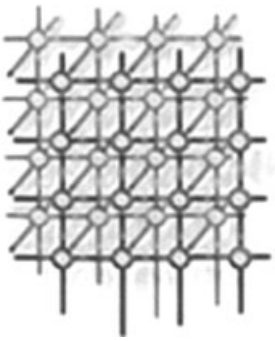
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2009)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1408



The HPC basic profile and SAGA: standardizing compute grid access in the open grid forum

Chris Smith[1,*,†], Thilo Kielmann[2], Steven Newhouse[3] and Marty Humphrey[4]

[1]*Platform Computing Inc, 101 Metro Dr, Suite 540, San Jose, CA, U.S.A.*
[2]*Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands*
[3]*Microsoft Corporation, 1 Microsoft Way, Redmond, WA, U.S.A.*
[4]*Department of Computer Science, University of Virginia, Charlottesville, VA, U.S.A.*

SUMMARY

After seven years of life the Open Grid Forum (OGF), previously the Global Grid Forum, is beginning to produce standards that meet the needs of the community and that are being adopted by commercial and open source software providers. Accessing computational resources, specifically high performance computing (HPC) resources, is a usage scenario that predates the emergence of the Grid, is well understood within the community, and is represented in numerous gathered use cases in different areas of Grid computing. Building on this consensus the HPC Profile Working Group was established within the OGF to standardize access to HPC resources. Its first specification, the HPC Basic Profile 1.0 has been developed and has established interoperability within the community. Alongside the development of this protocol, another group within the OGF, the Simple API for Grid Applications (SAGA) working group has been defining a programmatic interface relevant to application developers that encompasses access to compute resources. This paper examines the relationship between the 'standard' protocol of the HPC Basic Profile and the programmatic interface of SAGA to access compute resources and assesses how well these address the problems faced by the community of users and application developers. Copyright © 2009 John Wiley & Sons, Ltd.

Received 15 March 2008; Revised 27 August 2008; Accepted 1 November 2008

KEY WORDS: high performance computing; standards; web services; grid; APIs

*Correspondence to: Chris Smith, Platform Computing Inc, 101 Metro Dr, Suite 540, San Jose, CA, U.S.A.
†E-mail: csmith@platform.com

Contract/grant sponsor: National Science Foundation; contract/grant number: ANI-0222571
Contract/grant sponsor: Microsoft Research
Contract/grant sponsor: OMII-UK and the European Commission

1. INTRODUCTION

From the early days of the Open Grid Forum (OGF) [1], there has been particular focus on standardizing access to computing resources, whether by an end user submitting work directly to a scheduler or meta-scheduler, or by a meta-scheduler communicating with other peer or lower-level schedulers. There are many examples of this Grid usage pattern, from research-oriented 'eScience' Grids such as TeraGrid, DEISA, EGEE, and NAREGI, to the Enterprise Grids that are deployed within a single organization using software from commercial vendors such as Platform Computing and Microsoft Corporation. It is telling that the first three documents in the OGF document series (after the three community practice documents that set out the processes for the document process itself) are 'Ten Actions When SuperScheduling' (GFD-I.4) [2], 'Advance Reservation Appplication Programming Interface (API)' (GFD-I.5) [3], and 'Attributes for Communication between Scheduling Instances'(GFD-I.6) [4]. So how much progress has been made since July 2001 when GFD-I.4 was published? Does there exist sufficient specifications in the area of compute Grid access that would allow end users and peer scheduling services to communicate in an interoperable way? Does there exist a standardized API that facilitates the creation of portable applications that execute in the context of a compute Grid?

In this paper, we describe two recent activities in the OGF that form the basis of standardized access in compute Grids. First, we describe the High Performance Computing (HPC) Basic Profile [5], which is a Web services profile that realizes the vertical use case of batch job scheduling of scientific or technical applications. Then, we describe SAGA [6,7], the 'Simple API for Grid Applications', that includes support for job submission and management. We discuss and evaluate the HPC Basic Profile and SAGA with regard to the 'user' to whom the standard applies (whether a real person or another software system), the 'use cases' that the standard was designed to address, and the particular maturity level of the given specification. While the HPC Basic Profile and SAGA have been developed independently (and thus it is inappropriate to believe that one is necessary for the other or one is even an endorsement of the other), together they provide complementary capabilities for standardized access to compute Grids.

The remainder of this paper is as follows. Section 2 provides the basis upon which we evaluate the HPC Basic Profile and SAGA. Section 3 describes and evaluates the HPC Basic Profile, including the Job Submission Description Language (JSDL) [8] and the OGSA Basic Execution Service (BES) [9]. Section 4 describes and evaluates SAGA. Section 5 concludes and gives a brief overview of the subsequent steps for the HPC Basic Profile and SAGA specifications.

2. EVALUATING THE PROGRESS OF STANDARDS FOR COMPUTE GRID ACCESS

Standards can be challenging to evaluate. In this section, we describe in detail the basis for evaluating the HPC Basic Profile and SAGA: the user focus (Section 2.1), the use cases that the standard is targeting (Section 2.2), and the maturity of the specification (Section 2.3).

2.1. User focus

Standards specifications need to have a clear idea of who the consumer of a particular specification will be. It is obvious that the main consumers of a specification will be the software developers who

DOI: 10.1002/cpe

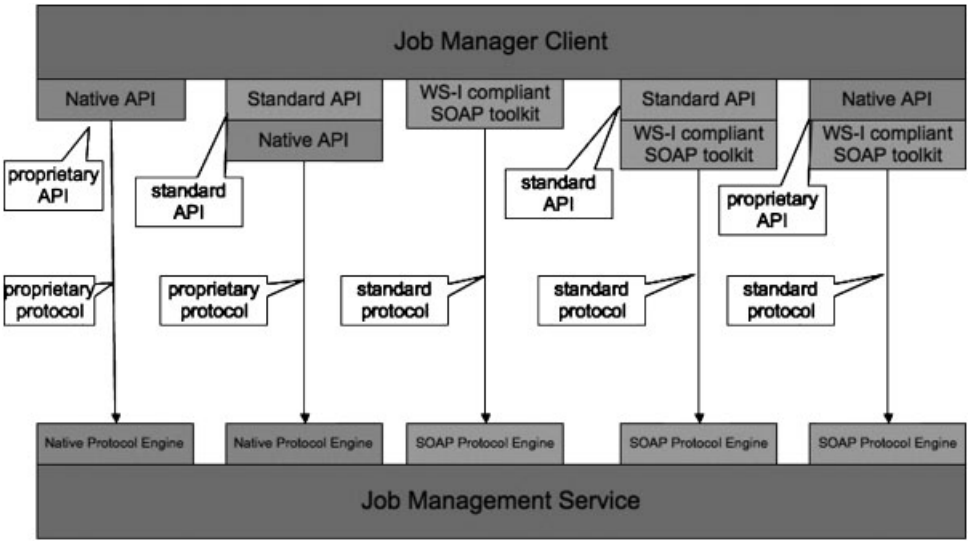


Figure 1. API versus protocol standards.

will be implementing the specification, but what kind of a developer is this, and who is the consumer of their work? It is useful to ask this question, because the type of software that a particular user is implementing, whether Grid services or Grid clients, will determine the type of specification that will be developed. In the OGF area of focus, where Grids are characterized by distributed services accessed over the network, there are three main types of specifications:

- Schemas
- Protocol specifications
- API specifications

The relationship between protocol and API specifications for a typical compute Grid scenario involving a job management client and service is shown in Figure 1. One can see that there are multiple combinations of API and protocol, standard and proprietary, which provide various levels of ease of use to client and service implementers. For instance, it is common to provide implementations of standard APIs on top of proprietary APIs (and thus protocols) if a particular provider of a service is not willing to implement a standard API or protocol themselves. It is also common to layer a native API over a standardized protocol in order to support legacy applications that cannot be re-coded to take advantage of standardized APIs, yet there is a desire to port these applications to make use of new services. Although not shown here, Schemas are vital in describing the capabilities of the Job Management Service to the client in order to derive the selection of the most appropriate resource for their needs.

2.1.1. Schemas

The first type of specification is a *Schema*, which defines the information being passed between clients and services, and between peer services. If these are defined using XML, then these

DOI: 10.1002/cpe

specifications represent the document formats that are used to structure the information flowing between nodes in the Grid. These schemas enable both clients and services to 'speak the same language' about Grid resources and activities.

Within the OGF there are two complementary activities relating to Information Schemas (or Models): use of the Common Information Model (CIM) schema [10] from the Distributed Management Task Force (DMTF) and the Grid Laboratory Uniform Environment (GLUE) Working Group [11]. These are by no means the only schema activities occurring within the OGF. For example, there is an OGF proposed recommendation for describing the resource consumption of a computing activity (Usage Record—GFD.98 [12]). Information schemas are most germane to the discussion of access to computing resources, though, as they provide the linkage between providing a description of available resources from the supplier side, and the necessary step of providing a specification for a desired allocation of resources within a JSDL document. The collaboration with the DMTF has led to the definition of an information model around the BES to describe the computation resources accessible through the service. Elements of this information model have been introduced as an experimental option in the latest version of CIM [10]. The GLUE-WG is building on the operational experience gained over many years in large research Grids. The information model published by the sites within a Grid enables both the monitoring and selection of resources by higher-level services by describing a site's computing resources, hosted applications, and its access policies. The group is defining how this model can be rendered within CIM, as an XML schema or within other structured data models.

2.1.2. Protocol specifications

The second type of specification is a *Protocol Specification*, which defines the messages that pass between Grid clients and services, and the ordering of these messages such that clients may request services to perform an action on their behalf.

While the format 'on the wire' (i.e. the message) is of primary concern, those alone only tell half the story. The order in which messages are exchanged, and the 'state' of the interactions is equally important when defining a protocol. In the case of many of the OGF protocol standards currently being defined, there is a move to use SOAP-based Web services [13] as a framework for defining these protocols. The use of SOAP allows the writer of a specification to adopt an existing 'pattern' for message exchange, such as request/response, and the use of the Web Service Description Language (WSDL) [14] allows for message definition using a standardized interface definition language. This reduces the burden on the specification writer since they can just refer to an existing messaging pattern and make use of an existing framework for defining the protocol operations, and it reduces the burden on the protocol implementer as they can make use of an existing SOAP protocol library to implement their client or service by automatically generating protocol stubs using code generators that parse the WSDL interface definitions. SOAP-based Web services also make use of XML Schema, so that any information schemas that can be rendered using XML Schema are immediately available to plug into the SOAP framework, thus providing some standardized way to encode protocol messages on the wire.

Protocol specifications are generally preferred by the implementers of Grid services. Grid services need to be exposed on the network in order to be useful, so that the burden of implementing a network endpoint already exists for the service implementer. Providing an endpoint that accepts standardized

DOI: 10.1002/cpe

messages is thus a straightforward implementation path, and can even be implemented in parallel with proprietary protocols. Contrast this with an API specification where multiple language bindings may need to be implemented and supported for a single specification.

2.1.3. API specifications

The third type of specification is an *API*. These types of standards define programming language interfaces that expose the desired Grid service functionality within a program running on the client's node in the Grid. In Grid computing environments, libraries exposing a particular API usually end up implementing a protocol in the library (whether it is a standard protocol, or a proprietary one), but the details of the protocol implementation, including message formats and the state of message exchange, are hidden from the API user.

API specifications are generally focused on the implementer of Grid clients. APIs provide access to Grid services using a programming model that is more natural to the client implementer. For example, an API specification might provide an object-oriented view of the Grid service, which would not naturally emerge from the underlying protocol messages. This is the primary reason to support both API and protocol specifications. The service implementer provides access via a protocol that might not be easy to use for the implementer of the client. By combining a standard protocol with a standardized API, the burden on both service and client implementers can be reduced.

2.2. Use cases

When defining specifications it is important to identify the problem being addressed in order to be able to measure progress and to identify when the specification is done. While this sounds obvious, in the OGF it is generally left to individual working groups to set these criteria, and in some cases in the past, this has not been done, leading to standardization efforts that seemingly never end. A typical way of defining the criteria for completion is through use cases. While the way of documenting use cases might be different from working group to working group, the idea is to 'tell a story' about the problem that is to be solved by a particular specification in enough detail to be able to analyze whether the elements of the specification will adequately solve the problem. For instance, if a use case states that a client must be able to ask a Grid service its name, and the specification does not provide an operation called 'get name', then the use case has not been solved by the specification.

For accessing compute Grids, it is worth identifying two relevant use case documents: 'HPC Job Scheduling: Base Case and Common Cases' (GFD-I.100) [15] and 'A Collection of Use Cases for a SAGA' (GFD-I.70) [16].

GFD-I.100 describes the scheduling of scientific applications as batch jobs within an organizational computing cluster, which is referred to as the 'typical' HPC use case. So what does this have to do with Grid computing? While described as 'HPC', this scheduling of scientific applications onto computing resources is equally applicable in Computing Grid environments where jobs flow from one organization to another, although some details, such as the authentication and authorization models, might differ. For this reason, it is worth categorizing much of the use of Compute Grids using the moniker 'HPC'.

DOI: 10.1002/cpe

GFD-I.100 defines the 'Base Case' of the HPC use case as having the following characteristics:

- User Interface: Users can make requests to a scheduler to submit a job, query a specific job, cancel a specific job, and to list the jobs they have submitted.
- State Diagram: Jobs submitted to a scheduler will be in one of three states: *queued* means it is waiting to run, *running* means the job is occupying resources allocated by the scheduler and *finished* when the job terminates (for whatever reason).
- Resource Descriptions: Users may provide resource requests using a small set of 'standardized' attributes, such as number of CPUs/nodes needed, operating system type required, etc.
- Fault Tolerance: If a job fails, it is up to the client to resubmit the job as a new job if it would like to attempt to rerun the job.
- Out-of-band Aspects: Data that are to be processed by the job is not provided by the job submission request. Data access is considered out-of-band to the job management requests and operations.
- Out-of-scope Considerations: Scheduling policies are considered out of scope, including notions of Service Level Agreements. Jobs are independent of each other, and running parallel jobs is not assumed for the base HPC use case. Resource reservation and interactively running jobs are also considered out-of-scope.

The set of characteristics that make up the base use cases represents an intersection of characteristics of the various use cases that were examined by the HPC Profile working group. What makes a characteristic 'basic' is the fact that all use cases needed that functionality. For example, all use cases express the need to be able to terminate a unit of work that had been submitted to a job management system, but since some job management systems do not support the capability to suspend the execution of a job, the need to suspend a given job would not be considered a basic characteristic of the base use case. Similar reasoning can be applied to elements of the basic use case such as the simplified state model, the data access mechanisms, etc.

While [15] goes on to describe some common use cases that increase the scope beyond the base case, the characteristics of the base use case provide the basic criteria for evaluating the state of job management specifications that will be discussed later in this paper. It is worth highlighting one of the common use cases described in [15] that together with the base use case could be considered the 'base' use case for *Grid* HPC jobs; that is, the multiple organization use case. In this use case, HPC jobs are submitted to a computing resource that resides within a different organization from the submitting user.

There are two elements of this use case that are worth mentioning. First, submission will require some form of additional security support to either federate identity systems between the organizations or more simply to provide a technique for 'outside users' to be mapped to some appropriate principal inside the organization hosting the job. Second, the data that the client will process might reside at the client's facility, and thus must be transferred from the client's site to the computing site, and results must be transferred back if desired. These two additional characteristics, along with the characteristics of the base HPC use case, define the base Grid HPC use case. It will be worth evaluating current specifications in light of these characteristics as well as the base HPC use case characteristics.

In GFD.I-70 a set of use cases are described that are used to inform the specification of SAGA. The scope of SAGA is much wider than just the use of compute Grids, since it includes use cases

DOI: 10.1002/cpe

for data management, visualization, and remote steering, but one of the focus areas of the API is on the submission and management of jobs. In GFD-I.70, use cases are categorized using a number of 'phrases', including 'high throughput computing' and 'HPC', which are used to indicate job management functions. Since these categories of use case are referring to the submission and management of compute jobs, typically run in batches, these phrases can be interpreted to be very similar to the HPC base use case described in GFD-I.100. Of the 23 use cases listed in GFD-I.70, 16 identified this type of job management function as very important to their use case, and a further four identified job management as of medium importance.

2.3. Specification maturity

The final factor to take into consideration when evaluating the progress of specifications is the maturity level of the specification. In the OGF, specifications go through various phases during their life cycle [17]. These are not 'official' document states, but are useful for helping to classify the maturity of a specification.

- Draft phase: This is the phase where the specification is being developed and written. At this stage, there might or might not exist implementations of the specification, but the specification is subject to change at this time.
- Interoperability phase: At this point, the draft is complete and could even be as far ahead as to be a 'proposed recommendation' in OGF nomenclature. For protocols, at this point there are two or more implementations of the specification, and the implementers are demonstrating that their implementations can communicate successfully. For APIs, either there is a compliance test suite for verifying that an implementation complies with the standard, or there exists an official reference implementation to compare with other implementations.
- Adoption phase: This is the phase where the implementers of Grid services in the specifications problem area have implemented the specification in their products and software stacks. Implementations at this phase are often going from 'prototypes' in the interoperability phase, to full product in the adoption phase.
- Deployment phase: This is the point in time when Grid clients have started to adopt the specification, and the implementations of the specification are going into use by those who deploy and use Grid middleware. Specifications at this phase could be considered 'very successful'.

Note that these categories are somewhat fluid depending on the state of the specification. For instance, after an interoperability phase, enough issues might have been found in the specification to require a new draft phase. It is also not guaranteed that a specification will go beyond any one phase.

3. HPC BASIC PROFILE

The HPC Profile Working Group (HPCP WG) [5] was formed in May 2006 within the OGF (GGF at the time). The objective of the working group was to define a profile of the protocol specifications needed to realize the use case of batch job scheduling of scientific/technical applications, i.e. the HPC use case. A profile specification defines no new schemas or protocols; rather it is intended

DOI: 10.1002/cpe

to describe the use of other existing protocols and schemas to meet a particular use case. Profiles generally contain statements that are used to clarify any ambiguities in referenced specifications that would cause interoperability issues, and will also restrict the usage of certain aspects of referenced specifications in order to promote interoperability of implementations. Profiles should not expand the scope of the referenced specifications.

The 'HPC Basic Profile, Version 1.0' (High Performance Computing Basic Profile (HPCBP), GFD-R-P.114) [18] references three other specifications: the 'JSDL, Version 1.0' (JSDL, GFD-R-P.56) [8], the 'JSDL HPC Profile Application Extension, Version 1.0' (GFD-R-P.111) [19], and the 'OGSA BES, Version 1.0' (BES, GFD-R-P.108) [9]. It also normatively defines a simple XML Schema called the BasicFilter extension. The purpose of HPCBP is to profile the use of these specifications in order to solve the base HPC use case. It was envisioned that additional, composable profiles would be defined to solve the other, common HPC use cases, such as the multi-organizational use case. It is worth exploring the specifications that make up the HPCBP and map them back to the characteristics of the base HPC use case in order to see how the base HPC use case is solved.

3.1. Job Submission Description Language (JSDL)

The JSDL (GFD-R-P.56) [8] is an XML-based language used to describe the requirements for computational jobs. The specification defines an XML schema from which the elements are taken to build up the job description document. JSDL does not provide elements for describing the actual status of the job after it has been submitted, and the JSDL specification does not describe any mechanism for submitting a JSDL document to a job management system. As such, JSDL is equally applicable to use within SOAP-based Web services, as to being consumed by a command-line or rich client program, which then interacts with a job management system in a proprietary way.

In the context of both, the base HPC use case and the base Grid HPC use case, JSDL provides the following capabilities:

- The ability to describe the resources needed to run the job. JSDL can describe resources such as number of CPUs, the amount of required memory, disk space requirements, and operating system and CPU model requirements.
- A place to hold a description of the run-time environment of the job, including attributes such as the executable to run, arguments to the executable, etc. The JSDL specification defines an element called POSIXApplication intended for this purpose, but the HPC Basic Profile made use of a different element, as described below.
- For the base Grid HPC use case, JSDL allows the description of data transfers that should occur before the job runs, and after the job has completed.

It was mentioned that JSDL defines POSIXApplication, which is an element used to describe the run time attributes of the job, including executable, *stdio* files, program arguments, environment, user name to run the job as, and various POSIX shell limits. This choice of elements was made based on the dominance of POSIX-based systems (e.g. Linux, Solaris, HP-UX, IRIX, AIX) within the Grid and HPC community. Since the HPC Basic Profile was also intended to be used on systems that do not support POSIX shell limits (i.e. Windows), a different application element,

DOI: 10.1002/cpe

called HPCProfileApplication, was defined to take the place of POSIXApplication. It is the same except for the exclusion of the POSIX limit elements. This ability to support multiple different application types using XML Schema extensibility was built into the JSDL specification from day one. It is envisioned that new application types such as parallel application, SQL query, or even web service invocation could be described using a JSDL document, with the appropriate application extension defined.

The HPC Basic Profile adds some restrictions to the JSDL that is accepted by an HPC Basic Profile compliant service. In basic JSDL, there are no elements within the schema that must be supported. The implementation is free to choose which ones to support. In the case of the HPCBP, in order to make sure that there is some level of interoperability, certain elements are restricted such that they must be supported by a compliant implementation. One of these elements is the HPCProfileApplication element as described above. The other JSDL elements required by the HPC Basic Profile are: JobDefinition, JobDescription, JobIdentification, JobName, JobProject, Application, Resources, CandidateHosts, ExclusiveExecution, OperatingSystem, CPUArchitecture, and TotalCPUCount. This set of elements were determined by the working group to meet the requirements for expressing job requirements derived from the characteristics of the base HPC use case.

3.2. Basic Execution Service (BES)

The BES specification (GFD-R-P.108) [9] defines the service operations used to submit and manage jobs within a job management system. BES defines two WSDL port types: BES-Management for managing the BES itself, and BES-Factory for creating, monitoring, and managing sets of activities. A BES manages 'activities', not just jobs, but the activity in the context of the HPCBP is an HPC or Grid HPC job. The notion of the activity is much like the notion of the application in JSDL. It is intended to be an extensibility point so that the operations defined by the BES have wider applicability than just batch job execution.

In terms of the base HPC and Grid HPC use cases, BES provides the bulk of the functionality. In terms of user interface, BES-Factory port type supplies the following operations:

- CreateActivity: The operation used to submit a job. It takes an ActivityDocument (a JSDL document) describing the activity (job) to create, and returns an identifier for the job in the form of a WS-Addressing EndpointReference (EPR) [20]. This EPR can be used in subsequent operations.
- GetActivityStatuses: This operation accepts a list of EPRs (previously returned from CreateActivity) and will return the state of each activity represented. The state model will be described below.
- GetActivityDocuments: This operation accepts a list of EPRs and will return an ActivityDocument for each EPR requested. The ActivityDocument just wraps a JSDL document. The BES specification does not mandate whether the returned JSDL document is the original JSDL document submitted to the service, or some 'reified' JSDL document that fills in more elements based on the status of the activity (e.g. a list of CandidateHosts could be supplied once an Activity has transitioned to the running state). The use of a reified JSDL document in this way is perhaps not in the 'spirit' of the JSDL specification, since this JSDL document is used to describe a job instance, not a job to submit, but it was considered a low effort way by some

DOI: 10.1002/cpe

implementers to provide some simple reporting of the status of the job. As such, there is no description as to which JSDL elements will be returned within this JSDL document.

- TerminateActivities: Takes a list of EPRs of activities that the BES should attempt to terminate (i.e. move to the Terminated state). Will return 'true' or 'false' for each activity depending on whether the operation was successful or not.
- GetFactoryAttributesDocument: This operation returns various attributes of the BES back to the client. These attributes include information about the BES itself, such as if it is accepting new activities. It also contains information about the resources that the BES has access to when scheduling jobs, and will return a list of Activity EPRs of the activities running within the BES container. Given that the list of either activity EPRs or the list of contained resources could be large, the HPC Basic Profile defines the BasicFilter extension, which is an XML element that can be passed to the GetFactoryAttributesDocument operation using the extensibility elements in the input document (a normally empty document). This BasicFilter allows the client to specifically request that the BES include or not include either the activity EPRs or the contained resource list in the output document from GetFactoryAttributesDocument. The filter makes explicit the fact that otherwise, the BES can choose to return or not return this information based on its implementation.

Using these operations, one can fulfill the requirements of the User Interface characteristics of the base HPC use case described earlier in the paper. All the required operations are covered.

BES also defines the state model for activities, shown in Figure 2 (as reproduced from [9]). The BES states map to the states described in the base HPC use case. The *Pending* BES state is the same as the previously described *queued* state, *Running* is the same, and *Finished* is the same. It is interesting to note that the mismatch in state names reflects the flexibility of the BES specification versus the concrete 'batch job management' use case described in the base HPC use case and profiled in the HPC Basic Profile. BES was intended to be applicable to workloads other than just batch jobs, thus the term 'activity' instead of 'job' (as 'job' usually describes some form of operating system process to be run), and a 'pending' rather than a 'queued' state ('queued' carrying the connotation of a batch queuing environment being used to manage jobs).

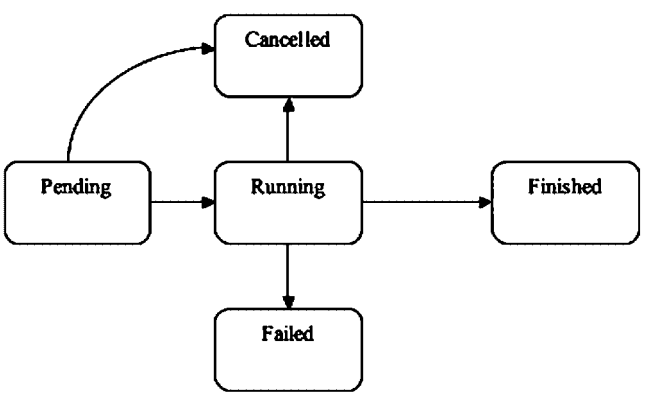


Figure 2. BES state model.

DOI: 10.1002/cpe

One difference between the BES states and the states described in the base HPC use case is the inclusion of two additional terminal states, *Terminated* and *Failed*, which allow the client to get a little more information about how the activity met its end. These extra terminal states, along with a rich set of operation faults for CreateActivity, allow the client to be made fault tolerant, in the sense that the client has the information needed to decide whether to resubmit a job depending on the terminal state.

Another feature of the state model is that it is extensible, but in a structured way that allows clients to reason about the state model, even if they do not understand the extensions. Later specifications and profiles can define 'sub-states' of the existing state model, and they can introduce state transitions between their sub-states, but they cannot define state transitions that violate the basic state model. Thus, a person could define a sub-state for *Running* and a sub-state for *Pending*, but the state extensibility model says that no transition could be made from my *Running*:*sub-stateA* to *Pending*:*sub-stateB*, since the transition from *Running* to *Pending* does not exist in the basic state model. This is a powerful model that builds in extensibility for meeting future use cases, while ensuring basic interoperability between basic clients and services.

For example, consider a state model extension that introduces three new sub-states to the *Running* state: *Staging-in*, *Executing*, and *Staging-out*. A BES client that recognizes this extension can make use of this information. A meta-scheduler could decide to pre-empt an activity if it is in *Running*:*Staging-in* or *Running*:*Executing*, but leave it to complete if it is in the *Running*:*Staging-out* sub-state. Another BES client that does not recognize this extension would just see that the activity was *Running*, and would wait for transitions between the base states. A simple workflow engine might implement dependencies based on the basic state model, and would then be 'future proofed' against extensions to the state model that do not really change this basic flow of states.

3.3. Basic HPC and Grid HPC use cases

One can see that the combination of JSDL and BES provides the capabilities needed to satisfy the basic HPC use case. How about the basic Grid HPC use case? It was noted that this use case differed in two aspects: the security model and the ability to stage data from one site to another, both before and after the job.

Security is directly addressed in the HPC Basic Profile. The Profile defines two mechanisms for authentication to be used when communicating with a BES endpoint. In both cases, the BES service is authenticated using SSL and server X.509 certificates. For client authentication, the HPC Basic Profile service either must support username and password-based authentication as described by WS-Security, or it must perform client authentication using an X.509 certificate at the SSL level. This second method is common to many inter-organizational Grid deployments, thus this supports the basic Grid HPC use case. Regarding authorization, it should be noted that in the broader context of Web services, authorization has not yet been standardized. As such, in the HPC Basic Profile, the account mapping functionality is not specified, but this is generally a BES implementation specific operation, and varies from middleware to middleware. While this clearly impacts interoperability, it would be premature to 'endorse' a particular approach to authorization.

In terms of data transfer, while the HPC Basic Profile does not preclude the use of JSDL DataStaging elements, it does not make them mandatory, thus in that sense the HPC Basic Profile cannot support the Grid HPC use case characteristic of data access. That said, there is currently a second

DOI: 10.1002/cpe

profile going through the OGF standardization process, produced by the HPC Profile WG, that aims to define how to use the JSDL DataStaging elements, and which defines the protocols that must be supported by a compliant endpoint, etc. This 'HPC File Staging Profile', when published, will provide the necessary capability to support the basic Grid HPC use case.

3.4. Experience and maturity

The HPC Basic Profile and associated specifications (JSDL and BES) are well advanced in the sense of the maturity of the specifications. There are a number of interoperable implementations of the specification.

The first round of interoperability testing of the HPC Basic Profile occurred at the IEEE/ACM Supercomputing 2006 conference. At this time, even though the HPC Basic Profile and BES specifications were not out of their draft stage, there was an interoperability demonstration performed between ten independent implementations of the specification (see [21]). At this stage the specification was firmly in the interoperability phase, and results of the interoperability demonstration and testing were used to refine the specification itself.

There was then a second round of interoperability testing that culminated in another interoperability demo at IEEE/ACM Supercomputing 2007. This round of testing was performed between seven independent implementations, but this time the specification had progressed from draft status to a proposed recommendation (see [22]). There were two interesting differences with this round of testing. One was the use of a 'service compliance checking' web site, hosted by the eScience research group at the University of Virginia (see [23]) that allowed service implementers to test their implementations with various combinations of both valid and invalid message formats. The second difference with this round of testing was that an early draft of the HPC File Staging Profile was also demonstrated by a subset of the interoperability demo participants. The results of the second round of interoperability tests have been documented in 'Interoperability Experiences with the HPCBP, Version 1.0' (GFD-E.124) [24], which is the experimental evidence necessary in OGF document process to bring the HPC Basic Profile, BES, and JSDL to full recommendation status (i.e. to become fully ratified standards).

At Supercomputing 2007, a number of vendors of job management systems declared that they would be implementing HPC Basic Profile service endpoints in their products, thus leading one to believe that the maturity of the HPC Basic Profile is at least at the adoption phase of its lifecycle.

4. SIMPLE API FOR GRID APPLICATIONS (SAGA)

SAGA provides a single, unifying, application-oriented programming interface for Grid applications. The SAGA Research Group was formed in 2005 within GGF, based on the observation that Grid application development was a challenging task, mostly because of the complexity and instability of the programming interfaces to Grid middleware, and the fact that Grid middleware exposed dissimilar APIs for similar functionality. While middleware APIs focus on providing access to functionality provided by middleware services, SAGA has been designed based on the needs of actual application use cases, as described in GFD-I.70, and thus has a focus on the Grid client implementer as described previously. SAGA was designed to be simple for developers to use, and

DOI: 10.1002/cpe

```
01: // Submitting a simple job and wait for completion
02: //
03: saga::job_description jobdef;
04: jobdef.set_attribute ("Executable", "job.sh");
05:
06: saga::job_service js;
07: saga::job job = js.create_job ("remote.host.net", jobdef);
08:
09: job.run();
10:
11: job.wait();
12:
13: if (job.get_state() == saga::job::Done )
14: {
15:   std::cout << "Job with ID: "
16:             << job.get_attribute("JobID")
17:             << " completed successfully" << std::endl;
18: }
```

Figure 3. SAGA job submission.

was intended to provide a familiar programming paradigm for developers. Figure 3 shows a simple SAGA code example of submitting a job to a possibly remote resource. SAGA's job attributes are based on JSDL and SAGA's job state model directly maps to the OGSA-BES state model. The job_service object is used to interface to a resource broker in general, or a BES service in particular. By this design, SAGA omits all middleware-related aspects, allowing the application to focus on the relevant functionality instead.

4.1. SAGA core

The SAGA API defines two general sets of packages: the functional packages and the 'look and feel' packages. The functional packages provide access to resources and services in a Grid. Reflecting the use cases from GFD-I.70, the SAGA Core specification (GFD-R-P.90) [25] provides functional packages for job submission and management, for physical and replicated files, for stream communication, and for GridRPC. Of great importance is the 'look and feel' packages that address non-functional aspects in a systematic and uniform way. SAGA provides such packages for authentication (security contexts) and authorization (permissions), for exception handling, application-level monitoring ,and steering, as well as asynchronous operation through SAGA's task model. The 'look and feel' packages apply to all functional packages, exposing a common programming style to an orthogonally designed and extensible API.

By design, SAGA abstracts the middleware services actually invoked in order to provide its functionality. At the same time, it leverages-related OGF standards for middleware services, exposing matured Grid 'terminology', where this makes sense. For example, SAGA provides a state model for tasks and jobs that closely mirror the state model from OGSA BES. In addition, SAGA creates job descriptions based on JSDL keywords, providing the application programmer with the

DOI: 10.1002/cpe

terminology widely accepted in the community. For data access, URL schemes are used to access common services while security credentials are completely treated as black box elements.

In the context of the characteristics of the basic HPC use case, the SAGA job package provides much of the capability required by the use case. The saga::job_service and saga::job classes provide methods for submitting a job to a scheduler (saga::job_service.create_job), querying a job by examining the attributes of an instance of class saga::job, cancelling a submitted job (saga::job.signal), and listing already submitted jobs (saga::job_service.list).

As mentioned above, the state model for a SAGA job is very similar to the OGSA BES state model, and thus meets the needs of the basic HPC use case for describing a job in *pending*, *running*, and *finished* states (with the same three terminal states as BES). Since an API user can query this current state of the job, the characteristic of client-driven fault-tolerance behaviour is supportable using the API.

The saga::job_description class allows one to specify the attributes of a job to be submitted, including a description of the resources required for execution, as well as the attributes of the execution environment itself. The attributes of saga::job_description are inclusive of the elements defined by JSDL, and thus can support the same capability.

And how well does SAGA support the added characteristics of the basic Grid HPC use case? Since each SAGA API 'session' is associated with some form of credentials, SAGA can easily be used with a multitude of security systems, that are then passed through to underlying middleware, thus any type of identity system (federated or mapped) is useable through the SAGA API. In terms of data transfer, SAGA provides a host of capabilities. First, much like in JSDL, one can define a set of file transfers that must be performed before and after running a job. This approach allows one to associate data transfer on a job-by-job basis. Second, there are APIs for accessing both physical and logical files managed by Grid file access and replica services, which allow users of the SAGA API to apply a finer grained control of data movement than data transfers in line with job execution.

4.2. Experience and maturity

After a long phase of careful design, re-design, and consensus building in the community, SAGA is now on a fast track towards standards maturity. It always was an important asset of the SAGA working group to have a prototypical reference implementation available during the API design process. This allowed the group to evaluate design choices and their implications on both expressiveness and feasibility directly in the process. The SAGA Core specification [18] became a proposed recommendation in January 2008. Immediately afterwards, the reference implementation prototype in C++ was made available by members of the SAGA working group for early adopters. Despite the short time since the publication of the SAGA Core specification document, the API is already seeing a lot of community uptake. For example, the DEISA infrastructure is using SAGA as its job submission interface, OMII-UK is providing the SAGA API to its middleware stack, and the XtreemOS Grid operating system is providing SAGA as its API. Further uptake is in progress; SAGA supports for EGEE's gLite middleware, for the Globus toolkit, and for OGSA BES is currently under development.

For API specifications, the notion of compliance with the API is measured somewhat differently than with protocols. With an API, a particular program that makes use of the API should

DOI: 10.1002/cpe

operate in a semantically identical manner no matter which particular implementation of the API is used. The reference implementation of the specification provides the semantic reference for how other API implementations should behave. Given that SAGA has the reference implementation, other implementations of SAGA can easily be examined for compliance with the specification. It would thus appear that SAGA falls somewhere between the 'interoperability' and 'adoption' phases, as there is a reference implementation that can be used to test other implementations for compliance.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown how clear consumer-oriented use cases of a well-understood problem provide the basis for rapid (for a standards process) successful development and adoption of standards. This has been illustrated for computationally oriented Grids by considering access to HPC resources in the context of the basic HPC and basic Grid HPC use cases.

For Grid client implementers, SAGA provides a programmatic access to HPC resources. Grid service implementers can implement protocol level access to HPC resources using the HPC Basic Profile. Both SAGA and the HPC Basic Profile fulfill the requirements of the basic HPC use case and (with the addition of the HPC File Staging profile) can both fulfill the requirements of the basic Grid HPC use case as well. SAGA and the HPC Basic Profile are mature specifications that have surpassed the 'draft' phase. Furthermore, SAGA has a reference implementation for testing compliance, and the HPC Profile is well within the adoption phase, as evidenced by the adoption of the HPC Basic Profile within various Grid middleware stacks from commercial vendors and open source projects. This represents a significant milestone in the usability of HPC resources and the transportability of the applications that are being built to exploit this increasingly common resource.

There is much work that both the HPC Profile team and the SAGA team intend to pursue. The HPC Basic Profile specification is already the basis of a number of increasingly mature extensions. In addition to the 'HPC File Staging' extension, the 'Advanced Filter' extension improves the control a client has over the information returned about the computational resource, while the 'Application Template' extension removes the burden from the user of specifying site specific configuration information which is instead provided by the HPC Basic Profile service provider. Both of these are mature working group drafts.

Currently, the SAGA working group is continuing its activities in two directions. One important stream of activities is the specification of language bindings of the (language independent) SAGA Core specification. The first language binding specification is currently under way. It will provide SAGA syntax and semantics for the Java language. Similar efforts for C, C++, and Python will follow. Simultaneously, work is under way to extend the scope of the API, based on the experiences gained so far through the reference implementations. Owing to the orthogonal API design, such extensions can be done in the form of well-defined (functional) extension packages. The first such extension package is addressing the important aspect of service discovery. Likely candidates that are being considered as future extensions include a messaging extension, Grid checkpointing and recovery, or a DRMAA API 'compatibility' package.

DOI: 10.1002/cpe

ACKNOWLEDGEMENTS

The authors would like to express their gratitude to many members of the Open Grid Forum (and Global Grid Forum before that) for their contributions to the HPC Basic Profile WG and the SAGA WG/RG (particularly the JSDL WG and the BES WG). We additionally thank Glenn Wasson for his contributions to the HPC Basic Profile WG and Andre Merzky for being the driving force behind SAGA.

REFERENCES

1. Open Grid Forum. http://www.ogf.org [26 November 2008].
2. Schopf JM. Ten Actions When SuperScheduling. Grid Forum Document GFD-I.4. July 2001.
3. Roy A, Sander V. Advance Reservation API. Grid Forum Document GFD-E.5. 23 May 2002.
4. Schwiegelshohn U, Yahyapour R. Attributes for Communication between Scheduling Instances. Grid Forum Document GFD-I.6. December 2001.
5. Open Grid Forum High Performance Computing Profile WG (HPCP-WG). http://forge.ogf.org/sf/projects/ogsa-hpcp-wg [26 November 2008].
6. Open Grid Forum Simple API for Grid Apps RG (SAGA-RG). http://forge.ogf.org/sf/projects/saga-rg [26 November 2008].
7. Open Grid Forum Simple API for Grid Apps Core WG (SAGA-CORE-WG). http://forge.ogf.org/sf/projects/saga-core-wg [26 November 2008].
8. Savva A (ed.). Job Submission Description Language (JSDL) Specification, Version 1.0. Grid Forum Document GFD-R.056. 7 November 2005.
9. Foster I, Grimshaw A, Lane P, Lee W, Morgan M, Newhouse S, Pickles S, Pulsipher D, Smith C, Theimer M. OGSA Basic Execution Service Version 1. 0. Grid Forum Document GFD-R-P.108. 8 August 2007.
10. Distributed Management Task Force (DMTF) Common Information Models (CIM) Standards. http://www.dmtf.org/standards/cim/ [26 November 2008].
11. Open Grid Forum GLUE WG (GLUE). http://forge.ogf.org/sf/projects/glue-wg [26 November 2008].
12. Mach R, Lepro-Metz R, Jackson S, McGinnis L (eds.). Usage Record—Format Recommendation. Grid Forum Document GFD-R-P.098. September 2006.
13. Simple Object Access Protocol (SOAP) 1.1, W3C. http://www.w3.org/TR/soap11 [8 May 2000].
14. Web Services Description Language (WSDL) 1.1, W3C. http://www.w3.org/TR/wsdl. [15 March 2001].
15. Theimer M, Smith C, Humphrey M. HPC Job Scheduling: Base Case and Common Cases. Grid Forum Document GFD-I.100. 2006.
16. Jha S, Merzky A. A Collection of Use Cases for a Simple API for Grid Applications. Grid Forum Document GFD-I.70. 9 May 2006.
17. Catlett C. Global Grid Forum Documents and Recommendations: Process and Requirements. Grid Forum Document GFD-C.1. June 2001, revised April 2002.
18. Dillaway B, Humphrey M, Smith C, Theimer M, Wasson G. HPC Basic Profile, Version 1.0. Grid Forum Document GFD-R-P.114. 28 August 2007.
19. Humphrey M, Smith C, Theimer M, Wasson G. JSDL HPC Profile Application Extension, Version 1.0. Grid Forum Document GFD-R-P.111. 28 August 2007.
20. Web Services Addressing 1.0—Core. W3C recommendation. http://www.w3.org/TR/ws-addr-core/ [9 May 2006].
21. Open Grid Forum SC2006 HPC Profile WG Interoperability Demonstration Status Wiki. https://forge.gridforum.org/sf/wiki/do/viewPage/projects.ogsa-hpcp-wg/wiki/SC2006WikiPage [26 November 2008].
22. Open Grid Forum SC2007 HPC Profile WG Interoperability Demonstration Status Wiki. https://forge.gridforum.org/sf/wiki/do/viewPage/projects.ogsa-hpcp-wg/wiki/HomePage [26 November 2008].
23. University of Virginia eScience Group High Performance Computing Basic Profile Interoperability Tester v1.0. https://opteron4.cs.virginia.edu:45885/FormServerTemplates/HPCBasicProfileComplianceTester.aspx [26 November 2008].
24. Wasson G. Interoperability Experiences with the High Performance Computing Basic Profile (HPCBP), Version 1.0. Grid Forum Document GFD-E.124. 21 February 2008.
25. Goodale T, Jha S, Kaiser H, Kielmann T, Kleijer P, Merzky M, Shalf J, Smith C. A Simple API for Grid Applications (SAGA). Grid Forum Document GFD-R-P.90. 15 January 2008.

DOI: 10.1002/cpe