Jim Pruyne, Hewlett-Packard Laboratories
Volker Sander, Forschungszentrum Jülich

GGF DOCUMENT SUBMISSION CHECKLIST (include as front page of submission)	
	COMPLETED (X) - Date
1. Author name(s), institution(s), and contact information	X 6/2/2003
2. Date (original and, where applicable, latest revision date)	X 6/2/2003
3. Title, table of contents, clearly numbered sections	X 6/2/2003
4. Security Considerations section	
5. GGF Copyright statement inserted (See below)	X 6/2/2003
6. GGF Intellectual Property statement inserted. (See below) NOTE that authors should read the statement.	X 6/2/2003
7. Document format - The GGF document format to be used for both GWD's and GFD's is available in MSWord, RTF, and PDF formats. (note that font type is not part of the requirement, however authors should avoid font sizes smaller than 10pt).	X 6/2/2003

pruyne@hpl.hp.com, v.sander@fz-juelich.de

Jim Pruyne, Hewlett-Packard Laboratories
Volker Sander, Forschungszentrum Jülich

Grid Resource Allocation Agreement Protocol Operations

Status of This Memo

This memo provides information to the Grid scheduling community. It does not define any standards or technical recommendations. Distribution is unlimited.

Abstract

This document describes a set of usage scenarios of the Grid Resource Allocation Agreement Protocol (GRAAP). The intention is to derive the required capabilities of GRAAP from this document.

Contents

pruyne@hpl.hp.com, v.sander@fz-juelich.de

1. Introduction

The Grid Resource Allocation Agreement Protocol (GRAAP) Working Group addresses the protocol between resource consumers, such as an end-user or a Super-Scheduler (Grid Level Scheduler), and local Schedulers necessary to reserve and allocate resources in the Grid. This document formalizes the related protocol operations.

2. Scope and Background

The coordinated allocation of multiple resources is a challenge. The start-up of the individual service requests somehow must be synchronized without wasting potentially scarce and thus expensive resources by an allocated service request which has to wait for the allocation of related tasks. One potential solution to this is given by the ability to reserve resources in advance. More formally, GRAAP defines the term advance reservation as follows:

> An advance reservation is a possibly limited or restricted delegation of a particular resource capability over a defined time interval, obtained by the requester from the resource owner through a negotiation process.

An appropriate mechanism to model advance reservation is given by the use of Service Level Agreements (SLAs). An SLA allows clients to understand what to expect from resources without requiring detailed knowledge of competing workloads or resource owners' policies. This concept holds whether the managed resources are physical equipment, data, or logical services. Given that each of the resources in question may be owned and operated by a different provider, establishing a single SLA across all of the desired resources is not possible. The decomposition of management functions into different types of SLAs that can be composed incrementally is an elegant solution to this problem. As a consequence, we can model the Grid Resource Allocation Agreement Protocol around the concept of SLAs. The operations of GRAAP relate to the lifetime and state of an SLA. The reservation of a particular resource capability is different than the allocation of a given activity. It is therefore important to state, that though SLAs are in the focus of GRAAP, there have to be different types of SLAs. Following the formal definition of advance reservation, three types of SLAs are proposed for this:

- Task service level agreements (TSLAs) in which one negotiates for the performance of an activity or task. A TSLA is, for example, created by submitting a job description to a queuing system. The TSLA characterizes a task in terms of its service steps and resource requirements.
- Resource service level agreements (RSLAs) in which one negotiates for the right to consume a resource. An RSLA can be negotiated without specifying for what activity the resource will be used. For example, an advance reservation takes the form of an RSLA. The RSLA characterizes a resource in terms of its abstract service capabilities.
- Binding service level agreements (BSLAs) in which one negotiates for the application of a resource to a task. For example, an RSLA promising network bandwidth might be applied to a particular TCP socket, or an RSLA promising parallel computer nodes might be applied to a particular job task. The BSLA associates a task, defined either by its TSLA or some other unique identifier, with the RSLA and the resource capabilities that should be met by exploiting the RSLA.

These SLAs can be used to separate the complex service acquisition process into multiple phases, each negotiating the appropriate level of agreement. This modelling facilitates a

pruyne@hpl.hp.com, v.sander@fz-juelich.de

distributed coordination of resources without any resource-resource trust relationship. It is the scope of this document to clarify the operations needed to maintain an SLA during its lifetime.

Fortunately, GRAAP can rely on existing work. Within the Global Grid Forum, the Advance Reservation API document (GFD-E.5) addresses the issue of interfacing with advance reservation capabilities. Recently, the Service Negotiation and Acquisition Protocol (SNAP) extended this work and introduced the above listed concept of SLAs. This document is intended to propose the operational semantics of the Grid Resource Allocation Agreement Protocol based on the work of SNAP. To clarify the proposal, it also maps the operations of the Advance Reservation API to the GRAAP operations. This document does not present a final binding of the protocol operations to any particular representation such as WSDL, IDL or other precise, machine-readable formats. It is assumed that this will follow as a refinement of the general presentation given here, and we anticipate this taking a form which is compatible with the OGSI specification for Grid services.

3. Basic Elements of the Grid Resource Allocation Agreement Protocol

3.1 Operations

The Service Negotiation and Acquisition Protocol (SNAP) defines a method for reliable creation and management of remote SLAs. GRAAP follows these concepts. Protocol operations are therefore modelled as unidirectional messages sent from client to service or service to client. All of these operations follow a client-server remote procedure-call (RPC) pattern, so it assumes the underlying transport will provide correlation of the initiating and responding messages. One way of interpreting the following descriptions is that the client to service message corresponds to the RPC, and the return messages represent the possible result values of the call. Using an RPC-style model nicely fits to the specification of related WSDL documents and allows for an easy understanding of a protocol state diagram.

Following the SNAP approach, GRAAP proposes for its specification an abstract agreement meta-language for maintaining a set of manager-side SLAs using client-initiated messages. The fundamental idea of this is to abstract the operational description from the selection of an actual resource description language. Instead, the meta-language allows for a formalization of SLA negotiation by claiming a few common attributes: SLA identification, client with whom the SLA is made, and an expiration time, and a generic syntax for the creation of resource metrics, composites, alternatives, and configurations.

Based on this common SLA modelling, GRAAP operates with only three basic operations:

1. Allocate Identifier Operation: Before any SLA negotiation can start, a naming agreement must be established which binds a unique identifier to a particular negotiation. This light-weight operation represents the first state of a multi-state negotiation process. By associating a limited lifetime of an identifier, it is analogous to opening a timed transaction in a database system. Note that the lifetime itself might be negotiated between the partners as well.
2. Agreement Operation: SLAs can be established by performing agreement operations for valid SLA naming identifiers. The service requester specifies requested SLA attributes to initiate the negotiation process. Note that the specification is formulated in a Resource Description Language (RDL) which allows clients to request resources by property, e.g. by capability, quality, or configuration. This includes complex resource metrics, topologies and configuration directives. While the service requester specifies the desired SLA constraints, the service provider either declines the service, or responds with SLA

pruyne@hpl.hp.com, v.sander@fz-juelich.de

attributes that fulfil the demanded service attributes, but which might add additional constraints. This allows for a negotiation process based on specialization. Further agreement operations on this SLA identifier will perform updates on this agreement. Note that agreement operations are idempotent, i.e. they follow an at-most-once semantics.

3. Set Termination Operation: Each SLA has a termination time, after which a well-defined reclamation effect occurs. Clients can explicitly set the lifetime of a valid SLA identifier. This operation allows for a soft-state model in which SLAs (which might list different time constraints) have to be permanently refreshed.

These three basic operations are the building blocks for GRAAP. It is important to note that they apply for different types of SLAs. While GRAAP gives a simple framework for the establishment of SLAs, the ontology of SLAs is the key factor for understanding the actual progress in negotiating SLAs. Without specifying this ontology, GRAAP helps to model the agreement negotiation as a discovery process by which the client determines the willingness of the manager to serve the client according to the requested level of service. By formulating future agreements with weak commitment and changing them to stronger agreements, a client is able to perform a multi-phase commit process to discover more information in an unstructured environment. Resource virtualization is introduced by the composition of SLAs. A service provider is therefore able to incorporate a particular set of resources (or services) to a virtual organization and thus helps in terms of service discovery by aggregating policy knowledge into a private discovery service. This relates the presented concept to the construction of higher-level services such as a super-scheduler.

4. Operational Semantics

We now describe the operational semantics of GRAAP. We therefore start on describing the basic SLA operations and how they are mapped to the elementary protocol operations. This mapping is summarized by a state transition diagram of GRAAP.

4.1 A Function Oriented View: Service Functions

Service functions are used to abstract complex SLA operations. These functions are characterized by their intended manipulation on SLAs, but are not exactly defined, since they are meant to isolate the formal model from implementation details. This fuzziness allows for a specification without defining the underlying ontology.

4.1.1 Negotiating an RSLA: Function Reserve()

Advance reservations are modeled by RSLA. From an operational perspective, the process of establishing an RSLA can be handled by a reserve() function. It maps a particular capability request to an SLA that can be used to claim the related service under the constraints listed in the SLA. The negotiation of an RSLA requires the agreement on a valid SLA identifier, i.e. the exchange of a getident() request and a useident() response message, and the required agreement operations, i.e. potentially multiple request(RSLA) – agree(RSLA) messages.

4.1.2 Negotiating an TSLA: Function Schedule()

An TSLA is negotiated when a service requester delegates an activity to a service provider. A schedule() function is introduced to establish an TSLA. It maps a particular activity to a virtualized resource. Again, it relies on a naming agreement established by an exchange of getident() and useident() messages. To model the negotiation process, multiple agreements operations might follow. Each of it might negotiate an offer which has to be committed, once the entities accept the result of the negotiation process.

pruyne@hpl.hp.com, v.sander@fz-juelich.de

4.1.3 Negotiating a BSLA: Function Bind()

An SLA of particular interest is the Bind SLA. The particular interest is caused by its multifarious use. We associate a bind() function for the operations related to establish a BSLA. It links a particular activity associated with a TSLA to the resource capabilities offered by an RSLA. Bind() SLAs allow for a provisioning of reservation relevant attributes which were only existent when the activity was already started. Binds() could be used to commit an existing TSLA. However, we describe further commit options below. Binds() could also be used in the context of an aggregated reservation. Here, multiple binds() refer to an existing RSLA.

4.1.4 Polymorphism of SLAs

An important aspect of the operational description of GRAAP is caused by the polymorphism of SLA, i.e. SLAs which were related to more then one of the described types. These polymorph SLAs were used in implicit operations. So far, service activities did require the existence of on a TSLA, i.e. an agreement on an SLA naming and a successful negotiation of the TSLA attributes. On the other hand, SLAs were introduced to abstract the negotiation process by modeling the protocol with only a few types of messages. As a logical consequence, a single agreement negotiation can create an SLA which is both: a TSLA and a RSLA. We call this implicit SLA. In GRAAP, a TSLA might automatically create an RSLA and vice versa. GRAAP also allows for an implicit creation of an RSLA when a bind() operation is performed.

4.1.5 Renegotiating SLAs

SLAs might vary during their lifetime. Renegotiation SLAs is an important feature and is thus addressed within GRAAP. From an operational perspective, three functions can be introduced to perform the related message exchange: Rereserve(), Reschedule(), and Rebind(). The associated messages are always agreement operations.

4.2 Transition Rules

So far, we described the basic service functions. However, all of them were related to only two types of messages. The missing third type of message is used to model the transition rules of SLAs.

4.2.1 Lifetime Management

A fundamental task of managing SLAs is to manage their lifetime. A limited lifetime allows for a soft-state model in which robustness is implemented by the expiration SLAs in case of sever failure conditions. As described above, an SLA relies on a naming agreement which is of limited lifetime. The related getident() and useident() messages associate an initial lifetime of the naming agreement.

4.2.2 Timeout Changes

Each partner is capable of changing the lifetime this agreement by issuing the setdeath() and willdie() operations. Note that the willdie() response might not accept the updated termination time because of policy reasons. Also note that these operations are related to the lifetime of the naming agreement. If the related SLA makes promises about times in the future beyond its current lifetime, those promises expire with the lifetime of the naming agreement. Thus, it is a client's responsibility to extend or renew an SLA for the full duration required. Further on, agreements can be abandoned with a simple request of setting the lifetime to zero which forces expiration of the agreement.

pruyne@hpl.hp.com, v.sander@fz-juelich.de

4.3 State Diagram

We now describe the protocol operation in presenting a state diagram for service providers. It is based on the following states:

No State: The service provider is waiting for new service requests.

SLA Reference: There is a valid SLA naming agreement.

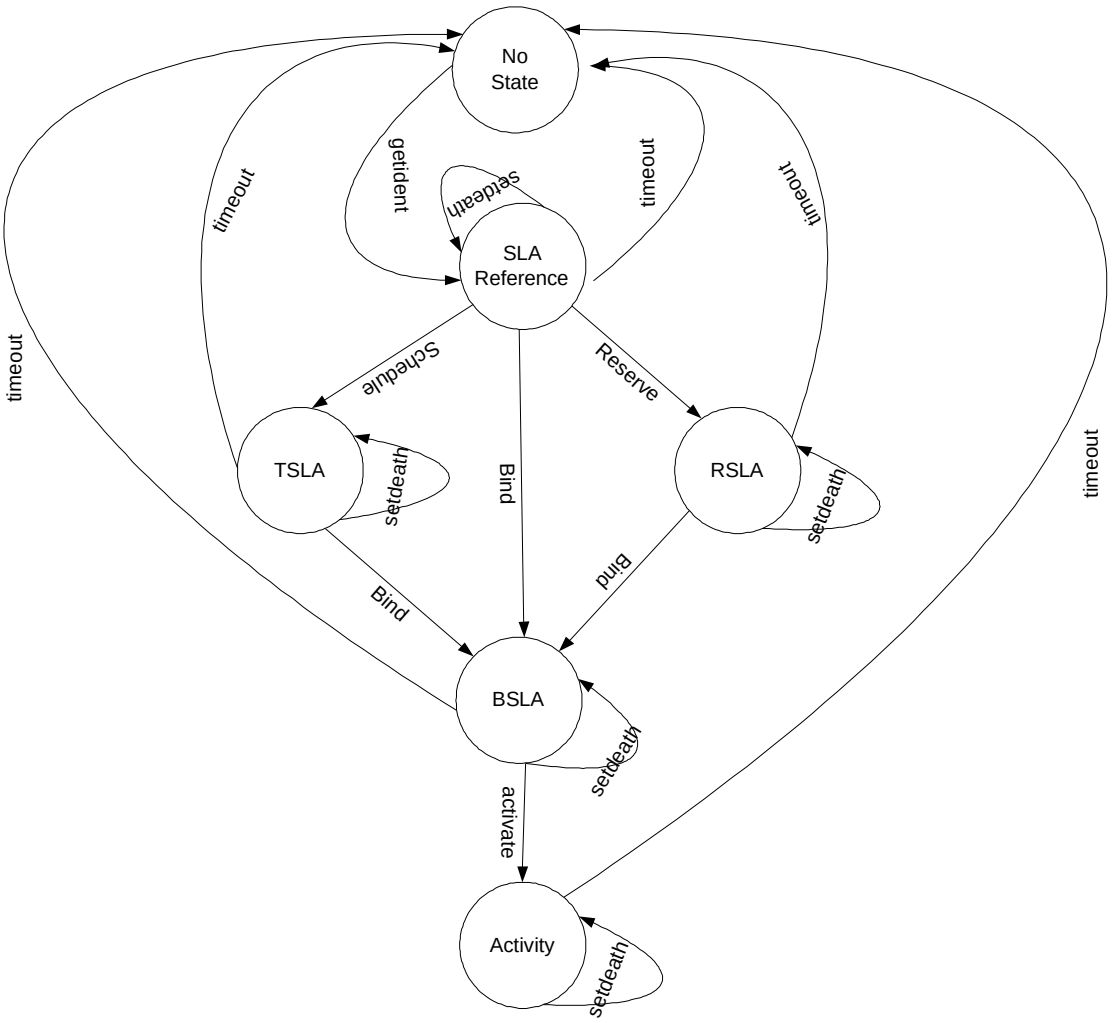
TSLA: An SLA was accepted to perform an activity, but the activity is not yet performing.

RSLA: An SLA was accepted in which capacity is delegated, but not yet fully claimed.

BSLA: A capacity delegation was mapped to activity, but the time for execution has not yet arrived, or is about to be be activated.

Activity: Under activity is performed under the constraints of a BSLA.

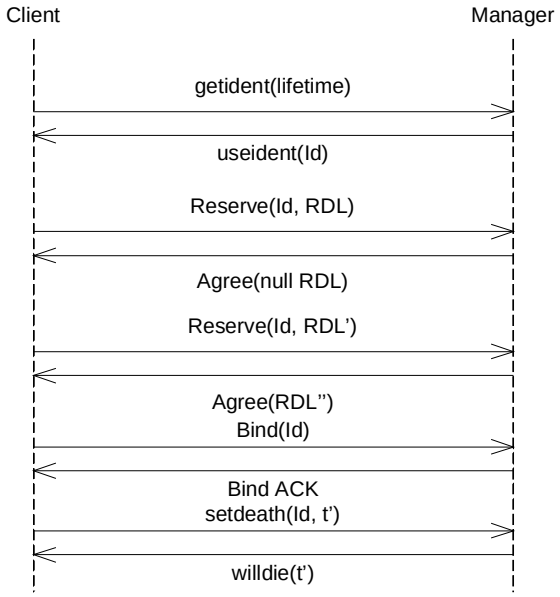
pruyne@hpl.hp.com, v.sander@fz-juelich.de



4.4 An SLA Negotiation Scenario

The following figure provides an example of messages that would flow between a client and the manager in creating an RSLA, and to bring it into effect. The exchange shows one path through the state machine above until the activity state is reached. We start with the creation of the RSLA via a getident message. As specified, this message contains a desired lifetime for the RSLA. The manager responds by creating the SLA state locally, and responds with the SLA's identifier. This is followed by a reserve message that contains the id of the SLA as returned previously, and the description of the desired resources. We assume the existence of an RDL for this purpose, though the contents of the RDL are outside the scope of GRAAP. In the initial exchange, we assume that the manager cannot provide any sort of agreement that satisfies the RDL created by the client. It responds with agree message containing a null RDL. That is, no set of resources can be provided that satisfies this RDL or any refinement of this RDL. We can consider this a form of "disagreement" message. The client is free to try again with another reserve message, this time containing an updated request, RDL'. In this case, we assume that the manager is able to provide an SLA based on this description. It does, however, refine the description to RDL''. At this point, the client accepts the provided SLA, and brings it into effect via a Bind call. The manager acknowledges this, and the SLA is now in effect until the time specified in the initial

getident message is reached. The client can then attempt to change the lifetime of the SLA, either shorter or longer, by sending a setdeath message. In this flow, we assume that the manager is willing to alter the lifetime of the SLA, and responds with a willdie message containing the same lifetime, t', as proposed by the client.



5. Modeling the Operations of GFD-E.5

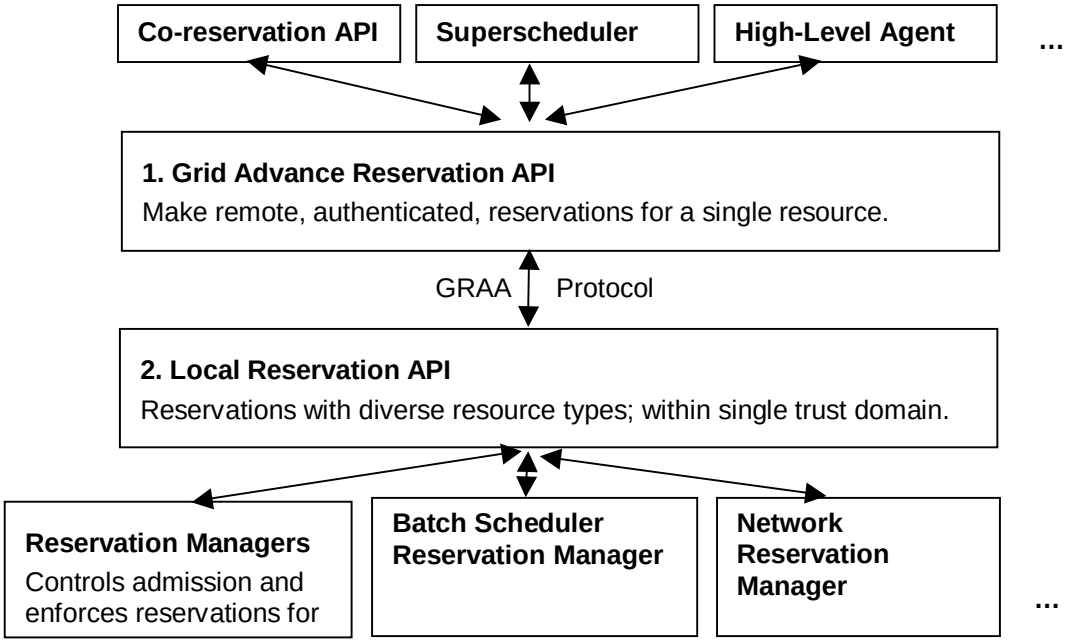
This section relates the GRAAP specification to the experimental Interface described in GFD-E.5

5.1 GFD-E.5: Advance Reservation API

The GFD-E.5 Advance Reservation API document describes an experimental interface to advance reservation capabilities. The proposed Grid Advance Reservation API can be considered a remote procedure call mechanism to communication with a reservation manager. A reservation manager controls reservations for a resource: it performs admission control and controls the resource to enforce the reservations. Some resources already can work with advance reservations, so the reservation manager is a simple program. Most resources cannot deal with advance reservations, however, so the reservation manager tracks the reservations and does admission control for new reservation requests.

To create a flexible architecture that supports co-allocation, resource location, and resource acquisition steps, it proposes a layered architecture with three levels of APIs and one level of low-level mechanisms.

pruyne@hpl.hp.com, v.sander@fz-juelich.de



Layered advance reservation architecture

The document uses the term *Reservation* as a promise from the system that an application will receive a certain level of service from a resource. In the context of GRAAP, a reservation corresponds to a valid RSLA.

Based on this, it defines the following operations:

§ *Create Reservation:* This function interfaces to the negotiation of an RSLA. It identifies five important attributes associated with the operation:

Start Time: The earliest time that the reservation may begin. A reservation always has a start time, even if it is an immediate reservation, which begins as soon as the reservation is made. The start time is in seconds from 00:00:00 UTC, January 1, 1970. For example, to make an immediate reservation, the user can call the Unix time() function to discover the current time and use that as the start time.

Duration: The amount of time the reservation lasts, in seconds. All reservations must specify their duration, so that the underlying reservation managers can do appropriate admission control for reservations granted in advance.

Resource Type: The type of underlying resource, such as a network, a computer, or a disk.

Reservation Type: A particular kind of reservation.

Resource-Specific Parameters: Parameters that are unique to each type of resource, such as bandwidth for a network reservation and number of nodes for a computation reservation.

Optionally, a reservation may specify the attribute *End Time* in the same way that *Start Time* is specified. If the difference between *End Time* and *Start Time* exceeds the value of *Duration*, any given time interval of the correct duration starting at or after *Start Time* and not ending past *End Time* can be used for the RSLA.

The mapping of this operation is quite simple as it corresponds to the above described function reserve().

- *Modify Reservation:* This operation allows for a modification of an existing reservation. It is thus used to renegotiate an existing RSLA. We referred to this as rereserve().

- *Cancel Reservation:* This operation terminates an existing agreement. It corresponds to a setdeath()-willdie() message exchange.

- *Bind Reservation:* When an application is ready to use a reservation, it may need to provide run-time information that was not available at the time the reservation was made. This is known as *binding* a reservation. For example, network reservations require port numbers to be specified, but they are not usually known at reservation time. Not all reservations require such run-time parameters. Binding a reservation is modeled as a separate negotiation process. As it links a particular activity associated with a TSLA to the resource capabilities offered by an RSLA, a bind operation operates on an existing SLA naming agreement. Hence, it does not involve an exchange getident()-useident() messages. Instead, it converts an RSLA to BSLA. Here, the resulting BSLA has more constraints. Note that the RSLA remains in principle valid, i.e. it can be used by additional requesters.

- *Unbind Reservation:* A reservation can be unbound. It then will no longer be usable by the person using the reservation. It can be rebound, however, with new run-time parameters. Note that unbinding eliminates the BSLA, it therefore corresponds to a setdeath()-willdie() message exchange.

- *Commit Reservation:* When a reservation is created, it can be specified as a two-phase commit reservation. Such reservations time-out after a specified time-period, unless the reservation is committed. GRAAP does not explicitly model a two-phase commit. Instead, it allows for lifetime management in general. It is therefore straight forward to implement a two-phase commit by limiting the lifetime of a naming agreement and extending it, once the commit has been decided.

- *Query Reservation Status:* One can discover the status of a reservation by polling it. The status includes whether the start of the reservation has begun and whether the reservation has been committed. GRAAP does not explicitly cover this operation. However, as the protocol specification will follow the Open Grid Services Architecture (OGSA), it can be implemented using the notification port-types of OGSA.

- *Query Reservation Attributes:* One can discover attributes associated with an existing reservation. These include begin and end time of the given reservation and whether it is a two-phase commit reservation. The attributes also include specific information required to actually use a reservation. Example attributes are a directory name where data was staged

on, or a queue name which has to be used for submitting a job. It is expected that this information is encoded within the SLA.

- *Register Callback:* One can provide a function that will be called when the status of a reservation changes or when the reservation manager wishes to provide extra information to the application. This information may include notification that the related reservation appears to be too small. Again, GRAAP assumes that this will be implemented by using the notification port-types of OGSA.

5.2 Task Service Level Agreements

While the described mapping of the Advance Reservation API helps to explain the principle operation of GRAAP and the relation between RSLA and BSLA, it misses the most important SLA: The Task Service Level Agreements. Neither an RSLA nor a BSLA did directly result in an activity. The interface for performing an activity is modeled by a TSLA. There are two fundamental ways of negotiating a TSLA: explicit and implicit. We now refine the description of the relation between Advance Reservation API and GRAAP.

In GRAAP, SLAs are polymorph, i.e. a single SLA negotiation process causes the establishment of two types of SLAs: TSLA and RSLA. Hence, a reservation create() operation might automatically create an agreement on an activity. As even a bind() may be omitted in some cases, i.e. implicitly created BSLA, a single exchange of agreement messages can be convenient for submitting an activity with a reservation. When the time constraint of the SLAs arrive, and the naming agreement is still valid, the activity will be automatically performed by the service provider according to the capacity delegation attributes.

6. Security Considerations

Security considerations are presently not included.

Author Information

Jim Pruyne
Hewlett-Packard Laboratories
1501 Page Mill Rd.
Palo Alto, CA
650-857-8310
pruyne@hpl.hp.com

Volker Sander
Zentralinstitut für Angewandte Mathematik,
Forschungszentrum Jülich GmbH,
52428 Jülich
Germany
+49 (0) 2461 61-6586
v.sander@fz-juelich.de

Intellectual Property Statement

Full Copyright Notice

References

[GFD-E.5] Advanced Reservation API, V. Sander and A. Roy, Grid Forum Document (GFD), Experimental 5 (E-5)

[SNAP] SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems.
K. Czajkowski, I. Foster, C. Kesselman., V. Sander, and S. Tuecke.
Lecture Notes in Computer Science, 2537:153-183, 2002.

pruyne@hpl.hp.com, v.sander@fz-juelich.de