

# SLA4D<sup>2</sup>GRID

## Initial Version of D-Grid SLA D3.1

Workpackage 3 SLA Modelling

Date of publication: 15/10/2009

Editor: Wolfgang Ziegler

Version: 0.8

Status: Final

GEFÖRDERT VOM



Bundesministerium  
für Bildung  
und Forschung

## Autoren

Bastian Baranski, con terra

Dominic Battré, TU Berlin

Vincent Keller, Uni Bonn

Wolfgang Ziegler, Fraunhofer Institute SCAI

## Internes Review

Miguel Rojas, TU Dortmund

## Versionshistorie

<b>Version</b>	<b>Datum</b>	<b>Kommentar</b>	<b>Autoren</b>
0.1	25.09.09	initial draft	Vincent Keller, Wolfgang Ziegler
0.2	4.10.09	added advance reservation profile, agreement states, conclusions	Vincent Keller, Wolfgang Ziegler
0.3	7.10.09	GDI additions	Bastian Baranski
0.4	9.10.09	NextGRID SLA	Philipp Wieder
0.5	10.10.09	Proof reading	Dominic Battré
0.6	10.10.09	added BEinGRID and BREIN SLA, final editing before internal review	Wolfgang Ziegler
0.7	24.10.09	Implemented review recommendations	Wolfgang Ziegler
0.8	24.10.09	Appendix & NextGRID details added	Philipp Wieder

## Abstract

The SLA4D-Grid project aims at realising and providing a management layer for Service Level Agreements (SLA) in D-Grid. Through this layer and accompanying extensions of the current D-Grid infrastructure the existing and future D-Grid communities as well as service providers will have the opportunity to consider economical aspects when using D-Grid and to create their specific business models in D-Grid. SLA modelling is a major building block for realising this management layer for SLAs. This report gives a brief overview on some of the technology stacks available for creating SLAs and in more detail describes WS-Agreement the technology selected for creating SLAs in D-Grid. Furthermore, the report addresses some of the specific renderings for term languages to be used with WS-Agreement. This initial selection is based on the use-cases selected and is resulting from a first query the D-Grid communities participated in. Finally, the report gives an outlook on the evolvement towards the next version of the D-Grid SLA, which will include extensions to WS-Agreement towards multi-round negotiation and re-negotiation of SLAs and more and refined term languages reflecting the needs of the D-Grid communities.

## Table of contents

1	Introduction .....	5
2	Selection of the base technology for D-Grid SLAs.....	7
2.1	Existing approaches .....	7
2.1.1	Web Service Level Agreement - WSLA.....	7
2.1.2	Outcome of European projects .....	7
2.1.3	WS-Agreement .....	11
2.2	Rationales for using WS-Agreement .....	11
2.2.1	Wide distribution .....	11
2.2.2	Standard .....	11
2.2.3	Extensibility .....	12
3	WS-Agreement .....	13
3.1	Structure.....	13
3.2	Monitoring of an Agreement.....	14
3.3	WS-Agreement states .....	14
3.3.1	Agreement States .....	14
3.3.2	Service Term States .....	16
3.3.3	Guarantee States.....	17
3.4	WS-Agreement Framework.....	17
3.5	Community SLAs.....	18
3.5.1	Requirements .....	18
3.5.2	Job Submission Description Language .....	18
3.5.3	Advance Reservation Profile .....	20
3.5.4	Spatial Data Infrastructure (SDI).....	21
3.5.5	Further Community SLAs .....	24
4	Roadmap to further versions of the D-Grid SLA .....	25
5	Conclusions .....	26
6	References.....	27
Annex A	Acronyms .....	29
Annex B	SDI WS-Agreement Template.....	30

# 1 Introduction

The SLA4D-Grid project aims at realising and providing a management layer for Service Level Agreements (SLA) in D-Grid. Through this layer and accompanying extensions of the current D-Grid infrastructure the existing and future D-Grid communities as well as service providers will have the opportunity to consider economical aspects when using D-Grid and to create their specific business models in D-Grid. Figure 1 depicts the planned D-Grid SLA infrastructure. The white components or services will be provided by the SLA4D-Grid project while the grey ones are components or services the project plans to use since they are already available in the current D-Grid infrastructure. However, these components or services probably need extensions to fulfil the needs of the D-Grid SLA management layer.

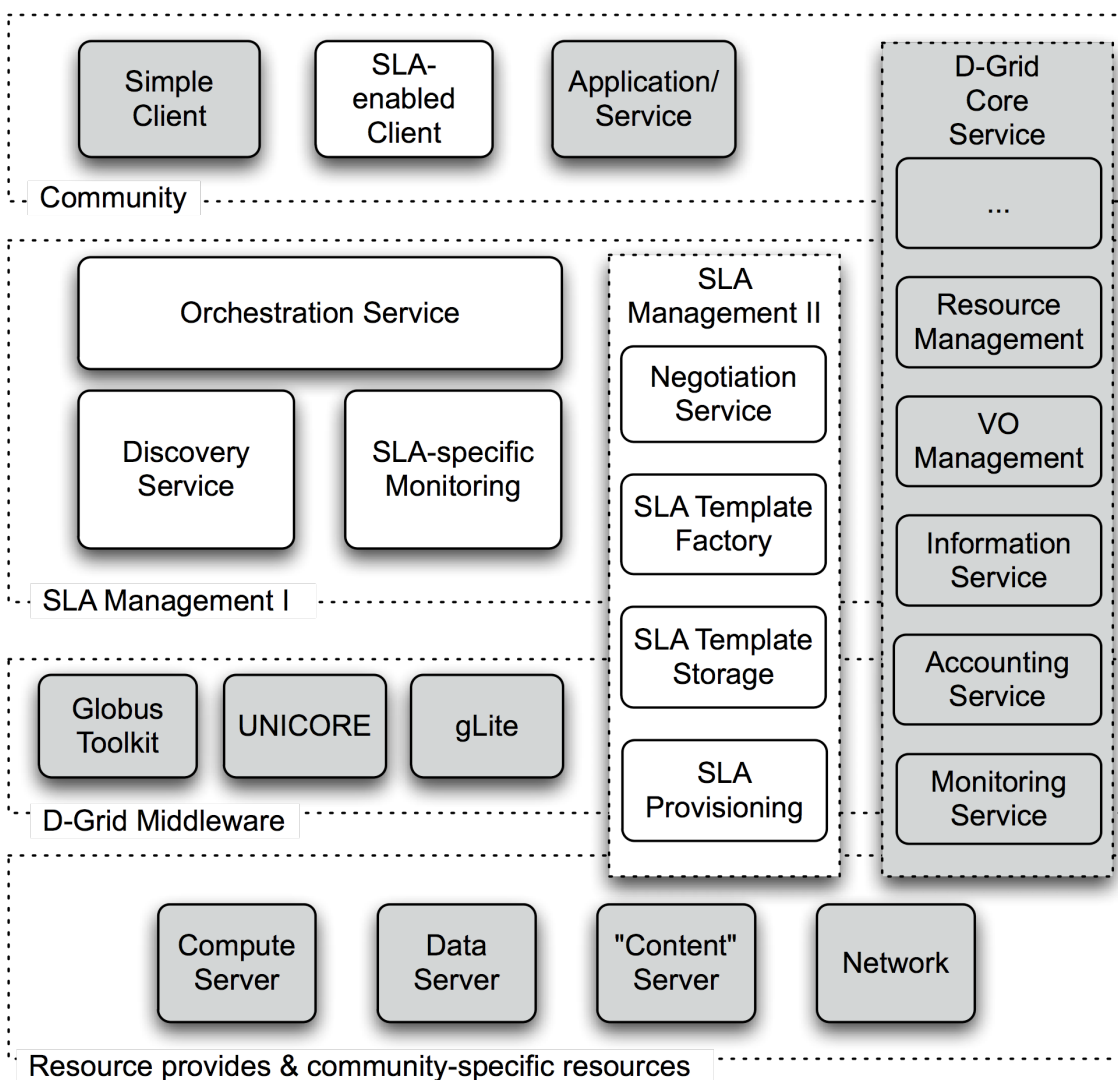


Figure 1: D-Grid SLA management layer

SLA modelling is a major issue for realising the D-Grid management layer for SLAs. It is not provided through just one single component, but distributed across and used by all

components of the architecture described in Figure 1. It is used to describe services and resources through community specific term languages, it contributes to providing the infrastructure and mechanisms to negotiate and create the agreements and it links to the external resource management, monitoring and accounting services.

In the following sections we will discuss several aspects of SLA modelling in D-Grid. Section 2 briefly presents and evaluates some of the existing approaches for the creation of SLAs. For D-Grid we considered Web Service Level Agreement (WSLA) of IBM, the NextGRID SLA (a result of the European project NextGRID), and WS-Agreement[1], the proposed recommendation of the Open Grid Forum (OGF)[3]. This section is concluded discussing the rationale for using WS-Agreement for the D-Grid SLA. Here we highlight the fact that WS-Agreement is widely used in Grids, that it is a proposed recommendation on the transition to a full recommendation of the OGF in 2009, and the extensibility of WS-Agreement, which allows to easily adapt it to the specific needs of the D-Grid communities.

Section 3 gives an overview on WS-Agreement describing the structure of WS-Agreement and the main elements. Furthermore, this section addresses some of the specific renderings for term languages to be used with WS-Agreement. This initial selection is based on the use-cases selected and is resulting from a first query the D-Grid communities participated in. Finally, in section 4 the report gives an outlook on the evolution towards the next version of the D-Grid SLA, which will include extensions to WS-Agreement towards multi-round negotiation and re-negotiation of SLAs and more and refined term languages reflecting the needs of the D-Grid communities.

## 2 Selection of the base technology for D-Grid SLAs

### 2.1 Existing approaches

#### 2.1.1 Web Service Level Agreement - WSLA

The Web Service Level Agreement (WSLA) project of IBM [8] addresses service level management issues and challenges in a Web Services environment on SLA specification, creation and monitoring. The project results comprise a language to specify Service Level Agreements that can be monitored by the service provider, customer and by a third-party, support for creating SLA templates, and a distributed monitoring framework which is separately available.

An important aspect of WSLA is its capability to deal with specifics of particular domains and technologies. The language is based on XML and extensible to include specific types of operation descriptions, (e.g., using WSDL to describe a Web services operation), measurement directive types for specific systems, special functions to compose aggregate metrics and predicates to evaluate specific metrics. The extension mechanism makes use of the ability to create derived types using XML schema. The WSLA language encompasses a set of standard extensions that allow WSLA authors to define complete agreements that relate to Web services and include guarantees for response time, throughput and other common metrics.

The WSLA development was finished around 2003. Several members of the project thereafter were also active in the WS-Agreement specification.

#### 2.1.2 Outcome of European projects

##### 2.1.2.1 NextGRID SLA

The NextGRID SLA [4] aims at providing robust, reliable, secure and performant services through QoS guarantees, while allowing providers to operate those services in an efficient and profitable manner. This happens in a heterogeneous, loosely coupled and service oriented environment made up from multiple organisations, each owning and operating a segment of the infrastructure. Therefore, the NextGRID SLA has a strong focus on the secure and trust-driven operation of SLA-enabled infrastructures. In addition, customer obligations and provider promises are captured symmetrically to cover both business parties in the same way.

The general structure of the NextGRID SLA is captured in Figure 2 and selected elements are described below. For all the details regarding the different terms please see [4].

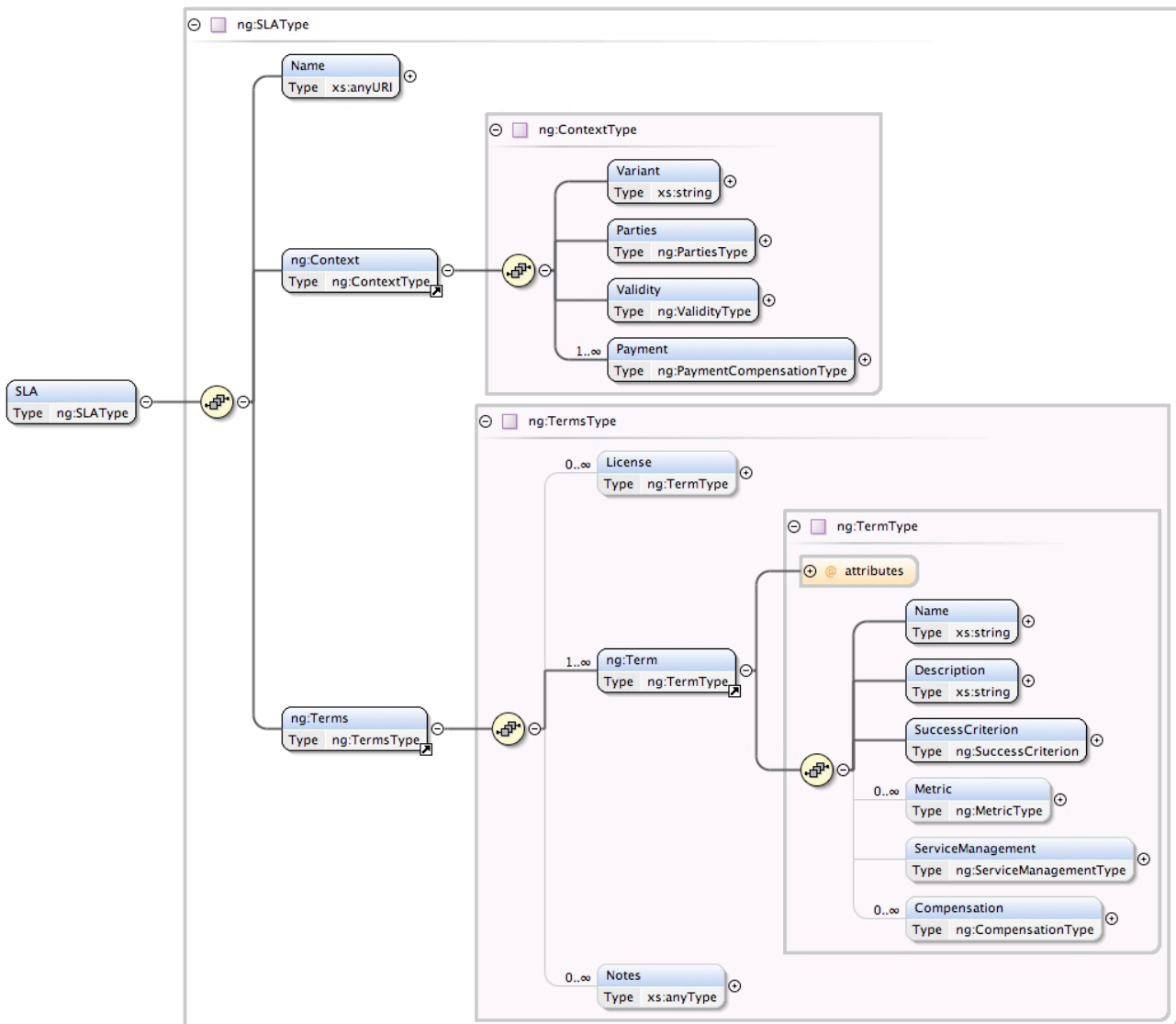


Figure 2 High-level NextGRID SLA structure

#### 2.1.2.1.1 SLA Context

The SLA Context includes a number of contractual details which set the context of the service provision. These include the SLA Variant, Agreement Parties, the Validity Period and Payment details. The purpose of the latter three elements is quite obvious (and described in detail in [4]), whereas the SLA Variant is a provider-specific element to identify different stages of an SLA. This element is applicable in cases where the provider creates new variants of an SLA (based on the agreed upon SLA) throughout the SLA lifecycle, as e.g. realized in NextGRID.

#### 2.1.2.1.2 SLA Terms

The NextGRID SLA pre-defines a number of SLA Terms, as for example License or Security terms, but in general arbitrary SLA Terms can be defined following the format shown in Figure 2 and described below.

**Name:** Every SLA Terms has a name assigned which allows it to be unambiguously identified within the scope of the SLA.

**Term Description:** Mainly for human interaction it is necessary to provide a textual description of what an SLA Term actually comprises. This is what the Term Description is for.

**Success Criterion:** The Success Criterion of every SLA Term is defined through a value (or a range of values) and the respective unit. An example for a success criterion could be the waiting time (value) of a Grid job measured in seconds (unit) or the number (value) of available software licenses (unit). Whether a Success Criterion is met or not is defined through the Metric element.

**Metric:** Metrics are used to determine whether a Success Criterion of an SLA Term is met or not. To achieve this, the address of a measurement service is provided, which monitors the Success Criterion's value and measures its success. Regarding the abovementioned examples this would e.g. assure that the waiting time is not longer than 3600 seconds or that at least 5 licenses of some software are available.

**Service Management:** Via the Service Management it is possible to specify auxiliary services which are used to manage the service that is subject of the agreement. In the NextGRID context this implies the indication of the service address and it is distinguished between resource provision services, customer escalation services, and more general management services. In general, one may predefine any service here that is used to manage SLA Terms.

**Compensation:** Compensation details are defined for cases where the Success Criterion cannot be met and no solution calling an escalation service could be found. To this end, compensation actions are defined which handle the different SLA breach variants which may occur.

The NextGRID SLA lifecycle consist of four stages

1. SLA Publishing and Discovery;
2. SLA Negotiation;
3. SLA Operation; and
4. SLA Decommissioning.

The NextGRID SLA has been used in a number of use case implementations and NextGRID demonstrators. In addition, concepts from this work have been incorporated in a several ongoing SLA initiatives. Apart from that, there is no existing open source framework based on the NextGRID SLA and the effort to develop it further is discontinued.

#### 2.1.2.2 BEinGRID SLA Cluster

BEinGRID – Business Experiments in Grid - is an ICT FP6 project of initially 75 partner organisations. The main objective of BEinGRID is to foster the adoption of Grid technologies for businesses and thereby crossing the chasm between the early market dominated by few visionary customers and the mainstream market dominated by a large number of pragmatic customers.

The BEinGRID project released 25 so called Grid Business experiments (BEs). Based on a clear business case, each BE developed a prototypic implementation for their specific requirements. Several BEs required SLAs, and provided a support for requirement analysis. The obligation to accommodate different needs produced a generic architecture for SLAs, which accommodates SLA Negotiation, SLA Evaluation, SLA accounting, as well as scheduler optimization through SLAs.

One of the BEinGRID SLA analysis lines followed the BEs that were based on GRIA (a service-oriented infrastructure designed to support B2B collaborations). These use a different SLA specification and are not considered in this document. On the other hand, three different BEs decided to use SLAs in their Globus Toolkit 4 environment. One dealt

with Mobile Fraud Management, another with dynamic capacity markets, and the last one targeted eHealth. They are described next.

#### 2.1.2.2.1 BEinGrid BE20 – Mobile Fraud Management

The main purpose of the SLA is to set the DataFederation information (included in one of the Service Description Terms (SDT)). Also, at negotiation time it is checked that the consumer server is accessible (what we call “Consumer Obligations”); this is not checked again at run-time. The telecom operators and the fraud management system have a high-level, legally binding contract, which is the one which actually regulates their business relationship.

#### 2.1.2.2.2 BEinGrid BE22 – Agrogrid

The SLA-Negotiation is provided to the user through a portlet (based on gridsphere) inside the AgroGrid portal. It serves as frontend to the BEinGRID SLA-Negotiator implementation, which allows negotiation of SLAs between capacity requester and provider (delivery of food between food-provider and food-consumer).

#### 2.1.2.2.3 BEinGrid BE25 – Business Experiment in enhanced Intensity-Modulated Radiation Therapy (IMRT) planning using Grid services on-demand (BEinEIMRT)

When a service provider is under peak demand, its resources cannot match the QoS, which has been signed in a framework agreement. To comply with this contract, an external resource provider is contacted. External resources are then added for a limited time period to the infrastructure, so the framework SLAs can be respected. The extra resources are needed on a short period (several hours to a maximum of a week). The external resources are obtained through the signature of an SLA with the external provider, using the BEinGRID SLA Negotiator component.

The negotiation is automatic and performed by the scheduler (GridWay), within the limits of the preSLAs the administrator has defined according to the framework agreement with the external provider. Furthermore, the CESGA administrator can configure some additional parameters about the terms and conditions of the negotiation.

#### 2.1.2.3 BREIN

Brein is a European FP6 project. It provides an e-business concept developed in recent Grid research, namely the concept of so-called "dynamic virtual organisations" towards a more business-centric model, by enhancing the system with methods from artificial intelligence, intelligent systems, semantic web etc. Thus, the BREIN project will enable business participants to easily and effectively use Grid technologies for their respective business needs.

The SLA management Framework is based on Globus Toolkit 4 environment and apply the GT4 components for WS-Notification, object serialization.

WS-Agreement implementation: In general, SLA Management components are based on the WS-Resource design pattern, because the component was implemented as GT4 Java WS Core service, which in turn allows the implementation of stateful services as defined by WS-RF. The SLA contracts are based on WS-Agreement specification, which defines schemas for SLA Templates.

SLA Negotiation provides two main functionalities. One is to manage Templates, which include the task of creating SLA Templates, storing the Templates in a repository and retrieve the templates. The implementation of the interfaces is suggested by WS-Agreement. The other functionality is to compare the received offers and provide the best bid to the customer.

After the negotiation, the SLA Manager will start the monitors and supervise the resource usage in order to make sure, if the agreements are met.

#### *2.1.2.4 Reference to specific solutions from other FP6 projects*

Apart from the NextGRID SLA other European projects also implemented frameworks for SLA definition and management. Most of them are based on WS-Agreement, extending WS-Agreement to satisfy the specific needs of a project. As mentioned above, notable work has been done in the BEinGRID and the BREIN projects. A summary of the work in European projects has been published as CoreGRID technical report TR-0129 [6],

### **2.1.3 WS-Agreement**

WS-Agreement is a proposed recommendation of the Open Grid Forum defined by the Grid Resource Allocation Agreement Protocol working group (GRAAP-WG) [2].

The objective of the WS-Agreement specification is to define a language and a protocol for advertising the capabilities of service providers and creating agreements based on templates, and for monitoring agreement compliance at runtime.

An agreement between a service consumer and a service provider specifies one or more service level objectives both as expressions of requirements of the service consumer and assurances by the service provider on the availability of resources and/or on service qualities.

An agreement life-cycle includes the creation, termination and monitoring of agreement states.

For example, an agreement may provide assurances on the bounds of service response time and service availability. Alternatively, it may provide assurances on the availability of minimum resources such as memory, CPU MIPS, storage or a software license.

## *2.2 Rationales for using WS-Agreement*

### **2.2.1 Wide distribution**

Since the publication of the specification as proposed recommendation, WS-Agreement has been increasingly used in the context of Grids [5], [6]. The GRAAP-WG maintains and updates a list of implementations, which is maintained and updated by the GRAAP-WG [7].

Currently, WS-Agreement is used or has been used in more than 20 different Grid-related projects we have contact to; there are 100+ using WS-Agreement in different sectors like e.g. autonomic provisioning, multi-media content negotiation or WS-Agreement Based Semantic Partner Selection. While most of them are research projects there are also developments integrating WS-Agreement in commercial software, e.g. in elasticLM, a product development for software license management in Grids.

### **2.2.2 Standard**

WS-Agreement is the only open standard specifying a language and a protocol for creating Service Level Agreements. Besides WS-Agreement only proprietary solutions are available, most of them developed in and for a certain ecosystem and no longer maintained or further developed, like e.g. the NextGRID SLA mentioned above, or available under a commercial license only, like IBM's Web Service Level Agreement - WSLA.

WS-Agreement uses the Web Services Resource Framework (WSRF) a generic and open framework for modelling and accessing stateful resources using Web services, which has been published as a standard by the OASIS consortium. Both UNICORE 6 and Globus Toolkit also are using WSRF, thus, basic interoperability for accessing stateful resources should be provided.

Moreover, several of the members of the SLA4D-Grid project are actively contributing to the GRAAP-WG and SCAI is co-chairing the working group. Thus, we can easily ensure that requirements arising during the evolvement of SLA4D-Grid find their way into the standardisation process.

### **2.2.3 Extensibility**

The intrinsic extensibility of WS-Agreement is a important feature when using it as the general technology for creating SLAs in the heterogeneous environment of D-Grid comprising 20+ communities with different requirements. Since the specification is completely neutral with respect to specific term languages to express Service Description Terms (SDT) and Service Level Objectives (SLO), D-Grid community-specific term languages can be defined within the SLA4D-Grid project and plugged into WS-Agreement. This mechanism allows providing common mechanisms for creating agreements while supporting community-specific requirements.

While the current specification of WS-Agreement only provides a basic, single round mechanism for negotiating an agreement, developments are under way at the OGF to extent the negotiation capabilities towards multi round negotiations for creating an SLA and re-negotiation of SLAs already in force.

## 3 WS-Agreement

### 3.1 Structure

An agreement consists of the agreement name, its' context and the agreement terms. The context contains information about the involved parties and metadata such as the duration of the agreement.

Agreement terms define the content of an agreement: Service Description Terms (SDTs) define the functionality that is delivered under an agreement. A SDT includes a domain-specific description of the offered or required functionality (the service itself). Guarantee Terms define assurance on service quality of the service described by the SDTs. They define Service Level Objectives (SLOs), which describe the quality of service aspects of the service that have to be fulfilled by the provider.

The Web Services Agreement Specification allows the usage of any domain specific or standard condition expression language to define SLOs. The specification of domain-specific term languages is explicitly left open. Figure 3 shows the structure of an Agreement template, which is basically the same as that of an Agreement, except that the final Agreement does not contain Agreement Creation Constraint.

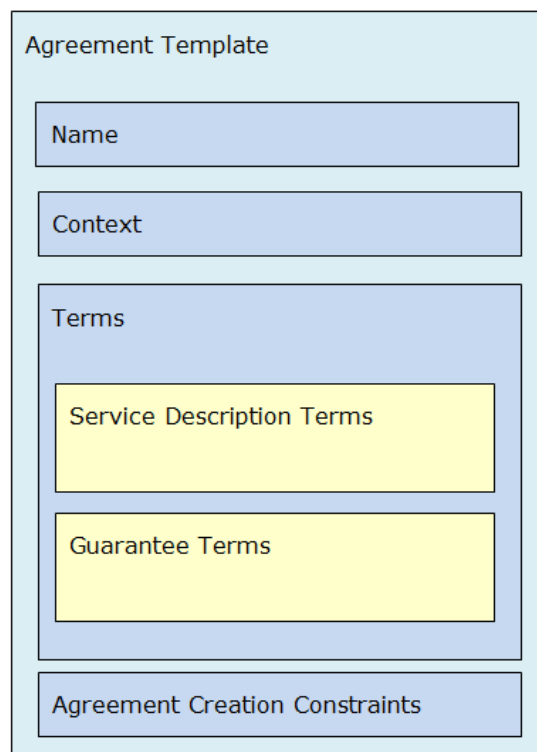


Figure 3 Agreement Template

The Agreement Creation Constrains in the template allow the agreement provider to define a number of constraints defining e.g. the ranges within the agreement initiator may modify the individual SDTs. Besides the specification of actual limits of the services provided, the

Creation Constraints help to make a negotiation of the SDTs more focussed and thus converge faster.

### 3.2 *Monitoring of an Agreement*

Following the creation of an SLA with WS-Agreement, both parties of the agreement need access to the state information of the agreement. In the case of WS-Agreement a set of resource properties is defined that allows monitoring different aspects of the agreement. This information may be retrieved through the `GetResourceProperty`-function provided by the Web Service Resource Framework (WSRF), which is used by the WS-Agreement specification. Monitoring a SLA also empowers both parties to detect potential violations of the agreement and to take appropriate measures.

The WS-Agreement specification allows the monitoring of

- Agreement state (see Figure 4)
- Service Description Terms (see Figure 5)
- Guarantee Terms (see Figure 6)

By default it is the task of the agreement responder hosting the agreement to provide the services that may gather the necessary data from the system environment to feed the information for allowing retrieving the different status information.

Data gathered from the system environment may include for example state of the scheduled job, its start time, or system monitoring data, e.g. from Ganglia. As said before, it is in the responsibility of the party hosting the agreement to set up the services to be able to extract the required information.

The SLA4D-Grid SLA management layer will provide functionality for SLA-specific monitoring while for the monitoring of services the D-Grid monitoring will be used (which probably needs some extensions of the current functionality to support service monitoring).

In case of re-negotiating the agreement already in the state *Observed* it remains in force until the re-negotiation ends successfully. Only if the re-negotiation leads to a new SLA the new one supersedes the currently active SLA: the latter changes its state to *Terminated* and the new one changes its state from *Pending* to *Observed*. Since the new SLA also has a new EPR this has to be used for the SLA-specific monitoring.

### 3.3 *WS-Agreement states*

The proposed recommendation WS-Agreement version 1.0 defines three classes of states which can be used to monitor the state of the Agreement as a whole, the state of individual service description terms, and the state of the guarantee terms.

#### 3.3.1 **Agreement States**

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

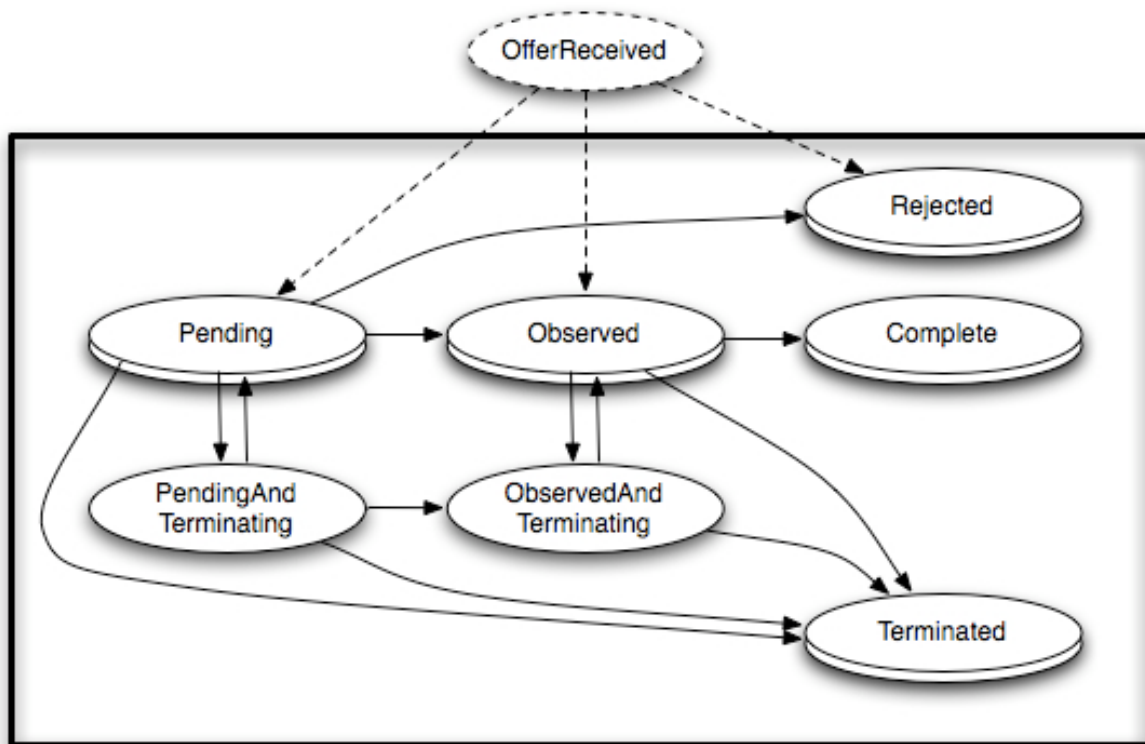


Figure 4 Agreement states, as proposed in the WS-Agreement protocol version 1.0

*Pending*, *PendingAndTerminating*, *Observed*, *ObservedAndTerminating*, *Rejected*, *Complete* and *Terminated* are the normative primary states of an Agreement State. Each state can be extended with one or more sub-states in a specific usage domain. *OfferReceived* is an initial transition state (that is not exposed to the initiator — represented by the dashed lines) to clarify that exposed initial states can be "*Pending*", "*Observed*" or "*Rejected*".

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

has been issued by the Agreement Initiator and is being processed. This state MAY follow Pending. This state MAY be followed by the Pending state in a case where a termination request is made but not accepted by the responder.

- *ObservedAndTerminating*. This state means that that an Agreement offer has been made and accepted. Furthermore, a Terminate operation has been issued from the Agreement Initiator and is being processed by the Agreement Responder. This state MAY follow Observed or PendingAndTerminating. This state MAY be followed by the Observed state in a case where a termination request is made but not accepted by the responder.

### 3.3.2 Service Term States

The service term state observes the following state model:

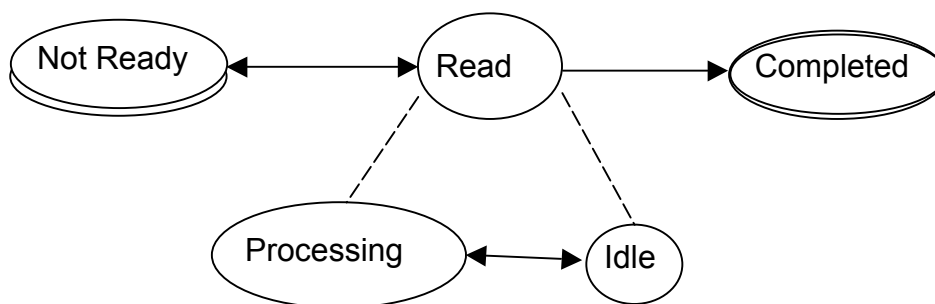


Figure 5 State model of the service term states

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

The semantics of the states is as follows:

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

*NotReady* is the initial state of a service description term while the service is being activated or provisioned. Once a service is ready, it may cycle through the periods of active use and idling, represented by the sub-states of *Processing* and *Idle*, respectively. Once a service is completed and cannot be reused further, the service description term reaches the terminal state, marked *Completed*. It is important to understand, that the *Completed* state is reached regardless for which reason the service is completed, e.g. it ended normally fulfilling the request, it failed for some reason, the user aborted it. It is up

to the service monitoring to figure out the reason and to eventually initiate corrective measures.

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

### 3.3.3 Guarantee States

The guarantee states follow a simple state model:

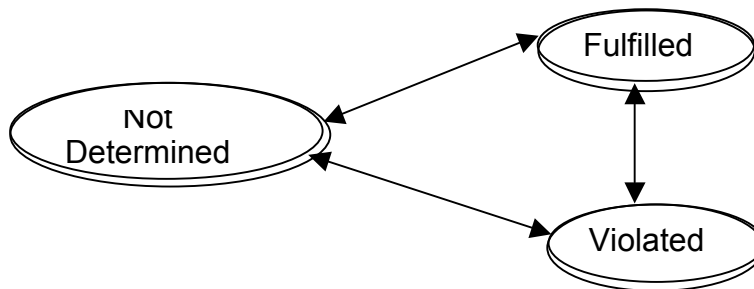


Figure 6 State model of the guarantee states

The semantics of the states are as follows:

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

*NotDetermined* is the initial state of a guarantee term, until a service is invoked or fulfilled and assessment is made. Depending on the assessment the terminal state can be either *Fulfilled* or *Violated*. For guarantee terms that require recurring assessment, the term state after every assessment period may be in *Fulfilled*, *Violated* or *NotDetermined* state. If there was no service activity in the preceding window, then the term state will be in *NotDetermined* state.

## 3.4 WS-Agreement Framework

The SLA management layer of D-Grid will base on, extent, and port existing implementations of WS-Agreement to benefit from experience already made, e.g. described in [10]. One of the most complete and advanced implementation of the WS-Agreement specification is WS-Agreement for Java (*WSAG4J*)[12].

The WS-Agreement implementation for Java is a framework that intends to provide an easy to use framework to create, monitor and terminate service level agreements based on the WS-Agreement specification.

To accomplish this goal, the framework implements the basic features of the WS-Agreement protocol and uses a number of standards in conjunction with WS-Agreement. Hence WSAG4J provides a complete development framework for SLA based services.

The framework comprises features such as the WS-Agreement `AgreementFactory` port type and the WS-Agreement `AgreementState` port type. It makes use of digital signatures to enforce message integrity (WS-Security) and also provides the listing of existing agreements via WS-Service Groups.

Before executing an agreement, the agreement offers are validated based on template creation constraints. The validation is also available during the negotiation process, when validating negotiation offers.

As described in the WS-Agreement specification, an agreement factory exposes an operation for creating an agreement out of an initial set of terms and returns an Endpoint Reference (EPR) to an `Agreement` service.

The agreement factory also exposes resource properties like templates – representations of acceptable offers for the creation of an agreement.

To create an agreement, a client makes an agreement offer which has exactly the same structure as the final agreement. For this purpose the client receives beforehand a list of templates the factory exposes.

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

The agreement factory provides also the capability of negotiating templates, which is an improvement of the initial WS-Agreement protocol.

To monitor an agreement, several states were proposed in the WS-Agreement specification. Those states are described in the following section.

## 3.5 Community SLAs

### 3.5.1 Requirements

During a face-to-face meeting in Sankt Augustin, September 1-2, 2009, initial requirements towards the D-Grid SLA have been presented as identified in the two SLA4D-Grid use-cases. As a common requirement of both use-cases the availability of a term language to express properties of a job was requested. Using the OGF standard Job Submission Description Language (JSDL) [11] has been considered as the best way to achieve this. Furthermore, for the Spatial Data Infrastructure (SDI) additional requirements have to be fulfilled as described below. Another requirement to be addressed by D-Grid SLA version I is providing the possibility to express specifications for the time frame of a job execution or resource reservation. The Advance Reservation Profile described below will be providing the basic functionality to satisfy these kind of requirements.

### 3.5.2 Job Submission Description Language

#### 3.5.2.1 Motivation and objective

The Job Submission Description Language (JSDL) is a language for describing the requirements of computational jobs for submission to resources, particularly in Grid environments, though not restricted to the latter. The JSDL language contains a

vocabulary and normative XML Schema that facilitate the expression of those requirements as a set of XML elements. In particular, it can be used as term language for describing job-related SDTs inside WS-Agreement.

Using JSDL already provides an environment with a high level of interoperability for the description of SDTs because the rationale behind the JSDL specification was to overcome incompatibilities in describing job parameters and environments as stated in the specification: Many organizations accommodate a variety of job management systems, where each system has its own language for describing job submission requirements. This makes interoperability between these systems for job management difficult. In order to utilize all these different systems these organizations have to prepare and maintain a number of different job submission documents, one for each system, and all describing the same submission. Using a standardized language, such as JSDL, which can be easily mapped to the various systems, would clearly alleviate this problem.

### 3.5.2.2 JSDL Document Structure

A JSDL document is described using XML and adheres to the normative XML schema contained in Appendix 1 of the JSDL specification [11].

A JSDL document is organized as follows: The root element `JobDefinition` contains a single mandatory child element named `JobDescription`. The `JobDescription` contains elements that describe the job: `JobIdentification`, `Application`, `Resources`, and `DataStaging`. The pseudo schema definition follows:

```
<JobDefinition>
  <JobDescription>
    <JobIdentification ... />?
    <Application ... />?
    <Resources ... />?
    <DataStaging ... />*
  </JobDescription>
  <xsd:any##other/>*
</JobDefinition>
```

Complete examples of JSDL documents are in Appendix 3 of the JSDL specification.

The JSDL specification does not define default values for elements not present in a JSDL document. (Typically the values of such elements are left up to the consuming system. Refer to the specification of each element for more details.) The JSDL elements needed to describe a specific job submission are expected to be present, with appropriate values, in a JSDL document.

All elements present in a JSDL document must be satisfied for the entire document to be satisfied. It is, of course, possible to construct JSDL documents that cannot be satisfied. For example, a JSDL document that contains contradictory resource requirements cannot be satisfied.

Adding further details of the JSDL specification is out of scope of this document. For more information readers should refer to the OGF document GDF.136.

### 3.5.2.3 JSDL usage

Most of the projects using WS-Agreement for implementing a framework for SLA creation also use JSDL when describing SDTs for a computational job. A number of publications describing best practice are available from these projects and can be used as reference when starting to implement the D-Grid SLA.

## 3.5.3 Advance Reservation Profile

### 3.5.3.1 Motivation and objective

Many scenarios require restricting the time frame of a job execution or resource reservation. Examples are scenarios where a human aims at reserving resources during his working hours or such scenarios where different resources should be reserved for the same time (co-allocation) or in a sequence.

### 3.5.3.2 Proposal for an Advance Reservation profile

The profile for advance reservation presented here is currently discussed in the GRAAP working group of the OGF to reach consensus about the profile prior to publishing it as an OGF document. Thus, in later versions of the D-Grid SLA this profile still might be modified to adapt to the official publication.

The goal of this Advance Reservation Profile is introducing a minimal number of types, which allow simple reservations. More complex reservations, for example as supported by the iCalendar format (e.g. “each day of the week from 10:00 to 12:00” or “each first Monday of a month from 10:00 to 12:00”), are out of scope since these would raise a number of new questions related with resource scheduling.

The definition of the types should allow determining a time slot within which a reservation can be allocated (`AllocationTimeConstraint`). It is not determined whether the reservation fills the entire time slot or whether the time slot just described the boundaries of a reservation. This can be decided when the duration of the reservation is specified separately.

Example 1:

A time slot could limit a reservation of 4 hours to the interval [6:00, 12:00]. Then every starting time within [6:00, 8:00] would be valid.

Example 2:

A time slot [6:00, 12:00] without specification of the duration of a reservation should be taken as a reservation with a duration of 6 hours starting at 6:00.

A time slot always should be interpreted as closed interval. However, it might be unlimited if required depending on the context.

Besides specifying boundaries for the reservation time, also a concrete reservation time may be specified (`AllocationTime`). This can be necessary for example when specifying the effective reservation time e.g. as a response to a query or exposing it as a property of a concluded SLA.

### 3.5.3.3 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ar="http://www.ogf.org/graap/AR/2009/04/"
  targetNamespace="http://www.ogf.org/graap/AR/2009/04/"
  attributeFormDefault="qualified"
  elementFormDefault="qualified">

  <xs:complexType name="TimeWindowType">
    <xs:sequence>
      <xs:element ref="ar:StartTime" minOccurs="0"/>
      <xs:element ref="ar:EndTime" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:element name="StartTime" type="xs:dateTime" />
  <xs:element name="EndTime" type="xs:dateTime" />
  <xs:element name="AllocationTimeConstraint" type="ar:TimeWindowType" />
  <xs:element name="AllocationTime" type="ar:TimeWindowType" />

</xs:schema>
```

### 3.5.3.4 Future work

The profile for Advance Reservations described before already may be used in a large number of scenarios (planning-based scheduling, co-allocation, reservation of licenses, etc.). More complex scenarios may require more complex profiles with expressiveness similar to that of iCalendar. It is up to future investigation identifying the necessity and their representation as XML.

## 3.5.4 Spatial Data Infrastructure (SDI)

This section introduces the concept of Spatial Data Infrastructures (SDI) with a focus on the Open Geospatial Consortium (OGC) Web Processing Service (WPS) specification. Additionally, this section gives a brief overview about the resource reservation and application description requirements from the SDI-scenario perspective. Finally, this section provides a very first draft of a simple SLA-template document that matches the presented SLA requirements.

### 3.5.4.1 Motivation and objective

With the advent of the Service-Oriented Architecture (SOA) paradigm and general advancements in Web Service technology, Geographical Information Systems (GIS) underwent a substantial change from stand-alone applications to distributed service architectures manifested in Spatial Data Infrastructures (SDI) [13]. While existing SDIs are mainly focused on geodata retrieval and geodata portrayal [14] by means of Web Services standards created by the Open Geospatial Consortium (OGC), the geodata processing is still done by human actors with more or less proprietary and monolithic GIS. With increasing availability of network bandwidth, server side processing capacity and advancements regarding the standardization of Web Service technologies, it now becomes feasible to process geodata in a SDI.

The OGC Web Processing Service (WPS) interface specification [15] became an official standard in mid 2007 and is a major attempt to address geospatial processes in a standardized way. The WPS interface specification defines a standardized approach to publish and perform geospatial processes over the web. Such a process can range from a simple geometric calculation (for example a simple spatial intersect operation) to a complex simulation process (for example rain water discharge or a complex Multicriteria Evaluation in Spatial Decision Support Systems). The WPS interface is based on three operations. The *GetCapabilities* operation provides service metadata and general information about the offered processes. The process metadata including the corresponding input- and output-parameters are provided by the *DescribeProcess* operation. Via the *Execute* operation it is possible to run an offered process. The implementation of specific processes must be realized by the service provider itself.

However, highly specialized geospatial applications based on large volumes of distributed data such as live sensor data streams at different scales combined with high resolution geodata, which have to be analyzed in real-time for risk management issues, often require the functionality of multiple processes. To improve the computational performance of processing such large amounts of dynamic geodata, Grid Computing provides appropriate tools. Although the application of Grid Computing is not novel to the mainstream IT-world, in the context of geospatial applications and OGC Web Services (OWS) only little research has been conducted; see [16], [17] and [18].

The GDI-Grid<sup>1</sup> project has started focusing on the integration and processing of geospatial data and services in a Grid infrastructure. In particular, it addresses the implementation of security mechanisms and definition of service interfaces for Grid-enabled OWS. The purpose of SEE-GEO<sup>2</sup> project is to provide access to the EDINA national data center hosting geospatial services running on the UK National Grid Service. The integration and enforcement of SLA is not addressed in these selected research projects.

The overall goal of the SDI-scenario within the SLA4D-Grid project is to integrate SLAs into OGC-based SDI and to utilize Grid Computing (namely the D-Grid infrastructure) for distributed and parallel processing of geodata via the WPS interface under the terms of a previously negotiated SLA. Therefore, the SDI-scenario could be seen in some parts as an extension and continuation of the use-cases from the GDI-Grid project. The concepts and implementations from the SDI-scenario will be demonstrated in a real-world scenario. This real-world scenario will be grounded either by the governmental water and land management domain or by a decision support system in case of natural disasters.

#### 3.5.4.2 Resource Reservation Requirements

The Execute operation of the WPS calls one specific process that is offered from a WPS instance. If the called process uses a Grid infrastructure for distributed and parallel computation, the Execute operation causes one or several jobs in the D-Grid infrastructure. Each process offered by a WPS acts as the representative of a specific geospatial application that will be executed in the Grid. In the SDI-scenario the application executables are either already deployed on the computation nodes or will be deployed on an on-demand basis during the Execute operation call.

---

<sup>1</sup> <http://www.gdi-grid.de/>

<sup>2</sup> <http://edina.ac.uk/projects/seesaw/index.html>

Before a job can be performed via the Execute operation, a SLA must be negotiated to define the environment in which the geospatial application should be executed. Two scenarios for the negotiation process were identified.

In the first case, the user initiates the SLA negotiation process on an on-demand basis before executing a WPS process. The user requests the WS-Agreement template document from the SLA management layer and specifies the time frame of the job execution. The time frame will be a “short” period of time located in the “near” future. Additionally, the user specifies the corresponding resource requirements (for example the numbers of CPUs and the total amount of disk space) that are required to execute the WPS process and the corresponding geospatial application in the defined time frame. After the negotiation process is finished, the user executes the WPS process (and therefore initiates one or more jobs that are submitted to the D-Grid infrastructure) under the terms of the previously negotiated SLA once only. Such a scenario could be seen as a “one-time charge” business model.

The introduced Advanced Reservation Profile definitely could be used to define the time frame requirements within a WS-Agreement (template) document. The modelling of the resource reservation requirements with JSDL are described in the next section.

In the second case, the user initiates and performs the SLA negotiation process in the same way as described for the first case. But the time frame and resource reservations for the job execution are defined for a “long” period of time. In this more or less long period of time, the user would like to execute a WPS process more than once and whenever he wants - without having to negotiate a new SLA. Such a scenario could be seen as a “flatrate” business model.

The introduced Advanced Reservation Profile potentially could be used to define these time frame and corresponding resource reservation requirements within a WS-Agreement (template) document. Maybe some extension must be developed for the Advanced Reservation Profile to realize all the requirements from the “flatrate” scenario.

#### 3.5.4.3 JSDL Requirements

The geospatial application (including the application- and user-specific resource reservation requirements) could be described adequately by a JSDL description within the WS-Agreement (template) document.

Even though the `<jsd1:JobName>` element is not unique to a particular JSDL document, it is used to identify a job (a running instance of an geospatial application) to the user.

The `<jsd1:ApplicationName>` and the `<jsd1:ApplicationVersion>` elements define the name and the version of the geospatial application to be executed. These elements identify the application independently of the location of its executable on the host. Each process offered by a WPS acts as the representative of an application described by these two elements.

The `<jsd1:FileSystem>` element describes filesystems that are required by a job. Some well-known filesystem names are proposed by the JSDL specification. It is supposed that geospatial applications have access to filesystems similar to the “HOME”, “SCRATCH” and “TEMP” filesystems that are proposed by the JSDL specification. The sub-element `<jsd1:MountPoint>` describes a local location under which each specified filesystem must be available. The `<jsd1:DiskSpace>` sub-element describes the required amount of disk space on the containing filesystem.

The `<jsd1:OperatingSystem>` element could be used to identify an operating system environment in which the geospatial application will be executed. In the SLA4D-Grid project this element potentially could be used to identify prepared Virtual Machine images with a specific operating system (e.g. Linux), pre-installed geospatial software packages (e.g. the Open Source GRASS GIS) and pre-installed or embedded datasets (e.g. European climate data).

The `<jsd1:IndividualCPUCount>` element is a range value specifying the number of CPUs for each of the resources to be allocated to the job submission. The `<jsd1:IndividualCPUSpeed>` element is a range value specifying the speed of each CPU required by the job in the execution environment. The `<jsd1:IndividualCPUTime>` element is a range value specifying the total number of CPU seconds required on each resource to execute the job. All these elements could be used by the user in the SLA negotiation process to specify the application- and user-specific resource requirements to perform the geospatial application in a convenient manner. All the corresponding `<jsd1:Total*>` elements could be used in the same way, to specify resource requirements or constraints for resource consumption.

All these more or less abstract JSDL requirements will be evaluated in the next phase of this project under the terms of the new (and not finally specified) SLA layer and the utilized Grid middleware. Maybe some of the presented elements are not required or supported within the D-Grid infrastructure, in the context of the SLA4D-Grid project, at the Grid middleware level or for example by specific batch schedulers.

#### *3.5.4.4 Example Agreement*

In Annex B you will find attached an example WS-Agreement document template for the SDI-scenario. This example document will be used as a starting point for further conceptual developments.

### **3.5.5 Further Community SLAs**

Further Community SLAs will be driven by more detailed requirements of the second use-case scenario and by additional requirements of the D-Grid communities.

## 4 Roadmap to further versions of the D-Grid SLA

After providing the implementation of the initial version of WS-Agreement the use-cases selected for SLA4D-Grid will enter a first series of experiments. These experiments will deliver requests for additional or modified term languages or elements of them.

These requirements will be implemented in version II of the D-Grid SLA, which will be deployed in month 22 of the project. Moreover, at that time the D-Grid SLA will already have been integrated with the Prototype I of the SLA layer and the integration with the Prototype II will be under way. Given the current perspective of the WS-Agreement-Negotiation specification development in the GRAAP-WG of the OGF we expect to also have a first version of WS-Agreement-Negotiation available in the version II of the D-Grid SLA, which will be described in deliverable D3.2. In support of this objective members of the SLA4D-Grid project will actively participate in the specification, also bringing potential requirements stemming from the SLA4D-Grid project into the specification developments.

At month 30, work package 3 will release the final version of the D-Grid SLA, which will be presented in deliverable D3.3. Since at month 30 also the final version of the D-Grid SLA layer is delivered, the integration with this final version will also be under way.

## 5 Conclusions

This document describes the required components of the D-Grid SLA version I. Based on the requirements identified in several face-to-face meetings we give the rationale for selecting

- WS-Agreement for Java as the framework to be used for SLA creation, together with
- JSDL, the Job Submission Description Language, used as term language to describe the Service Description Terms of computational jobs,
- terms defined to create Agreements including Data from the Spatial Data Infrastructure (SDI), and
- the Advance Reservation Profile delivering the terms for expressing time constraints of a reservation

followed by a brief description of the current state of the components from which the development and implementation work will start.

The document also briefly discusses aspects of SLA monitoring, which partly have to be resolved together with the providers of the D-Grid Monitoring while transitioning from the D-Grid SLA version I to version II.

In the last section an outlook is given on the roadmap for transitioning from version I via an intermediate version II to the final version of the D-Grid SLA.

## 6 References

- [1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Kakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu: Web Services Agreement Specification (WS-Agreement), GFD.107. <http://www.ogf.org/documents/GFD.107.pdf>
- [2] GRAAP-WG. <http://forge.gridforum.org/sf/projects/graap-wg>
- [3] OGF. <http://www.ogf.org/>
- [4] NextGRID SLA Schema Generalized Specification. Available at: [http://www.nextgrid.org/GS/management\\_systems/SLA\\_management/NextGRID\\_SLA\\_schema.pdf](http://www.nextgrid.org/GS/management_systems/SLA_management/NextGRID_SLA_schema.pdf).
- [5] Jan Seidel; Oliver Wäldrich; Philipp Wieder; Ramin Yahyapour; Wolfgang Ziegler: Using SLA for Resource Management and Scheduling - A Survey. Grid 2007, Workshop on Using Service Level Agreements in Grids, Austin, USA, September 2007, Springer 2008, CoreGRID Series 8, P. 335 - 347.
- [6] Parkin, M., Badia, R, Martrat, J.: A Comparison of SLA Use in Six of the European Commissions FP6 Projects, CoreGRID Technical Report TR-0129, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence, <http://www.coregrid.net/mambo/images/stories/TechnicalReports/tr-0129.pdf>
- [7] GRAAP-WG, Open Grid Forum: List of WS-Agreement Implementations <https://forge.gridforum.org/sf/wiki/do/viewPage/projects.graap-wg/wiki/Implementations>
- [8] IBM Web Service Level Agreements (WSLA) Project: <http://www.research.ibm.com/wsla/>
- [9] Battré, D., Wieder, P., Ziegler, W., WS-Agreement Experience Report, Open Grid Forum,
- [10] Battré, D., Havestadt, M., Wäldrich, O., Lessons learned from implementing WS-AGREEMENT, Grid 2009, IEEE Workshop on Service Level Agreements in Grids, Springer, CoreGRID Series, to appear.
- [11] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, A. Savva: Job Submission Description Language (JSDL) Specification, Version 1.0, GFD.136. <http://www.ogf.org/documents/GFD.136.pdf>
- [12] Oliver Wäldrich: WS-Agreement for JAVA (WSAG4J), <http://packcs-e0.scai.fraunhofer.de/wsag4j/>
- [13] Masser, I (2005). GIS worlds : Creating spatial data infrastructures. 1st ed. ESRI Press, Redlands, California.
- [14] Kiehle, C, Greve, K, Heier, C (2007). Requirements for Next Generation Spatial Data Infrastructures-Standardized Web Based Geoprocessing and Web Service Orchestration. In: Transactions in GIS. 11(6):819–834.
- [15] OGC (2007). OpenGIS Web Processing Service (1.0.0). OGC 05-007r7.

- [16] Baranski, B (2008). Grid Computing Enabled Web Processing Service. GI-Days 2008, Münster, Germany.
- [17] Lanig, S, Schilling, A, Stollberg, B, Zipf, A (2008). Towards Standards-based Processing of Digital Elevation Models for Grid Computing through Web Processing Service (WPS). The 2008 International Conference on Computational Science and its Applications (ICCSA2008), Perugia, Italy.
- [18] OWS-6 (2009). OWS-6 WPS Grid Processing Profile Engineering Report. OGC 09-041.

## Annex A Acronyms

EPR	End-point reference
GRAAP-WG	Grid Resource Allocation Agreement Protocol working group
JSDL	Job Submission Description Language
OASIS	Organization for the Advancement of Structured Information Standards
OGF	Open Grid Forum
SDI	Spatial Data Infrastructure
SDT	Service Description Term
SLA	Service Level Agreement
SLO	Service Level Objective
UNICORE	Uniform Interface to Computer Resources
WS-Agreement	Web Services Agreement
WSAG4J	WS-Agreement for Java
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
XML	Extended Markup Language

## Annex B SDI WS-Agreement Template

The following SLA Template captures the requirements from the Spatial Data Infrastructure usage scenario as introduced in Section 3.5.3.1. It relies on a number of OGF standards, which are: (i) the standardised WS-Agreement XML schema [1], the standardised JSDL XML schema [11] and the recommendation for an Advance Reservation schema as described in 3.5.3.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Template xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
xmlns:jSDL="http://schemas.ggf.org/jSDL/2005/11/jSDL"
xmlns:ar="http://www.ogf.org/graap/AR/2009/04/"
xmlns:ows="http://www.opengis.net/ows/1.1" wsag:TemplateId="wfs-template"
xsi:schemaLocation="http://schemas.ggf.org/graap/2007/03/ws-agreement ./ws-
agreement.xsd http://schemas.ggf.org/jSDL/2005/11/jSDL ./jSDL.xsd
http://www.ogf.org/graap/AR/2009/04/ ./ar.xsd">

<wsag:Name>
  WS-Agreement Specification template for OGC Web Feature Service (WFS)
</wsag:Name>

<!-- This element contains various metadata about the agreement. -->
<wsag:Context>
  <wsag:AgreementInitiator>
    <ows:IndividualName>Bastian Baranski</ows:IndividualName>
    <ows:PositionName>Wissenschaftlicher Mitarbeiter</ows:PositionName>
    <ows:ContactInfo>
      <ows:Phone>
        <ows:Voice>+49 251 / 83 - 33 0 71</ows:Voice>
        <ows:Facsimile>+49 251 / 83 - 39 7 63</ows:Facsimile>
      </ows:Phone>
      <ows:Address>
        <ows:DeliveryPoint>Weseler Straße 253</ows:DeliveryPoint>
        <ows:City>Münster</ows:City>
        <ows:AdministrativeArea/>
        <ows:PostalCode>48151</ows:PostalCode>
        <ows:Country>Germany</ows:Country>
        <ows:ElectronicMailAddress>
          Bastian.Baranski@uni-muenster.de
        </ows:ElectronicMailAddress>
      </ows:Address>
      <ows:OnlineResource/>
      <ows:HoursOfService/>
    </ows:ContactInfo>
    <ows:Role>Wissenschaftlicher Mitarbeiter</ows:Role>
  </wsag:AgreementInitiator>

  <wsag:AgreementResponder>
    <ows:IndividualName>Bastian Baranski</ows:IndividualName>
    <ows:PositionName>Software Developer</ows:PositionName>
    <ows:ContactInfo>
      <ows:Phone>
        <ows:Voice>+49 251 / 7474 - 301</ows:Voice>
        <ows:Facsimile>+49 251 / 7474 - 100</ows:Facsimile>
      </ows:Phone>
    </ows:ContactInfo>
  </wsag:AgreementResponder>
</wsag:Context>
</wsag:Template>
```

```

    <ows:Address>
      <ows:DeliveryPoint>Martin-Luther-King-Weg 24</ows:DeliveryPoint>
      <ows:City>Münster</ows:City>
      <ows:AdministrativeArea/>
      <ows:PostalCode>48155 </ows:PostalCode>
      <ows:Country>Germany</ows:Country>
      <ows:ElectronicMailAddress>
        b.baranski@conterra.de
      </ows:ElectronicMailAddress>
    </ows:Address>
    <ows:OnlineResource/>
    <ows:HoursOfService/>
  </ows:ContactInfo>
  <ows:Role>Software Developer</ows:Role>
</wsag:AgreementResponder>

<wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
<wsag:ExpirationTime>2009-12-12T12:13:14Z</wsag:ExpirationTime>
<wsag:TemplateId>conterra.wps.simplebuffer.1</wsag:TemplateId>
<wsag:TemplateName>
  Default WSAG Template for OGC Web Processing Service (WPS)
</wsag:TemplateName>
</wsag:Context>

<!-- This element comprises one or more service definition and guarantee
terms. -->
<wsag:Terms>
  <wsag:All>
    <!-- This element provides an inline full or partial functional
description of a service. -->
    <wsag:ServiceDescriptionTerm wsag:Name="APPLICATION"
wsag:ServiceName="SIMPLE_BUFFER">
      <jSDL:JobDefinition>
        <jSDL:JobDescription>
          <jSDL:Application>
            <jSDL:ApplicationName>SimpleBuffer</jSDL:ApplicationName>
            <jSDL:ApplicationVersion>1.0</jSDL:ApplicationVersion>
            <jSDL:Description>
              A simple command line application that calculates a buffer
              around geometry input data.
            </jSDL:Description>
          </jSDL:Application>
        </jSDL:JobDescription>
      </jSDL:JobDefinition>
    </wsag:ServiceDescriptionTerm>

    <wsag:ServiceDescriptionTerm wsag:Name="RESOURCES"
wsag:ServiceName="SIMPLE_BUFFER">
      <jSDL:JobDefinition>
        <jSDL:JobDescription>
          <jSDL:Resources>
            <jSDL:FileSystem name="HOME">
              <jSDL:FileSystemType>normal</jSDL:FileSystemType>
              <jSDL:Description>Bastian's home </jSDL:Description>
              <jSDL:MountPoint>/home/bastian</jSDL:MountPoint>
              <jSDL:DiskSpace>
                <jSDL:LowerBoundedRange>
                  1073741824.0
                </jSDL:LowerBoundedRange>
              </jSDL:DiskSpace>
            </jSDL:FileSystem>
          </jSDL:Resources>
        </jSDL:JobDescription>
      </jSDL:JobDefinition>
    </wsag:ServiceDescriptionTerm>
  </wsag:All>
</wsag:Terms>

```

```

<jSDL:FileSystem name="HOME">
  <jSDL:FileSystemType>normal</jSDL:FileSystemType>
  <jSDL:Description>Bastian's home </jSDL:Description>
  <jSDL:MountPoint>/home/bastian</jSDL:MountPoint>
  <jSDL:DiskSpace>
    <jSDL:LowerBoundedRange>
      1073741824.0
    </jSDL:LowerBoundedRange>
  </jSDL:DiskSpace>
</jSDL:FileSystem>

<jSDL:FileSystem name="TMP">
  <jSDL:FileSystemType>temporary</jSDL:FileSystemType>
  <jSDL:MountPoint>/tmp</jSDL:MountPoint>
  <jSDL:DiskSpace>
    <jSDL:LowerBoundedRange>
      1073741824.0
    </jSDL:LowerBoundedRange>
  </jSDL:DiskSpace>
</jSDL:FileSystem>

<jSDL:OperatingSystem>
  <jSDL:OperatingSystemType>
    <jSDL:OperatingSystemName>
      LINUX
    </jSDL:OperatingSystemName>
  </jSDL:OperatingSystemType>
  <jSDL:Description>
    A Linux-based operating system with pre-installed
    applications from the geospatial domain (e.g. the
    SimpleBuffer application from this template).
  </jSDL:Description>
</jSDL:OperatingSystem>
<jSDL:IndividualCPUSpeed>
  <jSDL:LowerBoundedRange>1073741824</jSDL:LowerBoundedRange>
</jSDL:IndividualCPUSpeed>
<jSDL:IndividualCPUCount>
  <jSDL:Exact>1.0</jSDL:Exact>
</jSDL:IndividualCPUCount>
</jSDL:Resources>
</jSDL:JobDescription>
</jSDL:JobDefinition>
</wsag:ServiceDescriptionTerm>

<wsag:ServiceDescriptionTerm wsag:Name="ALLOCATION_TIME_CONSTRAINTS"
wsag:ServiceName="SIMPLE_BUFFER">
  <ar:AllocationTimeConstraint>
    <ar:StartTime>2009-11-29T12:00:00.000+02:00</ar:StartTime>
    <ar:EndTime>2009-11-29T18:00:00.000+02:00</ar:EndTime>
  </ar:AllocationTimeConstraint>
</wsag:ServiceDescriptionTerm>

<wsag:ServiceProperties wsag:Name="SIMPLE_BUFFER_PROPERTIES"
wsag:ServiceName="SIMPLE_BUFFER">
  <wsag:VariableSet>
    <wsag:Variable wsag:Name="REQ_CPU_SPEED" wsag:Metric="xs:integer">
      <wsag:Location>
        declare namespace
        jSDL='http://schemas.ggf.org/jSDL/2005/11/jSDL';

```

```

declare namespace
wsag='http://schemas.ggf.org/graap/2007/03/ws-agreement';
$this/wsag:AgreementProperties/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm[@wsag:Name =
RESOURCES']/jsdl:JobDefinition/jsdl:JobDescription/jsdl:Resources/jsdl:IndividualCPUSpeed/jsdl:LowerBoundedRange
</wsag:Location>
</wsag:Variable>
<wsag:Variable wsag:Name="ACT_CPU_SPEED" wsag:Metric="xs:integer">
  <wsag:Location>
    declare namespace
    jsdl='http://schemas.ggf.org/jsdl/2005/11/jsdl';
    declare namespace
    ag='http://schemas.ggf.org/graap/2007/03/ws-agreement';
    $this/wsag:AgreementProperties/wsag:Terms/wsag:All/
    wsag:ServiceDescriptionTerm[@wsag:Name =
    'RESOURCES']/jsdl:JobDefinition/jsdl:JobDescription/
    jsdl:Resources/jsdl:IndividualCPUSpeed/jsdl:Exact
  </wsag:Location>
</wsag:Variable>
<wsag:Variable wsag:Name="REQ_START_TIME" wsag:Metric="xs:integer">
  <wsag:Location>
    declare namespace ar='http://www.ogf.org/graap/AR/2009/04/';
    declare namespace
    wsag='http://schemas.ggf.org/graap/2007/03/ws-agreement';
    $this/wsag:AgreementProperties/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm[@wsag:Name =
    'ALLOCATION_TIME_CONSTRAINTS']/ar:AllocationTimeConstraint/ar:
    StartTime/ \
  </wsag:Location>
</wsag:Variable>
<wsag:Variable wsag:Name="ACT_START_TIME" wsag:Metric="xs:integer">
  <wsag:Location>
    declare namespace ar='http://www.ogf.org/graap/AR/2009/04/';
    declare namespace
    wsag='http://schemas.ggf.org/graap/2007/03/ws-agreement';
    $this/wsag:AgreementProperties/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm[@wsag:Name =
    'ALLOCATION_TIME_CONSTRAINTS']/ar:AllocationTimeConstraint/ar:
    StartTime/ \
  </wsag:Location>
</wsag:Variable>
<wsag:Variable wsag:Name="JOB_EXECUTION_STATE"
wsag:Metric="xs:integer">
  <wsag:Location>
    declare namespace
    wsag='http://schemas.ggf.org/graap/2007/03/ws-agreement';
    $this/wsag:AgreementProperties/wsag:ServiceTermState[@wsag:termName = 'APPLICATION']/wsag:State
  </wsag:Location>
</wsag:Variable>
</wsag:VariableSet>
</wsag:ServiceProperties>

```

```

<wsag:GuaranteeTerm wsag:Obligated="ServiceConsumer"
wsag:Name="ALLOCATION_TIME_CONSTRAINTS">
  <wsag:ServiceScope wsag:ServiceName="SIMPLE_BUFFER"/>
  <wsag:QualifyingCondition>
    (JOB_EXECUTION_STATE eq 'Ready')
    or
    (JOB_EXECUTION_STATE eq 'Complete')
  </wsag:QualifyingCondition>
  <wsag:ServiceLevelObjective>
    <wsag:CustomServiceLevel>
      (REQ_START_TIME le ACT_START_TIME) and
      (empty (ACT_END_TIME) or (ACT_END_TIME le REQ_END_TIME)) and
      (REQ_CPU_SPEED le ACT_CPU_SPEED)
    </wsag:CustomServiceLevel>
  </wsag:ServiceLevelObjective>
  <wsag:BusinessValueList>
    <wsag:Penalty>
      <wsag:AssessmentInterval>
        <wsag:TimeInterval>P30M</wsag:TimeInterval>
      </wsag:AssessmentInterval>
      <wsag:ValueUnit>EUR</wsag:ValueUnit>
      <wsag:ValueExpression>5</wsag:ValueExpression>
    </wsag:Penalty>
  </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
</wsag:All>
</wsag:Terms>

<!-- This element specify valid ranges or distinct values that the terms may
take. -->
<wsag:CreationConstraints>
  <wsag:Item>
    <wsag:Location>
      declare namespace ar='http://www.ogf.org/graap/AR/2009/04/';
      declare namespace wsag='http://schemas.ggf.org/graap/2007/03/ws
      agreement';
      this/wsag:AgreementProperties/wsag:Terms/wsag:All/wsag:ServiceDescri
      ptionTerm[@wsag:Name =
      'ALLOCATION_TIME_CONSTRAINTS']/ar:AllocationTimeConstraint/ar:StartT
      ime/
    </wsag:Location>
    <wsag:ItemConstraint/>
  </wsag:Item>
</wsag:CreationConstraints>

</wsag:Template>

```