa:Fixed"
                  wsa:string="-d"/>
            <wsa:stringElement wsp:Usage="wsp:Required"
                               wsa:Negotiability="wsa:Fixed"
                  wsa:string="-c"/>
            <wsa:stringElement wsp:Usage="wsp:Required"
                               wsa:Negotiability="wsa:Fixed"
                  wsa:string="myFile"/>
      </jsdl:arguments>

      <wsp:All wsp:Usage="wsp:Required">

            <jsdl:fileStageIn wsp:Usage="wsp:Required"
                              wsa:Negotiability="wsa:Fixed">
                  <remoteSource
                      wsa:fileURL="gass://submachine:3456/data/file1"/>
                  <localDestination
                      wsa:filePath="/jobs/input/type1_data"/>
            </jsdl:fileStageIn>

            <jsdl:fileStageIn wsp:Usage="wsp:Required"
                              wsa:Negotiability="wsa:Fixed">
                  <remoteSource
                      wsa:fileURL="gass://submachine:3456/data/file2"/>
                  <localDestination
                      wsa:filePath="/jobs/input/type2_data"/>
            </jsdl:fileStageIn>

            <jsdl:fileStageIn wsp:Usage="wsp:Optional"
                              wsa:Negotiability="wsa:Fixed">
                  <remoteSource
                      wsa:fileURL="http::://myserver/data/file2"/>
                  <localDestination
                      wsa:filePath="/jobs/input/type2_data"/>
            </jsdl:fileStageIn>

      </wsp:All>

      <jsdl:numberOfCPUs wsp:Usage="wsp:Required"
                         wsa:Negotiability="wsa:Negotiable">
            <wsa:minimum wsa:count="16"/>
            <wsa:maximum wsa:count="72"/>
      </jsdl:numberOfCPUs>

</wsp:All>
```

12.1.3 Observed Terms (Binding Both Parties)

```
<wsp:All>
      <jsdl:executable wsp:Usage="wsp:Observed"
                       wsa:Negotiability="wsa:Fixed"
            wsa:filePath="/bin/processData"/>

      <jsdl:arguments wsp:Usage="wsp:Observed"
                      wsa:Negotiability="wsa:Fixed">
            <wsa:stringElement wsp:Usage="wsp:Required"
                               wsa:Negotiability="wsa:Fixed"
                  wsa:string="-d"/>
            <wsa:stringElement wsp:Usage="wsp:Required"
                               wsa:Negotiability="wsa:Fixed"
                  wsa:string="-c"/>
            <wsa:stringElement wsp:Usage="wsp:Required"
                               wsa:Negotiability="wsa:Fixed"
                  wsa:string="myFile"/>
      </jsdl:arguments>

      <wsp:All wsp:Usage="wsp:Observed">

            <jsdl:fileStageIn wsp:Usage="wsp:Observed"
                              wsa:Negotiability="wsa:Fixed">
                  <remoteSource
                      wsa:fileURL="gass://submachine:3456/data/file1"/>
                  <localDestination
                      wsa:filePath="/jobs/input/type1_data"/>
            </jsdl:fileStageIn>

            <jsdl:fileStageIn wsp:Usage="wsp:Required"
                              wsa:Negotiability="wsa:Fixed">
                  <remoteSource
                      wsa:fileURL="gass://submachine:3456/data/file2"/>
                  <localDestination
                      wsa:filePath="/jobs/input/type2_data"/>
            </jsdl:fileStageIn>

      </wsp:All>

      <jsdl:numberOfCPUs wsp:Usage="wsp:Observed"
                         wsa:Negotiability="wsa:Negotiable">

            <wsa:minimum wsa:count="16"/>

            <wsa:maximum wsa:count="72"/>

      </jsdl:numberOfCPUs>

</wsp:All>
```

12.2 JSDL Terms XML Schema (proposed)

```
<xsd:schema targetNamespace="http://www.gridforum.org/namespaces/job"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsa="http://www.gridforum.org/namespaces/agreement"
xmlns:jsdl="http://www.gridforum.org/namespaces/job" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xsd:import namespace="http://www.gridforum.org/namespaces/agreement"
schemaLocation="agreement.xsd"/>
    <xsd:element name="executable" type="wsa:TermType"/>
    <xsd:element name="arguments" type="wsa:TermType"/>
    <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>
    <xsd:complexType name="FileStageOutTermType">
        <xsd:sequence>
            <xsd:element name="localSource" type="wsa:TermType"/>
            <xsd:element name="remoteDestination" type="wsa:TermType"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="fileStageIn" type="jsdl:FileStageInTermType"/>
    <xsd:element name="fileStageOut" type="jsdl:FileStageOutTermType"/>
    <xsd:complexType name="CheckpointableTermType">
        <xsd:sequence>
            <xsd:element name="localDirectory" type="wsa:TermType"/>
            <xsd:element name="period" type="wsa:TermType"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="checkpointable" type="jsdl:CheckpointableTermType"/>
    <xsd:element name="posixStandardInput" type="wsa:TermType"/>
    <xsd:element name="posixStandardOutput" type="wsa:TermType"/>
    <xsd:element name="posixStandardError" type="wsa:TermType"/>
    <xsd:complexType name="BindingTermType">
        <xsd:sequence>
            <xsd:element name="value" type="wsa:TermType"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="binding" type="jsdl:BindingTermType"/>
    <xsd:complexType name="EnvironmentTermType">
        <xsd:sequence>
            <xsd:element name="bindings" type="wsa:TermType"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="environment" type="jsdl:EnvironmentTermType"/>
    <xsd:element name="beginTime" type="wsa:TermType"/>
    <xsd:element name="endTime" type="wsa:TermType"/>
    <xsd:element name="CPUDescription" type="wsa:TermType"/>
    <xsd:element name="totalCPUConsumption" type="wsa:TermType"/>
    <xsd:element name="wallClockDuration" type="wsa:TermType"/>
    <xsd:complexType name="RunQueueLengthTermType">
        <xsd:sequence>
            <xsd:element name="averagedOver" type="wsa:TermType"/>
```

```
            <xsd:element name="length" type="wsa:TermType"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="runQueueLength" type="jsdl:RunQueueLengthTermType"/>
    <xsd:complexType name="TempDiskSpaceTermType">
        <xsd:sequence>
            <xsd:element name="directory" type="wsa:TermType"/>
            <xsd:element name="size" type="wsa:TermType"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="temporaryDiskSpace" type="jsdl:TempDiskSpaceTermType"/>
    <xsd:element name="pagingRate" type="wsa:TermType"/>
    <xsd:element name="IORate" type="wsa:TermType"/>
    <xsd:element name="CPUUtilization" type="wsa:TermType"/>
    <xsd:attribute name="hostName" type="xsd:string"/>
    <xsd:simpleType name="OSNameType">
        <xsd:annotation>
            <xsd:documentation>Pre-defined OS names for use as values of the jsdl:OSName attribute.
            </xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:QName">
            <xsd:enumeration value="jsdl:Linux"/>
            <xsd:enumeration value="jsdl:Windows"/>
            <xsd:enumeration value="jsdl:BSD"/>
            <xsd:enumeration value="jsdl:IRIX"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:attribute name="OSName" type="xsd:QName"/>
    <xsd:element name="host" type="wsa:TermType"/>
    <xsd:element name="perFileSize" type="wsa:TermType"/>
    <xsd:element name="realMemorySize" type="wsa:TermType"/>
    <xsd:element name="virtualMemorySize" type="wsa:TermType"/>
    <xsd:element name="dataSize" type="wsa:TermType"/>
    <xsd:element name="coreSize" type="wsa:TermType"/>
    <xsd:element name="reservationHandle" type="wsa:TermType"/>
    <xsd:element name="numberOfCPUs" type="wsa:TermType"/>
    <xsd:element name="holdForRelease" type="wsa:TermType"/>
    <xsd:element name="rerunOnFailure" type="wsa:TermType"/>
    <xsd:complexType name="ExitCodesTermType">
        <xsd:sequence>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute ref="wsa:listOfIntegers"/>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="rerunOnExitCodes" type="jsdl:ExitCodesTermType"/>
    <xsd:element name="rerunExceptExitCodes" type="jsdl:ExitCodesTermType"/>
    <xsd:element name="priority" type="wsa:TermType"/>
    <xsd:complexType name="PriorityEscalationTermType">
        <xsd:sequence>
            <xsd:element name="priorityOffset" type="wsa:TermType"/>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute ref="wsa:duration"/>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:element name="priorityEscalation" type="jsdl:PriorityEscalationTermType"/>
    <xsd:element name="project" type="wsa:TermType"/>
    <xsd:element name="userGroup" type="wsa:TermType"/>
    <xsd:element name="userName" type="wsa:TermType"/>
    <xsd:element name="userCredentials" type="wsa:TermType"/>
</xsd:schema>
```

12.3 WS-Agreement Term Language XML Schema

```
<xsd:schema targetNamespace="http://www.gridforum.org/namespaces/agreement"
xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
xmlns:wsa="http://www.gridforum.org/namespaces/agreement"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified" xml:lang="en">
    <!-- Target namespace is tentative for now -->
    <xsd:import namespace="http://schemas.xmlsoap.org/ws/2002/07/utility"
schemaLocation="http://schemas.xmlsoap.org/ws/2002/07/utility"/>
    <xsd:import namespace="http://schemas.xmlsoap.org/ws/2002/12/policy"
schemaLocation="http://schemas.xmlsoap.org/ws/2002/12/policy"/>
    <xsd:import namespace="http://www.gridforum.org/namespaces/2003/03/OGSI" schemaLocation="ogsi.xsd"/>
    <!-- TODO: replace this by a WSDL import and embed this schema in the OGSI-Agreement WSDL -->
    <xsd:complexType name="AgreementType" abstract="true">
        <xsd:complexContent>
            <xsd:restriction base="wsp:PolicyExpression">
                <xsd:group ref="wsp:CompositorContent" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ContextType">
        <xsd:complexContent>
            <xsd:restriction base="wsa:TermType">
                <xsd:sequence>
                    <xsd:element name="agreementInitiator" type="xsd:anyURI" minOccurs="0"/>
                    <xsd:element name="agreementProvider" type="xsd:anyURI" minOccurs="0"/>
                    <xsd:element name="service" type="xsd:anyURI" minOccurs="0"/>
                    <xsd:element name="relatedAgreements" type="ogsi:LocatorType" minOccurs="0"
maxOccurs="unbounded"/>
                    <!-- if these 2 roles can be more abstract like security identity ,
                        we need to make each element a choice? or link to abstract
                        type? or both?-->
                    <!-- minOccurs=0 because spec says SHOULD-->
                </xsd:sequence>
            </xsd:restriction>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="MonitoringCriteriumType">
        <xsd:annotation>
            <xsd:documentation>A type meant to be extended to define criteria for monitoring agreement
terms.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <xsd:complexType name="TerminationCriteriumType">
        <xsd:annotation>
            <xsd:documentation>A type meant to be extended to define criteria for terminating agreements (ex:
who can terminate and under what conditions).</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
    </xsd:complexType>
    <!--////////////// Generic Term Language-->
    <xsd:complexType name="TermType">
        <xsd:sequence>
            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:anyAttribute processContents="lax"/>
```

```
    </xsd:complexType>
    <xsd:simpleType name="NegotiabilityType">
        <xsd:annotation>
            <xsd:documentation>Pre-defined Negotiability flags that apply to individual terms. As in the schemas following the Global XML Web Services Architecture (GXA) norms, we use QNames rather than traditional token-based enumerations.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:QName">
            <xsd:enumeration value="wsa:Fixed"/>
            <xsd:enumeration value="wsa:Negotiable"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:attribute name="Negotiability" type="wsa:NegotiabilityType"/>
    <xsd:attributeGroup name="TermAttributes">
        <xsd:annotation>
            <xsd:documentation>Factors attributes contained by TermType so can be reused as a group, for instance when defining a domain-specific term type. The Negotiability attribute on a term is used for negotiation purposes. It specifies if the individual term, as part of the agreement, can be negotiated. </xsd:documentation>
        </xsd:annotation>
        <xsd:attributeGroup ref="wsp:CompositorAndAssertionAttributes"/>
        <xsd:attribute name="Name" type="xsd:NCName" use="optional"/>
        <xsd:attribute ref="wsa:Negotiability" use="optional"/>
    </xsd:attributeGroup>
    <!--////////////// Pre-defined Terms for Constraints-->
    <xsd:element name="minimum" type="wsa:TermType"/>
    <xsd:element name="maximum" type="wsa:TermType"/>
    <xsd:simpleType name="InclusionType">
        <xsd:annotation>
            <xsd:documentation>Pre-defined inclusion flags that apply to individual limit constraints. As in the schemas following the Global XML Web Services Architecture (GXA) norms, we use QNames rather than traditional token-based enumerations.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:QName">
            <xsd:enumeration value="wsa:inclusive"/>
            <xsd:enumeration value="wsa:exclusive"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:attribute name="inclusion" type="wsa:InclusionType" default="wsa:inclusive">
        <xsd:annotation>
            <xsd:documentation>It is possible to mark a limit as inclusive or exclusive for integer-like terms (with a discrete, ordered domain) by decorating the element with the wsa:Inclusion attribute (possible values: wsa:Inclusive and wsa:Exclusive). The default is wsa:Inclusive if the wsa:Inclusion attribute is not asserted, to better match the intuitive meaning of a minimum or maximum acceptable value.</xsd:documentation>
        </xsd:annotation>
    </xsd:attribute>
    <xsd:element name="enumeration" type="wsa:TermType"/>
    <xsd:element name="negotiabilityConstraints" type="wsa:TermType"/>
    <xsd:element name="useAttribute" type="wsa:TermType"/>
    <xsd:element name="attributeName" type="wsa:TermType"/>
    <!--////////////// Pre-defined Terms for Actions related to Constraint Violations-->
    <xsd:element name="faultPreventsViolation" type="wsa:TermType"/>
    <xsd:element name="terminateOnViolation" type="wsa:TermType"/>
    <xsd:element name="monitorViolation" type="wsa:MonitoringCriteriumType">
        <xsd:annotation>
            <xsd:documentation>A flag to mark term violations for monitoring. Content can be extended dynamically to define criteria for monitoring agreement terms.</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <!--////////////// Pre-defined Attributes for Primitive Data Types-->
    <xsd:attribute name="duration" type="xsd:duration"/>
    <xsd:attribute name="dateTime" type="ogsi:ExtendedDateTimeType"/>
    <xsd:attribute name="real" type="xsd:decimal"/>
    <xsd:attribute name="amount" type="xsd:nonNegativeInteger"/>
    <xsd:attribute name="count" type="xsd:nonNegativeInteger"/>
    <xsd:attribute name="filePath" type="xsd:string"/>
    <xsd:attribute name="fileURL" type="xsd:string"/>
```

```
<xsd:attribute name="string" type="xsd:string"/>
<!--////////////// Pre-defined Attributes for List Types-->
<xsd:simpleType name="ListOfInteger">
    <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>
<xsd:attribute name="listOfIntegers" type="wsa:ListOfInteger"/>
<!--////////////// Pre-defined Terms for Array Elements-->
<xsd:element name="stringElement" type="wsa:TermType"/>
    </xsd:schema>
```