Editors:
Morris Riedel, JSC/FZJ

http://forge.gridforum.org/sf/projects/pgi-wg
30 April 2012

OGF – Production Grid Infrastructure

European Middleware Initiative Execution Service

Version 1.0

Category: INFORMATIONAL

Status of This Document

This document provides information about the execution service specification achieved by the European Middleware Initiative. The document is given as an input of planned open standard specifications for production Grid infrastructures covering job submission and management as well as job description. It does not define any standards, requirements, or technical recommendations. Distribution is unlimited.

Abstract

The Production Grid Infrastructure (PGI) working group works on a well-defined set of standard profiles, and additional standard specifications if needed, for job and data management that are aligned with a Grid security and information model that addresses the needs of production Grid infrastructures. These needs have been identified in various international endeavors and are in many cases based on lessons learned obtained from the numerous activities in the Grid Interoperation Now (GIN) community group. Therefore, PGI can be considered as a spin-off activity of the GIN group in order to feed back any experience of using early versions of open standards (e.g. BES, JSDL, SRM, GLUE2, UR, etc.) in Grid production setups to improve the standards wherever possible. This particular document captures the a reasonable stable draft of the execution service specification of the European Middleware Initiative (EMI) that has been created based on many identified PGI concepts. A complementary implementation of this draft specification is provided by the EMI project including adoptions in ARC, gLite, and UNICORE. The goal of this document is to capture community practice and to have a foundation for a set of important standard specification updates to be addressed by the PGI set of profiles and/or specifications of the corresponding related groups (e.g. BES, JSDL, etc.).

Contents

1. Introduction

This document provides the interface specification, including related data models such as state model, activity description, and resource and activity information, of an execution service, matching the needs of the EMI production middleware stack composed of EMI products including ARC, gLite and UNICORE components. This service therefore is referred to as the EMI Execution Service (or "ES" for short).

This document is a continuation of the work previously knows as the GENEVA, then AGU ("ARC, gLite UNICORE") [1], then PGI execution service. As a starting point, the v0.42 of the "PGI Execution Service Specification" (doc15839) was used. This particular PGI document is based on the EMI-ES specification version 1.07.

The *targets for this specification are the so-called Computing Elements (CEs)*, that is Grid services providing access to computing resources usually localized at a site (e.g. a cluster, a computing farm), usually managed by a Local Resource Management System (LRMS). Higher level services (such as workload managers, brokering services, or workflow systems) are out of scope.

This document covers the following items:

- Interfaces to create and manage activities
- Activity description language ("EMI-ADL"), an XML dialect for describing the activity including data staging and resource requirements
- Data staging capabilities
- Activity related information, that can be retrieved by clients
- Resource related information, required for clients to access information relevant for making brokering and resource selection decisions
- Delegation, needed to implement data staging

The relationships between these "elements" are shown in Figure 1 below.

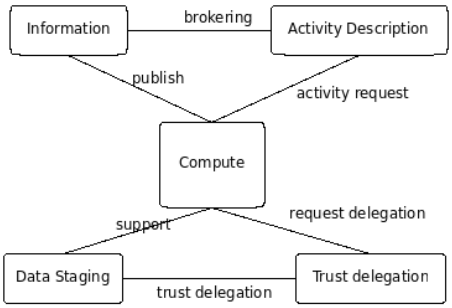

Figure 1: Relationship between execution service elements

An Execution Service can be implemented using different security setups (see Security Considerations). Therefore it is assumed that the clients obtain security setup info through out-of-band mechanisms which are not covered by this specification.

Resource and activity information is provided according to the GLUE2 specification 47 with some extensions, more specifically its XML rendering.

Several items were considered during the preparation of this version of the specification, but were postponed to a later version. A full list of these items is provided in Section 11.

1.1 Notational Conventions

The key words “MUST”, “MUST NOT,” “REQUIRED,” “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in RFC 2119 (see http://www.ietf.org/rfc/rfc2119.txt).

We consider an “activity” within this document as consisting of a computational job plus any associated data movement to and from the job. The executable part of the activity is referred to as “user job”, or just as “job”.

1.2 Architecture

The EMI ES consists of two main modules. The first module is the **Activity-Factory**, responsible to create activities and manage resource (CE) information. It implements the following port-types (for each one the relevant operations are listed):

- Create port-type: CreateActivity
- ResourceInfo port-type: GetResourceInfo, QueryResourceInfo
- Delegation port-type: InitDelegation, PutDelegation, GetDelegationInfo

The second module is the **Activity-Manager**, responsible to manage activities and activity related information. It implements the following port-types (for each one the relevant operations are listed):

- ActivityManagement port-type: GetActivityStatus, GetActivityInfo, NotifyService, PauseActivity, ResumeActivity, CancelActivity, WipeActivity, RestartActivity
- ActivityInfo port-type: ListActivities, GetActivityStatus, GetActivityInfo
- Delegation port-type: InitDelegation, PutDelegation, GetDelegationInfo

For convience, some (GetActivityStatus and GetActivityInfo) operations can be accessed via both the ActivityManagement andActivityInfo port-types.

It must be stressed that the ResourceInfo port-type refers **only to information related** to the Computing Element. Among this information there are also the endpoint(s) of the Activity-Manager(s) coupled to that Activity-Factory. Instead it doesn't include information about activities created through that Activity-Factory (not even the list of activity ids).

1.3 Data-Staging Functionality

This section describes the data staging functionality of the EMI Execution Service.

As entities to be considered for data staging, we refer to files.

We consider here only data staging done before or after job execution. Of course the job itself is free to do any data staging during its execution, but this is not something implemented using functionality provided by the EMI ES.

We call **stage-in directory** the place offered to clients to upload data. This is also the place to which the ES may pull input data in the “server data pull” stage-in scenario (see below). This directory is created by the ES and either returned as part of the CreateActivity response or obtainable from GetActivityInfo. There is a single stage-in directory per activity, possibly accessible by multiple protocols (in future specifications it will be investigated the possibility to have different stage-in directories for the different data objects). The Execution Service MUST provide this directory when entering the PREPROCESSING phase of states.

We call **stage-out directory** the place where the output data are collected and can be retrieved from the client. It is used for the “client data pull” scenario (see below). Output data for the “server data push” scenario (see below) can also appear in this directory. This directory is created by the ES and either returned as part of the Createactivity response or obtainable from GetActivityInfo. There is a single stage-out directory per activity, possibly accessible by multiple protocols (in future specifications it will be investigated the possibility to have different stage-out directories for the different data objects). The Execution Service MUST provide this directory upon entering the POSTPROCESSING phase of states.

We call **session directory** the directory on the worker node where the user job is executed. The provision of access to the end user to this session directory (which in general can not be accessible from the Execution Service) is optional. The access to the session directory enables client to access and modify the content of the session directory between stage-in and stage-out states. The session directory address is published as part of activity info. The possibility of accessing the session directory provides some sort of interactivity. The Execution Service MUST provide this directory in the PROCESSING phase of states.

For what concerns **stage-in**, that is the staging of input data to the execution service done before job execution, there are two possible scenarios:

Server data pull: the ES pulls the needed data from the specified (in the activity description document) sources and makes them available later in the session directory. These data MAY be first uploaded into the stage-in directory. This “server data pull” scenario requires delegation support, since the server has to act on behalf of the activity owner. Server data pull takes place in the PREPROCESSING or PROCESSING-RUNNING state. The “SERVER-STAGEIN” attribute is used to report about this server-initiated data transfer.

Client data push: The client uploads the data into the stage-in directory. The activity description MUST contain a flag informing the server that the client wishes to push data (attribute ClientDataPush of the DataStaging element, see section 42). When done with data push, the client explicitly tells the server to continue processing the activity via the NotifyService operation.
The data to be pushed MAY be declared in the activity description. In this case, client implementations MUST stage-in all the declared files. Data can be pushed when the stage-in directory has been created, and the activity is in a state with the CLIENT-STAGEIN-POSSIBLE attribute set.

For what concerns **stage-out**, that is the staging of output data from the execution service done after job execution, there are two possible scenarios:

Server data push: the ES pushes the relevant data to the specified (in the activity description document) targets. The “server data push” scenario requires delegation support, since the server has to act on behalf of the activity owner. Takes place when the SERVER-STAGEOUT state attribute is set.

Client data pull: the client pulls the data from the stage-out directory of the ES. The downloading MUST take place in states with the CLIENT-STAGEOUT-POSSIBLE state attribute set. The data to be pulled MUST be declared in the activity description.

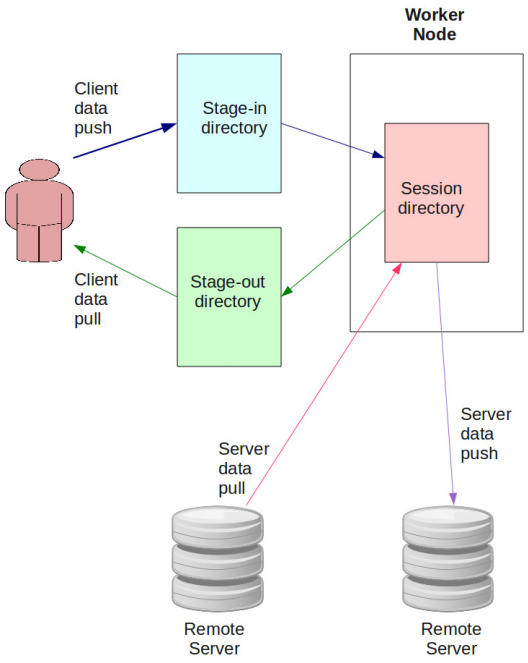

Figure 2: Data-staging: directories and actors

The Execution Service should properly advertise data staging capabilities as part of the resource information. Because one can't expect all implementations to be capable of supporting the same set of data transfer capabilities there must be a way for execution service to announce its capabilities and their possible combinations (e.g.: stage-out-to-https).

For what concerns the handling of failures:

- If there is a failure during “server pull” stage-in, the activity is moved to FAILED state and the user job is not run at all.
- If there is a failure during execution of the user job, the user can decide for each dat aobject whether stage-out should be performed, see the UseIfFailure attribute (section 44).
- If there is a failure during the stage out of a file, activity will go into the FAILED state. At any rate the stage out process must be done for all the other data output objects (i.e. it must not stop at the first data stage out failure)

If the user job is cancelled, the ES can optionally (configurable in the activity description document, see attribute UseIfCancel, section 44) try to carry out the stage-out phase. Possible options that could be specified for each data object in case of job cancellation are:

- Don't try the stage-out
- Let the output objects available in the stage-out directory (even for the “server data push” scenario) for manual download

1.4 Delegation

Clients need to be able to pass delegation tokens to the ES. The ES uses the token to access third party services, mostly storage services. This is needed in particular to support “server data pull” and “server data pull” staging scenarios (see Sect. 1.3).

There are two main delegation tokens in use:

- X.509 proxies: the content of the proxy is well agreed while the transfer of the proxy is not agreed (i.e. no standard exists for that). Two solution to “transfer” proxies are:
- GSI mechanisms, i.e. via a modified SSL protocol which is not compatible with off-the-shelf, industry-standard SSL
- Service-specific interfaces

SAML: transferred as part of SOAP communication. The content of SAML token is NOT standardized (EMI may offer a common profile). The transfer mechanism is instead standardized.

There are several “consumers” of X.509 delegation tokens, e.g. gridftp, SRM, LFC, LB. Instead currently there aren't yet EMI services (apart from the UNICORE services) that are able to consume SAML tokens.

For this reason, the scope of delegation in this specification refers only to X.509 proxy token delegation (SAML tokens will be supported in future versions of the EMI ES specification).

Only RFC3820 proxies are allowed. Any extension is allowed. Extensions marked as “must” must be understood, otherwise the service must throw failure.

The ES MUST support the direct “push” of X.509 proxy tokens to the ES directly from the client.

Section 6 describes the details of the delegation model.

2. Interface: Creation Port-Type

This section describes the Creation Port-Type and its operations.

2.1 CreateActivity Operation

Functionality

The operation is used to request the creation of a vector of activities where each activity is described by an activity description document. The service MUST perform a certain amount of validation. These validations MAY be performed after the activity creation i.e. non-validated activities can be created. The service creates an instance of each activity that is identified by a globally unique ID, the activityID assigned by the service. The activityID should not be assumed to contain service address, i.e. a activity is not directly addressable by its activityID.

Data-Staging Implications

One key feature of the EMI Execution Service is the support for client initiated data stage-in to the stage-in directory, as explained in Section 1.3.

To enable client data push,

- The service MAY immediately return a data-staging-in location as part of the response of the CreateActivity() operation, or
- The service MUST expose the data-staging-in location information via the GetActivityInfo() operation as soon as the activity enters a state with the CLIENT-STAGEIN-POSSIBLE attribute set.

If the client wants to perform client initiated data-staging-in,

- The client MUST specify this in the activity description
- After retrieving the stage-in directory, the client is able to push the data to the execution service

The following Figure 3 illustrates the activity creation and validation process, and the client interactions required for client data push.

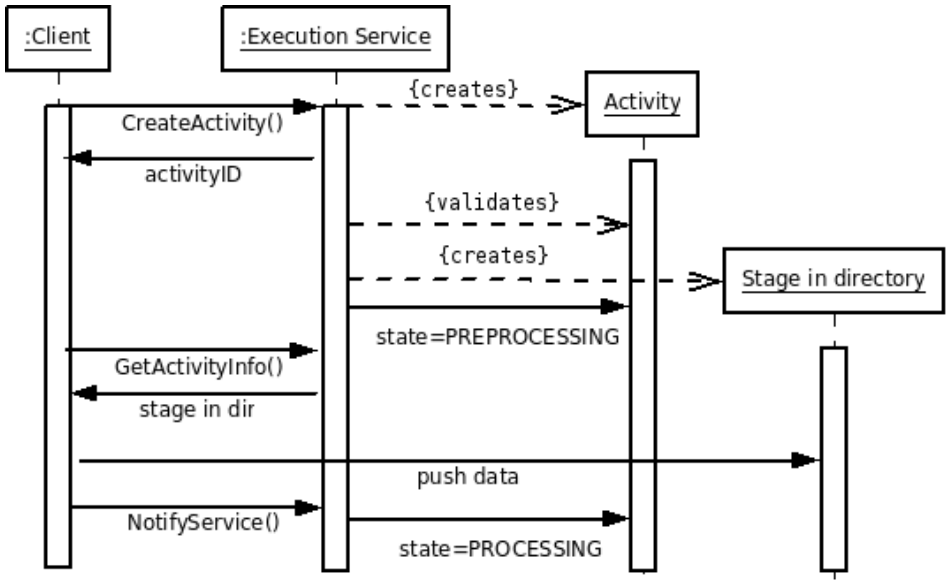

Figure 3: Activity creation, validation and client data push

State Model Implications

The service MUST perform all mandatory and optional validation steps (see below) of the activity description document as part of the ACCEPTED-VALIDATING state, before the activity can enter the PREPROCESSING state. Some of these validation steps MAY be performed after the CreateActivity response was returned.

Request

The CreateActivity() operation has one mandatory input parameter:

- Vector of activity descriptions composed of one activity description document per element that is compliant with Section 9 of this specification.

Response

The response of the CreateActivity operation returns a vector of values in the same order as in the request; each value is as follows:

For those activities for which the activity creation was successful the execution service returns the following:

- It MUST return a global unique activityID assigned to the activity by the service
- It MUST return the address of the Activity-Manager that MUST be contacted to manage the activity.
- It MUST return the current activity state
- It MAY return the estimated time of the next state change
- It MAY return the stage-in directory and/or the session directory and/or the stage-out directory, which are accessible via possibly multiple protocols

For those activities for which the activity creation failed the execution service SHOULD return one of the following

- For those activities where the service could perform validation and the activity description is represented by a malformed XML document, the environment wherein the service exists MUST return a corresponding fault.
- For those activities where the service could perform validation and the activity description is not compliant with the EMI-ADL schema, the service must return a response containing the message error InvalidActivityDescriptionFault
- For those activities where the service could perform validation and the activity description contains a semantic error that occurs during the semantic validation step (see below), the service MUST return a response message error InvalidActivityDescriptionSemanticFault
- For those activities where the activity description requests a capability that is not provided by the service, the service MUST return a response containing a message error UnsupportedCapabilityFault.

In case the number of submitted activities is too large, the service MUST return VectorLimitExceededFault, which contains the services's vector limit. In this case the service MUST NOT attempt to create any activities.

Faults

- InvalidActivityDescriptionFault
- InvalidActivityDescriptionSemanticFault
- UnsupportedCapabilityFault
- VectorLimitExceededFault
- InternalBaseFault

Activity Description Document Validation

The service takes a activity description document as input. The service MUST perform a certain amount of validation steps during the ACCEPTED-VALIDATING state. In case of validation failure the activity enters the VALIDATIONFAILURE secondary state of FAILED terminal state. The list of mandatory validation steps is as follows:

- XML Validation: Check whether XML is a valid XML document (i.e. well-formed and such like).
- Schema validation: Check against the schema of the service
- Semantic validation: This validation includes that the service is capable to understand the values of the XML elements of the activity description. It also includes checks that cannot be proofed against the schema.
- Service capability validation: several features are optional and thus a service MAY not be supporting some of the activity description elements. The service capability validation checks if all of the requested mandatory activity attributes are supported by the service. For example,
    - Notifications
    - specific runtime environments

The service MAY also perform an optional validation functionality that is called ‘matchmaking’, which checks whether the requested resources match the provided resources by the service.

It is important that activity description elements marked as critical MUST NOT be ignored by the service, a corresponding fault MUST be thrown in the case that the service does not support this particular client request.

3. Interface: ResourceInfo Port-Type

The resource information related with this port-type refers only to information related to the Computing Element, represented via GLUE2 resource attributes. It also contains the endpoint(s) of the Activity Manager(s). Instead it doesn't include information about activities (not even the list of activity ids).

3.1 GetResourceInfo Operation

Functionality

This operation provides the resource information according to the GLUE2 computing service model. Activity records are not part of the resource information.

This operation returns the resource information of a Execution Service, the model that encapsulates the resource information MUST abide by the GLUE2 computing service model. The main entity of the model being endorsed is the ComputingService, which further aggregates the ComputingManager, ComputingEndpoint, ExecutionEnvironment, and ApplicationEnvironment. Moreover, the Activity Manager(s) is a part of the response of the GetResourceInfo operation, since it is not specifically defined in the GLUE2 schema, thus it can either be exposed as a generic Service or defined as an Extension within the resource document being returned.

The resource information XML instance document is created (see section**Fehler! Textmarke nicht definiert.**) to succinctly describe the resources according to GLUE2 model.

Data-Staging Implications

None

State Model Implications

None

Request

None

Response

The response includes an XML document with resource info according to GLUE2 model with some extensions (see section **Fehler! Textmarke nicht definiert.**).

Faults

- InternalResourceInfoFault
- ResourceInfoNotFoundFault
- AccessControlFault
- InternalBaseFault

Example documents are presented in section 13.4.

3.2 QueryResourceInfo Operation

Functionality

This operation provides flexible access to query the full resource description formatted according to the XML rendering of GLUE2 information model. The Query operates on the resource information document (no activities!). The operation can be used to obtain any kind of resource characteristics through a query expressed in one of the supported query languages.

Query Languages

The Execution Service MAY support multiple query languages in addition to the mandatory one. The supported query dialects MUST be published as part of the resource description via the Capability.ComputingEndpoint property of the GLUE2 model.

Resource Model

The GLUE2 information model provides an XML rendering of the computing service concept. The QueryResource operation MUST be applied on the full GLUE2 document excluding the ComputingActivity elements.

Data-Staging Implications

None

Request

The request includes the following information:

- A query expression given in one of the supported query languages
- The type of the query language

An Execution Service MUST support Xpath 1.0 and MAY support additional query languages such as XQUERY, SQL, sparql and custom defined ones (e.g. python-based).

Response

The response includes the following information:

- The result of the query
- In case of a failed query, an appropriate FAULT is returned

Faults

- NotSupportedQueryDialectFault
- NotValidQueryStatementFault
- UnknownQueryFault
- AccessControlFault
- InternalBaseFault

4. Interface: ActivityManagement Port-Type

This section describes the ActivityManagement Port-Type including its operations.

4.1 GetActivityStatus Operation

Functionality

This operation provides the state information according to the state model of a vector of activities. The service MUST publish the primary state together with all the state attributes.

Data-Staging Implications

Since this method returns also the data-staging related state attributes, it plays an important role for deciding when it is possible to do client data push and client data pull.

Request

The request includes the following information

- A vector of activity identifiers

Response

The response includes the following information:

The following information is provided for each of the activities:

- activity identifiers
- the corresponding state
- a timestamp
- an optional description (or message)

For those activities where the state cannot be requested (for example, because the activity identifier does not exist), an appropriate fault structure should be returned.

Faults

- UnknownActivityIDFault
- UnableToRetrieveStatusFault
- OperationNotPossibleFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

4.2 GetActivityStatus Operation

Functionality

This operation provides the activity information of a vector of activities according to the GLUE2 activity model. As the set of affected activities MUST be explicitly provided in the request, no queries (i.e. XPATH, XQuery) are used in order to obtain a subset of the information according to the GLUE2 activity schema.

Data-Staging Implications

If the Execution Service possesses the corresponding capabilities then the activity information MUST be able to publish the session directory and stage-in and stage-out directories related information. This operation MAY be used to retrieve that information.

State Model Implications

The state information is a mandatory part of the activity information. The GLUE2 model enables the publication of activity states in multiple state models. The Execution Service MUST publish the activity state according to the EMI ES state model and MAY publish additional state model values as well.

Request

The request includes the following information:

- A vector of activity identifiers
- A vector of GLUE2 activity attributes (or supported extensions to GLUE2) which are requested. In case if this parameter is not specified then the full activity record MUST be returned.

Response

The response includes the following information:

- The detailed activity information about each of the activities is provided as a pair of activity identifiers and the corresponding activity information elements. The response information MUST contain the mandatory activity attributes according to the GLUE2 schema and the optional attributes specified in the request.
- For those activities where the activity information cannot be returned (for example, because the activity identifier does not exist or an invalid GLUE2 attribute was requested), an appropriate fault structure should be returned.

Faults

- UnknownActivityIDFault
- UnknownGlue2AttributeFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

Example XML documents are provided in section 13.5.

4.3 PauseActivity Operation

Functionality

This operation requests to stop execution of the activity (stop whatever the service was doing). It may be not possible for service to perform stop immediately. For example it takes time to propagate a stop request to the underlying batch system. In such case the service MUST inform the client that the operation is going to happen asynchronously and optionally provide estimated time.

Depending on implementations and execution environment it may be not possible to perform a stop in a particular state. In such a case either the request fails and the service informs the client about that, or the service performs the request asynchronously and stops processing in the next state. The last case should be acceptable only in very short transitional states.

If during application of PauseActivity request failure happens and the activity is transferred to one of states applicable for failure processing, the request to stop processing is still applicable unless that new state is TERMINAL.

The following list describes the effect of PauseActivity on various states:

ACCEPTED state - any activity validation and provisioning stops. Once the activity processing is paused the service assigns the activity CLIENT-PAUSED attribute.

PREPROCESSING state – any provisioning and data staging activities are stopped. Once the activity processing is paused the service assigns the activity CLIENT-PAUSED attribute.

PROCESSING-ACCEPTING – the job submission procedure to batch system is stopped. Once the job processing is paused the service assigns the job CLIENT-PAUSED attribute. Note: it is highly likely that due to short lifetime of this state service will choose to stop job processing in next PROCESSING-QUEUED state.

PROCESSING-QUEUED – the batch system is informed to pause processing of the job. If the batch system does not support such functionality, the request fails. Once job processing is paused the service assigns the job the CLIENT-PAUSED attribute.

PROCESSING-RUNNING – the batch system is informed to pause execution of the job. If the batch system does not support such functionality the request fails. Once job processing is paused the service assigns the job the CLIENT-PAUSED attribute.

POSTPROCESSING – any deprovisioning and data staging activities are stopped. Once job processing is paused the service assigns the job the CLIENT-PAUSED attribute.

TERMINAL – not applicable.

Request:

- vector of jobids

Response:

The response of the PauseActivity() operation returns a list of values; each value is as follows:

For those activities which can be paused, the execution service should return information on the timing in which the pause will be done (i.e., whether the activity has already been paused, or pause will be attempted in the future):

- Time estimation: ETP - Estimated time to pause, with the following special values:
- 0 means already paused (i.e. immediately),
- undefined if the service is not able to perform an estimation

For those activities which cannot be paused, OperationNotPossible is returned

If corresponding activity does not exist - ActivityNotFoundFault

Faults

- OperationNotPossibleFault
- ActivityNotFoundFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

4.4 ResumeActivity Operation

Functionality

This operation requests to stop execution of the activity (stop whatever the service was doing). It may be not possible for service

This operation is the counterpart of PauseActivity. It instructs the service to remove the CLIENT-PAUSED attribute of the activity state. This will resume activities stopped as result of a PauseActivity operation. This operation may be processed asynchronously. Whether activity processing is resumed exactly at the place where it was stopped or there is some activity repeated is implementation specific. But the final result must be independent of whether the job was paused or not.

Request:

- vector of jobids

Response:

The response of the ResumeActivity() operation returns a list of values; each value is as follows:

For those activities which can be resumed, the execution service should return information on the timing in which the resuming will be done (i.e., whether the activity has already been resumed, or resume will be attempted in the future).

- Time estimation: ETP - Estimated time to resume, with the following special values:
- 0 means already resumed (i.e. immediately),
- undefined if the service is not able to perform an estimation

For those activities which cannot be resumed (not paused), OperationNotPossible is returned

If corresponding activity does not exist ActivityNotFoundFault is returned

Faults

- OperationNotPossibleFault
- ActivityNotFoundFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

4.5 NotifyService Operation

Functionality

This operation is used to notify the service that the client completed an operation. It is in particular used to notify the service when the client completed the client data push or client data pull.

Data-Staging Implications

As discussed in section 1.3, there are two possible approaches for data stage in:

- Server data pull
- Client data push

In the client data push, the user is supposed to upload the needed data in the stage-in directory as soon as the activity reaches a status with the CLIENT-STAGEIN-POSSIBLE flag. When the client has completed this data push, it MUST inform the service (with the NotifyService operation) that it completed the stage in. Otherwise the activity won't proceed its processing.

Similarly, there are two possible approaches for data stage out:

- Server data push
- Client data pull

In the client data pull, the user is supposed to retrieve the output data from the stage-out directory as soon as the activity reaches a status with the CLIENT-STAGEOUT-POSSIBLE flag. When the client has completed the data push, it CAN (it is not mandatory) inform the service (with the NotifyClient operation) that it completed the stage out.

Request

This operation accepts as input a vector where each element of it contains:

- activity id
- A string, which specifies what the client wants to notify. For the first EMI-ES specification, this string can assume one of the possible values:
    - CLIENT-DATAPULL-DONE: this is used to notify the service that the client has completed the client data pull
    - CLIENT-DATAPUSH-DONE: this is used to notify the service that the client has completed the client data push

Response

The response of the NotifyService operation returns a vector of values; each value is as follows:

- For those activities for which the notification has been accepted, an acknowledgement is returned
- For those activities for which the notification could not be accepted, an appropriate fault structure is returned

Faults

- OperationNotPossibleFault
- OperationNotAllowedFault

- InternalNotificationFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

4.6 CancelActivity Operation

Functionality

This operation is used to request the cancellation of a set of activities while this operation does not wait for the cancellation of these activities. The execution service cancels the requested activities which in turn enter the final state "Cancelled" according to the state model. The cancel means that active data-staging processes SHOULD be cancelled and active executions in LRMS MUST be cancelled.

Data-Staging Implications

Any data-staging activities that are processed during the invocation of the CancelActivity() operation SHOULD be immediately cancelled.

In the context of data, any kind of data that has been marked as stage-in or any kind of temporarily generated data is not available when activities enter the 'cancelled' state.

For what concerns staging-out, if a job is cancelled, the user can configure for each data object whether the ES should perform the stage-out This is done using the UseIfCancel attribute, see section 44.

Data objects that have been already staged by the service to an external target (i.e. remote storage elements) are not affected by these operations.

State Model Implications

The cancel operation is applicable to any state of the state model except the terminal states. Upon successful cancellation the activity enters one the *.CANCEL states, depending on the current main state, i.e. PREPROCESSING.CANCEL, etc.

Request

The input parameter is a vector of activity identifiers. Other mechanisms of specifying a subset of the affected activity identifiers (e.g. filtering, grouping) are considered to be out of scope, because of simplicity of the service interface itself.

Response

The response of the CancelActivity() operation returns a list of values; each value is as follows:

For those activities which can be cancelled, the execution service should return information on the timing in which the cancellation will be done (i.e., whether the activity has already been cancelled, or cancellation will be attempted in the future);

- Time estimation: ETC - Estimated time to cancellation, special values: 0 means already cancelled (i.e. immediately), undefined if the service is not able to perform an estimation

For those activities which cannot be cancelled (for example, because the activity identifier does not exist), an appropriate fault structure should be returned. OperationNotPossibleFault

If the cancel operation request is performed on states that do not allow for a 'cancel' transition then a fault should be thrown. OperationNotAllowedFault

Faults

- OperationNotPossibleFault
- OperationNotAllowedFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

4.7 WipeActivity Operation

Functionality

This operation can be used to request the complete removal of a set of activities, including the activity-related information as well as the removal of all temporary data associated with these activities. As a result of this operation, the activity disappears from the service environment and thus no further operations can be invoked on wiped activity identifiers.

Data-Staging Implications

In the context of data, any kind of data (i.e. stage-in directory, stage-out directory, session directory) or any kind of temporarily generated data is NOT available after the invocation of the WipeActivity() operation.

State Model Implications

This operation is only allowed on any final (TERMINAL) state according to the state model. If the activity is not in any final state yet, the service must throw a fault ActivityNotInTerminalStateFault

Request

The input parameter is a vector of activity identifiers. Other mechanisms of specifying a subset of the affected activity identifiers (e.g. filtering, grouping) are considered to be out of scope, because of simplicity of the service interface itself.

Response

The response of the WipeActivity() operation returns a list of values; each value is as follows:

For those activities which can be wiped out, the execution service should return information on the timing in which the activities will be no longer in the system (i.e., whether the activity has already been wiped out, or it will be attempted in the future);

- Time estimation: ETW - Estimated time to wipe out the activity , special values: 0 means already wiped (i.e. immediately), undefined if the service is not able to perform an estimation

For those activities which cannot be wiped out (for example, because the activity identifier does not exist), an appropriate fault structure should be returned. OperationNotPossibleFault

If the wipe operation request is performed on states that do not allow for an invocation of the WipeActivity() operation then a fault should be thrown. OperationNotAllowedFault

Faults

- ActivityNotInTerminalStateFault
- OperationNotPossibleFault
- OperationNotAllowedFault
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

4.8 RestartActivity Operation (Optional)

Comment

The RestartActivity operation MAY be implemented by the execution service to allow failed activities to be restarted. Its may intention is to take advantage of already delivered, produced data and to recover from errors caused by problems external to Execution Service.

Functionality

This operation is requesting service to restart an activity at the state where it failed. The service SHOULD change the state of the requested activities to the state at which processing would resume at point as close to failure point as possible (implementation specific).

Data-Staging Implications

Restarting activity which failed during data staging phase MUST result in failed staging parts to be retried.

State Model Implications

The ChangeActivity operation can be only used for transitions according to the state model with the restriction of one-step transitions only.

Request

This operation accepts as input a vector of activity identifiers.

Response

The response of the RestartActivity operation returns a vector of values; each value is as follows:

Time estimation in which the restart will be done - 0 if already performed or undefined if can't etimate.

For those activities which cannot be restarted (for example, because the activity identifier does not exist or recovery to failed state not possible), an appropriate fault structure should be returned. OperationNotPossibleFault.

Faults

- OperationNotPossible
- VectorLimitExceededFault
- AccessControlFault
- InternalBaseFault

5. Interface: ActivityInfo Port-Type

This section describes the ActivityInfo Port-Type and its operations.

5.1 ListActivities Operation

Functionality

The main purpose of this operation is to allow the user to retrieve the list of his/her Activities (as allowed by access control) handled by the EMI Execution Service. The operation returns just a list of the ActivityIds while detailed information about those Activities can be obtained by invoking the getActivityInfo. Moreover the ListActivity provides as set of optional parameters useful for filtering the Activities by date (e.g. created in a well defined time window) and/or by current status. It is also possible to specify a threshold value for limiting the result size (e.g. size of matched Activities).

Request

The ListActivity operation accepts as input the following NOT mandatory parameters:

- fromDate, toDate: the activity creation time window
- statusList: the list of possible states
- limit: an non-negative integer giving the maximum number of ActivityIds within the returned list

If no parameters are specified, the operation returns the full list of ActivityIds belonging to that user.

Otherwise the IDs of those activities are returned that were created in the given window AND are in one of the listed states.

Response

The response of the ListActivity operation returns a list of ActivityIds or an appropriate fault if some input parameter is illegal.

An additional boolean flag “truncated” indicates that the result list was truncated.

Faults

- InvalidParameterFault
- AccessControlFault
- InternalBaseFault

5.2 GetActivityStatus Operation

This operation is described in section 4.1 above.

5.3 GetActivityInfo Operation

This operation is described in section 4.2 above.

6. Interface: Delegation Port-Type

Delegation is a critical ingredient for the Execution Service, since data staging must be performed on behalf of clients. Since delegation is critical for other services as well (such as data management services), a common solution for all the EMI services is desirable. At the time of writing, this delegation solution for all of EMI was not available. Thus, this specification contains a temporary solution for use by the execution service. In future versions of this specification, the EMI delegation solution will be adopted.

The delegation port-type provides operations by which the clients can temporarily convey their X.509 proxy certificates (that MAY carry also attributes) to the execution service. The delegated credential MAY be used in further delegations and additional extensions MAY be inserted during this process. This port-type is extremely important to support some parts of the data staging functionality (it is needed for “data server push” and “data server pull” data staging scenarios: see sect. 1.3).

The interface described in this section describes the direct delegation of RFC3820 X.509 proxies between clients and ES.

6.1 InitDelegation Operation

Functionality

The *InitDelegation* operation starts the delegation procedure by asking for a certificate signing request from the server. The server answers with a certificate signing request which includes the public key for the new delegated credentials. The *PutDelegation*() method of the Delegation portType has to be called to complete the delegation procedure.

Data-Staging Implications

The DelegationID response MAY be used in the DataStaging element of the activity description in order to assign a delegated credential (once the delegation process was completed) to a datastaging operation to be performed by the Execution Service on behalf of the client (in the “server data push” and “server data pull” data staging scenarios).

State model Implications

The two-step delegation process MUST be performed at least once before the invocation of the CreateActivity operation which uses the DelegationID passed within the DataStaging JSDL block [2]. The DelegationID MAY be (re-)used in multiple CreateActivity() operation invocations.

Request

- Credential type (currently “RFC3820 “, i.e. X.509 proxy), others TBD): required
- Renewal ID (optional): the ID of a previous delegation that the client wants to renew.
- Initdelegationlifetime (optional)

Response

The response includes the following information:

- DelegationID. This string is used to uniquely refer to this delegation session. The server assigns the delegation ID. In case of a renewal the server must re-use the previous delegation ID submitted in the request.
- Certificate signing request. For X.509 proxy type it is a string which contains an X.509 certificate signing request.
- A FAULT in case of an internal service error.

Faults

- InternalServiceDelegationFault
- InternalBaseFault

6.2 PutDelegation Operation

Functionality

The *PutDelegation* operation completes the delegation procedure by sending the signed proxy certificate along with the *DelegationID* to the server. The signed certificate is based on the certificate signing request previously retrieved together with the *DelegationID* via an *InitDelegation* operation invocation.

Data-Staging Implications

The delegated credential MAY be used by the Execution Service to carry out datastaging operations on behalf of the user. The delegated credential is assigned to the data transfer via a dedicated activity description element of the DataStaging block of the EMI-JSDL.

In general, it is possible that the same user delegates different kind of credentials (with different delegation IDs) to the same Execution Service. This could be useful, for example, if the user belongs to different Virtual Organizations (VOs) and wants to delegate credentials bound to different VOs to the same service. Different VOs could be necessary to access data on different Storage Elements (SEs).

State model implications

The two-step delegation process MUST be performed at least once before the invocation of the CreateActivity operation which uses the DelegationID passed within the DataStaging JSDL block. The DelegationID MAY be (re-)used in multiple CreateActivity() operation invocations.

Request

The request includes the following information:

- Credential type (currently: “RFC3820”, i.e. X.509 proxy, others TBD): required
- the DelegationID required
- the RFC 3280 style proxy certificate, signed by the client, required

Response

The response includes the following information:

- a SUCCESS string in case of successful PutDelegation operation.
- or if a DelegationID is unknown to the service an UNKNOWNDELEGATIONID fault to be thrown

Faults

- UnknownDelegationIDFault
- AccessControlFault
- InternalBaseFault

6.3 GetDelegationInfo Operation

Functionality

This operation is used to get information about a previously created delegation. For the first version of the specification, the only information that is returned by this operation is the lifetime of the delegated token.

Data-Staging Implications

No direct implication.

State model implications

No direct implication, since this operation is not related to activities.

Request

The request includes the following information:

- the DelegationID (required)

Response

The response includes the following information:

- the lifetime of the proxy (MUST)
- the issuer DN (MAY)
- the subject DN (MAY)
- if a DelegationID is unknown to the service an UnknownDelegationIDFault fault to be thrown

Faults

- UnknownDelegationIDFault
- AccessControlFault
- InternalBaseFault

7. Activity State Model

This section describes the state model of the EMI Execution service, i.e. it specifies the possible states of activities created using the EMI ES and lists the possible transitions between states.

The state model is the external one, i.e. intended for clients. Internally, the service implementation might use a different state model.

7.1 State Definitions

The EMI ES state model consists of states and state attributes. An activity can only be in one state but can have multiple state attributes.

The main states can be grouped into 5 phases:

ACCEPTED phase: the activity has been created and is being validated. This phase is represented by state **ACCEPTED**.

PREPROCESSING phase: the activity environment, including data is being prepared. This phase is represented by state **PREPROCESSING**.

PROCESSING phase: the job is being submitted to and processed by the underlying system or it is already handled by the batch system. This phase is split into the following states:

PROCESSING-ACCEPTING – intermediate state representing the time slot while the Execution Service communicates with the underlying batch system.

PROCESSING-QUEUED – this state indicates that the job was accepted by the batch system, but the payload is not yet running.

PROCESSING-RUNNING – this state indicates that the job was accepted by the batch system and the payload is running.

POSTPROCESSING phase: the job has left the batch system. It is possibly doing stage-out, releasing resources (de-provisioning). This phase is represented by state **POSTPROCESSING**.

TERMINAL phase: there is no more activity by the service, the activity is in a final state. Output data is available for client data pull. This phase is represented by state **TERMINAL**.

Each state may be assigned multiple attributes. The purpose of attributes is to provide information about particular functions being performed by the service. It is main source of information for clients to choose action to perform. There may be few or no attributes assigned to the current state. Not every attribute may be assigned to every state. The following table shows the applicability of attributes to states. Below the table there is a description of every attribute.

The following attributes are defined:

- VALIDATING informs that service is performing validation of job request.
- SERVER-PAUSED means server stopped job processing due to some internal decisions. This flag can be raised and removed only by service itself.
- CLIENT-PAUSED is raised by client through submitting a PauseActivity request. This is a flag which can be removed by client through submitting a ResumeActivity request. Job processing is stopped and will not continue until the CLIENT-PAUSED attribute is removed.
- CLIENT-STAGEIN-POSSIBLE and CLIENT-STAGEOUT-POSSIBLE are indicating that client can access stage-in/stage-out location. This is a flag which can be removed by client through submitting NotifyService request. The service will stop job processing at the point where it can't continue waiting for the CLIENT-STAGEIN-POSSIBLE to be removed. It must be noted that there is no need for removing the CLIENT-STAGEOUT-POSSIBLE attribute.
- PROVISIONING and DEPROVISIONING refer to any preparations before and after job goes to batch system not related to data staging.

- SERVER-STAGEIN and SERVER-STAGEOUT are representing service performing server data pull and server data push.
- BATCH-SUSPEND refers to situation when batch system decides to suspend execution of payload
- APP-RUNNING denotes execution of payload specified in job description. This attribute is meant to distinguish between execution of payload specified by client and other actions which service may choose to delegate to batch system.
- *-CANCEL attributes inform that job was cancelled on client request. First part of name refers to phase of job when cancellation request was executed.
- *-FAILURE attributes inform that job processing failed. First part of name refers to phase of job where failure was detected. Additionally VALIDATION-FAILURE refers to job failed due to failure to validate job request. APP-FAILURE means failure of specified payload.
- EXPIRED indicates that the job was canceled because its expiration time has been exceeded. The expiration time is optionally set in the activity description, see section **Fehler! Textmarke nicht definiert.**).

	ACCEPTED	PREPROCESSING	PROCESSING-ACCEPTING	PROCESSING-QUEUED	PROCESSING-RUNNING	POSTPROCESSING	TERMINAL
VALIDATING	X						
CLIENT-PAUSED	X	X		X	X	X	
CLIENT-STAGEIN-POSSIBLE	X	X					
SERVER-PAUSED	X	X		X	X	X	
PROVISIONING		X					
SERVER-STAGEIN		X		X	X		
BATCH-SUSPEND				X	X		
APP-RUNNING					X		
SERVER-STAGEOUT					X	X	
DEPROVISIONING						X	
CLIENT-STAGEOUT-POSSIBLE						X	X
PREPROCESSING-CANCEL						X	X
PROCESSING-CANCEL						X	X
POSTPROCESSING-CANCEL						X	X
VALIDATION-FAILURE						X	X
APP-FAILURE						X	X
PREPROCESSING-FAILURE						X	X
PROCESSING-FAILURE						X	X
POSTPROCESSING-FAILURE						X	X
EXPIRED							X

Table 1: Applicability of attributes to states

7.2 State Transitions

The optimal state transition chain is the following:

ACCEPTED → PREPROCESSING → PROCESSING-ACCEPTING → PROCESSING-QUEUED → PROCESSING-RUNNING → POSTPROCESSING → TERMINAL

The following table represents the other possible state transitions: for each state of the first row, the possible target states are listed

Allowed transitions	ACCEPTED	PREPROCESSING	PROCESSING-ACCEPTING	PROCESSING-QUEUED	PROCESSING-RUNNING	POSTPROCESSING	TERMINAL
	PREPROCESSING	PROCESSING-ACCEPTING	PROCESSING-QUEUED	PROCESSING-RUNNING	PROCESSING-QUEUED	TERMINAL	
	TERMINAL	POSTPROCESSING	PROCESSING-RUNNING	POSTPROCESSING	POSTPROCESSING		
		TERMINAL	POSTPROCESSING	TERMINAL	TERMINAL		
			TERMINAL				

From **TERMINAL** state with ***-FAILURE** attributes following failure recovery transitions are allowed:

VALIDATION-FAILURE → NO

APP-FAILURE → PROCESSING

PROCESSING-FAILURE → PROCESSING

PREPROCESSING-FAILURE → PREPROCESSING

POSTPROCESSING-FAILURE → POSTPROCESSING

The implementation of such "failure recovery transitions" is optional (i.e. some services can be able to support some recoveries)

8. Resource and Activity Representation

The EMI Execution Service, as one of its key features, has the capability of publishing information about the resource characteristic exposed by the service and the detailed properties of the activities being managed by the service.

The resource information is available through the ResourceInfo port type. The port type provides the GetResourceInfo and QueryResourceInfo operations. The former returns the full resource information while the latter enables flexible queries over the same information content. Resource information is defined as information related to the service characteristics only, therefore activity information is completely excluded. The resource description used by the EMI ES follows the GLUE2 model47. In particular, the resource information model of EMI-ES is based on the ComputingService and the generic Service elements of the GLUE2 XML rendering. The ComputingService object encapsulates all the “computing element” characteristics while the generic Service element is used to publish the associated Activity-Manager endpoints. In order to encapsulate some of the EMI features certain extensions were necessary:

a possibility to provide information about service features, specifically

- notification capabilities
- supported data staging protocols
- exclusive execution of jobs on worker nodes
- remote session directory access including supported protocols

parallel environments

session directory location

Section 13.4 provides the detailed schema-level example of the resource information model used within the ResourceInfo port type.

Activity information, that provides an elaborate description of activity properties, is available through the ActivityInfo and ActivityManagement port types via the GetActivityInfo operation. Activities of EMI-ES are modelled by the GLUE2 information model. In particular, the ComputingActivity element of the XML rendering of the GLUE2 is used as the basis for the EMI-ES activity information document. Necessary extensions include operation and job state change history. Section 13.5 provides the detailed schema-level example of the activity information model used within the GetActivityInfo operation.

8.1 GLUE2 Extension

For providing EMI-ES related information GLUE2 Extension consisting of following elements is specified:

StageInDirectory – contains URL which can be used by the client to stage in data as described in section **Fehler! Textmarke nicht definiert.**. This element is mandatory while activity state includes CLIENT-STAGEIN-POSSIBLE attribute.

StageOutDirectory – contains URL which can be used by the client to stage out data as described in section **Fehler! Textmarke nicht definiert.**. This element is mandatory while activity state includes CLIENT-STAGEOUT-POSSIBLE attribute.

SessionDirectory – contains URL which can be used by the client to access activity execution directory as described in section **Fehler! Textmarke nicht definiert.**. This element is optional and its presence signifies possibility to access the **session directory** of activity.

ComputingActivityHistory – represents historical record about activity status changes. Its exact content is still to be defined. This element is optional.

ComputingActivityProgress – contains percentage of activity completeness. Exact mapping of presented value to activity state is not defined and number is provided for reference only. This element is optional.

For representing activity state in GLUE2 model the State elements are used. By convention states are represented by string made of prefix and value separated by colon. For EMI-ES states **emies** prefix is used followed by the name of the main state as defined in section **Fehler! Textmarke nicht definiert.**. Presentation of the state attributes is yet to be defined if needed.

9. Activity Description

The rendering of the EMI execution service requires the definition of the following data structures: activity description, activity state model, execution service and activity resource models. This section defines the activity description.

Since the execution service interface is a web-service interface, the activity description rendering is in XML. Its normative XML schema is defined as part of the web service WSDL and XML schema.

There are several possible use cases of job description instances:

a) for describing a job to a end-user client

b) for controlling resource selection (matchmaking) by a client or a brokering service

c) to pass job parameters to the execution service (what to run, data staging, environment info)

All of these cases have different semantics and may require additional elements.

Therefore, *this specification defines the job description for direct consumption by the execution service*. Additional information and structures to be used for match making can be added, but will be ignored by the execution service. Job description instances for clients/brokers are a different level, and outside the scope of this specification.

9.1 Processing of the Activity Description by the execution service

The execution service uses the activity description to generate a set of steps (for example, encoded into an executable script) that is then processed by the underlying resource management and/or operating system. The exact script is of course implementation specific, but the following tasks will typically be performed by the execution services:

- adding resource information to be processed by the batch system (e.g. how many CPUs should be allocated)
- adding server data pull (stage in), for example using GridFTP or other data transfer tools
- setting environment variables, including those contained in the activity description
- generating the main command line from the requested runtime and parallel environments, and the user-specified executable and arguments
- generating the main command line from the executable and arguments contained in the activity description
- generating code to process the user job's exit code
- adding server data push (stage out)

Since the job description contains several abstractions, for example the Software, RuntimeEnvironment or ParallelEnvironment, this concretization is a mandatory step.

9.2 Optional Elements of the Submitted Activity Description

The following section defines the individual XML elements that make up a activity description instance. The importance of elements varies, and the following criticality levels can be defined:

“critical”: it is hard requirement. The service MUST provide this feature, satisfy the requirement otherwise the activity must be rejected (during the validation phase)

“non-critical”: it is a soft requirement, which the service SHOULD honor. However, the activity still MUST be run at a service even if the service is not capable satisfying the requirement.

As an example, one could consider user notifications on state changes. Notifications by SMS may be considered “nice to have”, while notification by email might be mandatory.

The default level is: “critical”, meaning an element MUST be honored by the service.

The implementation of the criticality level is by using an XML attribute called “optional”, with the values “true“ and “false”. If not set, it is interpreted as “false”, and the XML element is assumed to be critical.

Elements supporting this feature are marked “OPT-IN” in the following sections.

9.3 Activity Description Language

The following section defines the individual XML elements that make up a activity description instance. The importance

This section defines the individual elements that can be used to compose activity descriptions.

9.3.1 High-level structure

An activity description is composed of four major blocks, each of which is described in detail in the following sections.

```
<ActivityDescription>
  <ActivityIdentification... />?
  <Application .../>?
  <Resources .../>?
  <DataStaging .../>?
</ActivityDescription>
```

9.3.2 Types

In addition to the basic types such as string, integer, datetime and URI, the activity description utilizes the following types:

9.3.2.1 ExecutableType

This complex element denotes an executable having *path* and optional *argument* subelements. An optional attribute *failIfExitCodeNotEqualTo* allows to specify that the Execution Service should treat the job as failed if the exit code is not equal to the specified value. Typically, this value will be set to “0” to indicate that exit code of zero indicates successful execution. By default, the Execution Service MUST NOT check the exit code of the user application to decide whether to continue processing.

- *path:* the path to the executable, relative to the session directory. Multiplicity is one.
- *argument*: optional subelements specifying the arguments, multiplicity is zero or more.
- *failIfExitCodeNotEqualTo*: optional integer-valued attribute

9.3.2.2 SoftwareRequirement

The *SoftwareRequirement* structure provides the general envelope to express logical relation of Software requests.

The *SoftwareRequirement* element specifies the software requirements of a job. It MAY contain multiple *Software* elements. This element MAY contain a Boolean that specifies that all OR one of its child elements has to be satisfied. The multiplicity of this element is zero or more. There is no default value of this element.

9.3.2.3 Software

The *Software* is a triplet of strings. The multiplicity is zero or more. There is no default value of this element. The structure is composed of *Family, Name* and *Version* string elements.

9.3.2.4 BenchmarkType

The benchmark type is composed of the name of the benchmark as defined by GLUE2 and the non-negative integer benchmark value.

The benchmark name is an open enumeration with values of the GLUE2 Benchmark_t type:

- bogomips
- cfp2006
- cint2006

- linpack
- specfp2000
- specint2000

9.3.3 ActivityIdentification

The main goal of this element is to name the activity and define its type and identify the activity in general. The multiplicity of this element is zero or one.

9.3.3.1 Name

This optional string element MAY be specified by a user to name the activity. It may not be unique to a particular activity description, which means that a user MAY specify the same ActivityName for multiple activity descriptions. Some project defines their own format for the ActivityName in order to categorize and explicitly define the particular version of activity they run. However the recommended way to attach user specified categories to the job is to use ActivityAnnotation element.

This element has no default value. The multiplicity of this element is zero or one.

9.3.3.2 Description

This optional longer string element may contain a longer textual description of the activity. This element has no default value. The multiplicity of this element is zero or one.

9.3.3.3 Type

This optional element provides a classification of the activity in compliance with GLUE2. The default value of this element is *single*. The multiplicity of this element is zero or one. The type of this element is an enumeration with the following elements:

- *collectionelement*: an activity submitted as part of a collection of individual activities which do not
- communicate among them,
- *parallelelement*: an activity submitted as part of a collection of individual activities which communicate among them,
- *single*: an individual stand-alone activity,
- *workflownode*: an activity submitted as part of a workflow.

9.3.3.4 Annotation

This optional string-valued element is for human readable comments, tags for free grouping or identifying different activities. This element has no default value. The multiplicity of this element is zero or more.

9.3.4 Application

The main goal of this block is to explicitly describe the executed application and its software environment. This job description block is mandatory and its multiplicity is one.

9.3.4.1 Executable

This optional element of type ExecutableType is specifying the main executable of the job. Multiplicity is zero or one. If no executable is specified, it is assumed that the selected runtime environment provides an executable.

9.3.4.2 Input

This optional element is a string specifying the input (Standard Input) for the Executable. The Input element is a filename which should be relative to the session directory of the job. There is no default value of this element. The multiplicity is zero or one.

9.3.4.3 Output

This optional element is a string specifying the output (Standard Output) for the Executable. The Output element is a filename which should be relative to the session directory of the job. There is no default value of this element.. The multiplicity is zero or one.

9.3.4.4 Error

This optional element is a string specifying the error output (Standard Error) for the Executable. The Error element is a filename which should be relative to the session directory of the job. There is no default value of this element. The multiplicity is zero or one.

9.3.4.5 Environment

This optional element specifies the operating system environment variables which should be defined at the execution service in the execution environment of the job. It consists of a *Name Value* pair of strings. The multiplicity of this element is zero or more with strict ordering. There is no default value of this element.

Name
This mandatory string element defines the name of the environment variable. Multiplicity is one. There is no default value of this element.

Value
This mandatory string element defined the value of the environment variable. Multiplicity is one. There is no default value of this element. It is not recommended to use system specific notion, macro etc as a value of this element.

9.3.4.6 PreExecutable

This optional element of type ExecutableType specifies an command that should be executed before invoking the user's application. Multiplicity is zero or more.

9.3.4.7 PostExecutable

This optional element of type ExecutableType specifies an command that should be executed after invoking the user's application. Multiplicity is zero or more.

9.3.4.8 RemoteLogging

The optional elements specifies a logging service to send reports about job. There is no default value of this element. Multiplicity is zero or more. In case of multiple elements computing service MUST try to send reports to all specified logging services. It is not a failure if communication fails. It is up to a user to make sure the requested logging service accepts reports from the set of computing service he or she intends to use.

The content of this element and information sent may be very specific to type of logging service. So we only define minimal set of information. One example is the OGSA Resource Usage Service (RUS) [https://forge.gridforum.org/sf/projects/rus-wg] which accepts Usage Records (UR) [http://www.ogf.org/documents/GFD.98.pdf].

OPT-IN

9.3.4.8.1 ServiceType

This mandatory string element specifies the type of logging service.

9.3.4.8.2 URL

If applicable this optional element specifies the endpoint at which the service may be contacted.

9.3.4.9 ExpirationTime

The element is optional and specifies the date and time after which the processing of a job MUST be cancelled. Jobs not completed before the expiry date MUST be cancelled by the service. Multiplicity is zero or one. There is no default value of this element.

OPT-IN.

9.3.4.10 WipeTime

The requested duration the job MUST stay in the terminal phase before it MAY be wiped by the service. There is no default value of this element. Multiplicity is zero or one.

OPT-IN.

9.3.4.11 Notification

This optional element defines the request in custom format for notifications on job state change. Multiplicity is zero or more. There is no default value of this element. The service advertises its notification capabilities

OPT-IN.

9.3.4.11.1 Protocol

This mandatory element specifies the protocol to be used for notification. Multiplicity is one. The initial list of protocol is “email”. (TBD: others, e.g. “ws-notification”). This field will be used for validating whether the execution service has the required capability.

9.3.4.11.2 OnState

This optional element denotes the state which should trigger the notification. This can be one of the valid primary states of the job (ACCEPTED, etc). The default value of this element is “TERMINAL”, i.e. by default, notifications will be sent when the job enters a terminal state. Multiplicity is zero or more.

9.3.4.11.3 Recipient

The string-valued address to send notifications to, e.g. “user@domain.org”. Multiplicity is one or more.

9.3.5 Resources

The optional complex resource element describe the resource requirements of the job. Multiplicity is zero or one.

9.3.5.1 OperatingSystem

This optional complex element specifies the operating system required for the user job. Its type is SoftwareRequirement. Multiplicity is zero or more, where multiple values are interpreted as giving alternatives (i.e. “OR” semantics are implied). There is no default value of this element.

In case of OperatingSystem the Family element of the Software structure embedded in the SoftwareRequirement is open enumeration with values of the GLUE2 OSFamily_t type:

- *linux*: Family of operating systems based on Linux kernel
- *macosx*: Family of operating systems based on MacOS X
- *solaris*: Family of operating systems based on Solaris
- *windows:* Family of operating systems based on Windows
- *aix*: AIX
- *centos*: CentOS
- *debian*: Debian
- *fedoracore*: RedHat Fedora

- *gentoo*: Gentoo Linux
- *leopard*: Mac OS X 10.5 (Leopard)
- *linux-rocks*:
- *mandrake*: Mandrake
- *redhatenterpriseas*: RedHat Enterprise Server
- *scientificlinux*: Scientific Linux
- *scientificlinuxcern*: Scientific Linux CERN
- *suse*: SUSE
- *ubuntu*: Ubuntu
- *windowsvista*: Microsoft Windows Vista
- *windowsxp*: Microsoft Windows XP

In case of OperatingSystem the Name element of the Software structure embedded in the SoftwareRequirement is open enumeration with values of the GLUE2 OSName_t type:

- *aix*: AIX
- *centos*: CentOS
- *debian*: Debian
- *fedoracore*: RedHat Fedora
- *gentoo*: Gentoo Linux
- *leopard*: Mac OS X 10.5 (Leopard)
- *linux-rocks*:
- *mandrake*: Mandrake
- *redhatenterpriseas*: RedHat Enterprise Server
- *scientificlinux*: Scientific Linux
- *scientificlinuxcern*: Scientific Linux CERN
- *slackware:* Slackware Linux
- *suse*: SUSE
- *ubuntu*: Ubuntu
- *windowsvista*: Microsoft Windows Vista
- *windowsxp*: Microsoft Windows XP

9.3.5.2 Platform

Optional element specifies the platform architecture required for the user job. Multiplicity is one or one. There is no default value of this element. Its is an open enumeration with a values of the GLUE2 Platform_t type:

- *amd64*: AMD 64bit architecture
- *i386*: Intel 386 architecture
- *itanium*: Intel 64-bit architecture
- *powerpc*: PowerPC architecture
- *sparc*: SPARC architecture

9.3.5.2.1 Coprocessor

This is an open enumeration that specifies the type of coprocessing unit that is available.

- *CUDA*: Compute Unified Device Architecture, a parallel computing architecture developed by NVIDIA
- *FPGA*: Field programmable gate array

OPT-IN.

9.3.5.3 NetworkInfo

This optional element specifies the type of the interconnect, the internal network connection available inside the computing element. Multiplicity is zero or one. There is no default value of this element. Multiplicity is zero or one. Its is an open enumeration with the values of GLUE2 NetworkInfo_t type:

- *100megabitethernet*: Network based on 100 MBit/s Ethernet technology
- *gigabitethernet*: Network based on 1 GBit/s Ethernet technology
- *10gigabitethernet*: Network based on 10 GBit/s Ethernet technology

- *infiniband*: Network based on Infiniband technology
- *myrinet*: Network based Myrinet technology

OPT-IN

9.3.5.4 NodeAccess

The optional element defines the required connectivity of the executation node. Multiplicity is zero or one. If it is not defined, then network connection is not required for the user job. It is an enumeration with the following values:

- *inbound*: inbound network is required for the user job
- *outbound*: outbound network is required for the user job
- *inoutbound*: both directions are required for the user job

9.3.5.5 IndividualPhysicalMemory

This optional element is a integer value specifying the amount of physical memory required to be available on every node of the computing element used by (a multi slot) job. The amount is given in bytes. Multiplicity is zero or one.

9.3.5.6 IndividualVirtualMemory

This optional element is a integer range value specifying the amount of virtual memory required to be available on every node of the computing element used by (a multi slot) job. The amount is given in bytes. Multiplicity is zero or one.

9.3.5.7 DiskSpaceRequirement

This optional integer element specifies the total required disk space of the job in bytes.

9.3.5.8 RemoteSessionAccess

A boolean to request remote access to the session directory. Multiplicity is zero or one. If it is not defined that the user not interested to access session directory remotely (default is false).

9.3.5.9 SlotRequirement

This optional complex element specify the requested count of slots and its distribution for multi-slot jobs. Multiplicity is zero or one.

9.3.5.9.1 NumberOfSlots

This mandatory integer range element specifies the total number of slots to allocate on the batch system.

The term “Slot” is used to denote a logical CPU visible to and allocatable by the resource management system. It may correspond to a physical CPU, a physical CPU core or a virtual CPU or core, depending on the hardware capabilities. Multiplicity is one.

9.3.5.9.2 SlotsPerHost

This optional integer element specifies the number of slots to be allocated on each single host.

An optional attribute “useNumberOfSlots” can be set to “true” to indicate that the value of the NumberOfSlots element should be used. In this case the value of the SlotsPerHost element MAY be left empty, and if it is given anyway, it MUST be ignored by the execution service.

9.3.5.9.3 ExclusiveExecution

This optional boolean element specifies whether a host should be allocated for excusive use by the user job.

Each site has a default value for this, which should be advertised through GLUE2.

9.3.5.10 QueueName

This optional string element defines the name of the preferred queue. Multiplicity is zero or one. There is no default value of this element.

9.3.5.11 IndividualCPUTime

This optional element specifies the number of CPU seconds requested for each slot of the user job. There is no default value of this element. Multiplicity is zero or one.

9.3.5.12 TotalCPUTime

This optional element specifies the total number of CPU seconds requested for the user job. It is the sum of the times requested for each individual slot. There is no default value of this element. Multiplicity is zero or one.

9.3.5.13 WallTime

This optional element is the wall clock time requested for the user job, from the start of the first process until the completion of the last process. Multiplicity is zero or one.

9.3.5.14 Benchmark

This optional element of type BenchmarkType specifies a required minimum benchmark value (as performance indicator). The multiplicity is zero or one.

OPT-IN.

9.3.5.15 RuntimeEnvironment

This optional SoftwareRequirement element defines the runtime environment required by the user job. Multiplicity is zero or more. There is no default value of this element.

A runtime environment MAY provide a default executable for the job, i.e. the end-user need not specify an executable, but may rely on the default one for the given runtime environment.

The available runtime environments MUST be advertised in the services's description via GLUE2.

OPT-IN.

9.3.5.15.1 Name

The mandatory name of the runtime environment.

9.3.5.15.2 Version

The optional version of the runtime environment.

9.3.5.15.3 Option

This optional element specifies an option that should be enabled in the selected runtime environment. For example, it can be used to enable debugging or verbose mode. Multiplicity is zero or more.

9.3.5.16 ParallelEnvironment

The parallel environment is used to specify the execution environment for parallel jobs. Multiplicity is zero or one.

If a ParallelEnvironment element is present, the execution service MUST create the correct invocation for the requested parallel environment. The execution service MAY also add environment variables and path settings as appropriate.

The parallel environments available at an execution service MUST be advertised through the GLUE2 description of the execution service.

9.3.5.16.1 Type

This optional element defines the type of multi-slot application. There is no default value of this element. It is string valued, with the following initial set of values taken from the SPMD extension 47 for the JSDL

- *MPI:* Any MPI environment
- *GridMPI:*The GridMPI environment
- *IntelMPI*:The Intel MPI environment
- *LAM-MPI*:The LAM/MPI environment
- *MPICH1*:The MPICH version 1 environment
- *MPICH2*: The MPICH version 2 environment
- *MPICH-GM*: The MPICH-GM environment
- *MPICH-MX*: The MPICH-MX environment
- *MVAPICH*: The MVAPICH (MPI-1) environment
- *MVAPICH2*: The MVAPICH2 (MPI-2) environment
- *OpenMPI*: The Open MPI environment
- *POE*: The POE (IBM MPI) environment
- *PVM*: A Parallel Virtual Machine environment

Other values are possible.

9.3.5.16.2 Version

The optional version of the parallel environment.

9.3.5.16.3 ProcessesPerHost

This optional integer element specifies the number of instances of the executable that the consuming system MUST start on each allocated host. Multiplicity is zero or one. Default value is “1”.

An optional flag "useSlotsPerHost" allows to indicate that the value of "SlotsPerHost" should be used.

9.3.5.16.4 ThreadsPerProcesses

This optional integer element specifies the number of threads per process (i.e., per instance of the executable). There is no default value of this element. Multiplicity is zero or one.

An optional flag "useSlotsPerHost" allows to indicate that the value of "SlotsPerHost" should be used.

9.3.5.16.5 Option

This optional element is a string valued name/value pair allowing to specify additional options to the parallel environment. This allowed names and values depend on the type of environment. They MUST be advertised through the GLUE2 description of the execution service.

9.3.6 DataStaging

Data staging is an optional complex element which describes all the files that should be transferred to the computing element (stage in) and the files that should be transfered from the computing element (stage out). The data movement can be performed by both the client and execution service. Multiplicity is zero or one. There is no default value of this element.

9.3.6.1 ClientDataPush

This optional boolean element indicates that the client wishes to push data to the service under control of the client. If this element is present with the value “true”, the execution service MUST allow client access to the stage-in directory and MUST wait for the end of client data push indicated by a call to NotifyService, as detailed in section 1.3. If the file path in a file transfer request from the client includes hierarchy in respect to stage-in directory all intermediate directories MUST be created automatically.

If the element is not present, or has the value “false”, the execution service is not obliged to provide stage-in directory access for the client and will not wait for a call to NotifyService before moving the activity to the PROCESSING phase (also see Source element below).

9.3.6.2 InputFile

InputFile is an optional complex element which describes a file that should be transferred to the computing element (stage-in) and later made available in session directory. Multiplicity is zero or more.

Note: service MAY choose to provide access to input file in session directory using way other than ordinary copying. For example, an implementation might choose to cache input files and provide them to the job via a link). Hence the job should not expect any access to specified file other than read-only.

9.3.6.2.1 Name

This mandatory string element defines the name of the staging object on the execution service. The name is given as a relative path to the session directory.

Multiplicity is one.

9.3.6.2.2 Source

This optional complex element specifies the source location of the stage in data transfer of a file. Multiplicity is zero or more. In case of multiple sources it is up to the computing service implementation how utilize them. All Sources are treated as binary identical. The ordering of the Source sub-elements is not significant. All files to be transferred via server data pull (transfer initiated by the service) MUST be specified and MUST contain Source element.

If no Source sub-element is provided this means that the file will be staged by the client into the stage-in directory (client data push).

The files to be transferred via client data push (i.e. that will be uploaded on the stage-in directory by the client) MAY be specified (it is not mandatory).

9.3.6.2.2.1 URI

This mandatory URI element defines the source location of the file. It is up to a user to make sure the computing service or the client is able communicate to the given data source.

Multiplicity is one.

9.3.6.2.2.2 Option

This optional key/value pair can be used to convey additional parameters needed for the transfer. Multiplicity is zero or more.

9.3.6.2.2.3 DelegationId

This string attribute specifies the delegationId to be used for the transfer of this file. It is mandatory only if the protocol expressed in the URI element requires it.

There is no default value.

9.3.6.2.3 IsExecutable

This optional boolean element specify whether the executable flag has to be put on the file or not. Multiplicity is zero or one. The default value is false.

9.3.6.3 OutputFile

OutputFile is an optional complex element which describes a file that should be transferred from the computing element (stage-out). Multiplicity is zero or more.

All files for both client data pull (i.e. the ones that will be downloaded from the client from the stage-out directory) and server data push must be declared.

9.3.6.3.1 Name

This mandatory string element defines the name of the staging object on the execution service. Multiplicity is one. The name is given as a relative path to the session directory.

9.3.6.3.2 Target

This optional complex element specifies the target location of the stage out data transfer of a file. Multiplicity is zero or more. There is no default value of this element. The ordering of the Target sub-elements is not significant.

In case of multiple targets the execution service MUST upload the file to all the mandatory targets (see below) or at least one of the targets in case there was no mandatory element defined. In case of both mandatory and non-mandatory Target elements present non-mandatory elements are used only if all mandatory failed. If staging file to any of Target elements failed it is treated as having no Target elements (see below).

To inform the service that the file has to be copied in the stage-out directory for client data pull, there MUST NOT be a Target element. In that case file MUST be provided in stage-out directory under the specified name.

9.3.6.3.2.1 URI

This mandatory URI element defines the target location of the file, and refers to a remote location (server data push). It is up to a user to make sure the computing service or the client is able communicate to the given data target. Multiplicity is one.

9.3.6.3.2.2 Option

This optional key/value pair can be used to convey additional parameters needed for the transfer. Multiplicity is zero or more.

9.3.6.3.2.3 DelegationId

This string attribute specifies the delegationId to be used for the transfer of this file. It is mandatory only if the protocol expressed in the URI element requires it. There is no default value.

9.3.6.3.2.4 Mandatory

This optional boolean attribute defines if the given Target must be used during the stage out data transfer or not. There is no default value of this element.

9.3.6.3.2.5 CreationFlag

This optional flag allows to choose whether existing files should be overwritten or appended to, or whether an error should occur if the file already exists. It can take the values “Overwrite”, “Append”, or “DontOverwrite”. Default is “Overwrite”.

9.3.6.3.2.6 UseIfFailure

This optional element defines if the specified Target element is to be used if job failed in previous states. Default value is false.

9.3.6.3.2.7 UseIfCancel

This optional element defines if the specified Target element is to be used if job was cancelled by client request in previous states. Default value is false.

9.3.6.3.2.8 UseIfSuccess

This optional element defines if the specified Target element is to be used if job reached POSTPROCESSING state without failures. Default value is true.

Note: if all default values of UseIf* elements are applied in case of failure or cancelation of the job, the OutputFile is treated as stageable by client.

10. Security Consideration

Security setup is extremely important in the context of the execution service. So although the details of security setup are out of scope for this document, in this section some considerations are provided. Those are not required for implementing the Execution Service but rather reflect expectations the authors had of typical set-ups.

10.1 Authentication and Authorization

Access to the execution service should be authenticated, and the authenticated user should posses the appropriate rights to execute activities. However, the precise mechanisms are out of scope of this specification.

10.2 Security Bootstraping Information

In order to be able to communicate to the service, a client needs to know the security setup of the service. If information about the service is provided by the service itself, the so-called 'chicken-and-egg' problem of initially contacting a resource is present. We assume that either some generic information service is queried prior to contacting the EMI Execution Service or the ES itself accepts such queries using a well defined minimal security setup.

In any case, a suitable GLUE2 instance with service capabilities that provide security information about the Execution Service with TLS communication setup might be as follows:

<Capability>security.authentication.emi.ssl</Capability>

10.3 Delegation for Data-Staging

Describing all possible delegation mechanisms is out of scope for this specification. This specification only covers one of the possible scenarios (i.e. X.509 proxy delegation) through the adoption of a dedicated delegation portType (see section **Fehler! Textmarke nicht definiert.**) and a way to use the provided delegation tokens for performing data-stagings as requested in the activity description document. Common examples are GridFTP transfers and SRM access as part of a computational activity.

Furthermore, the concrete delegation interface and mechanism covered in this specification is intended as a temporary solution, since other groups within EMI are in the progress of defining an EMI-wide mechanism for delegation, which eventually will be used also by the Execution Service.

11. Editor Information

Morris Riedel
Juelich Supercomputing Centre, Germany
Email: m.riedel@fz-juelich.de

12. Authors

Bernd Schuller
Juelich Supercomputing Centre, Germany
Email: b.schuller@fz-juelich.de

Balazs Konya
University of Lund, Sweden
Email: balazs.konya@hep.lu.se

Oxana Smirnova
University of Lund, Sweden
Email: urbah@lal.in2p3.fr

Alexandr Konstantinov
University of Oslo, Norway
Email: aleksandr.konstantinov@fys.uio.no

Martin Skou Andersen
University of Copenhagen, Denmark
Email: skou@nbi.ku.dk

Morris Riedel
Juelich Supercomputing Centre, Germany
EMail: m.riedel@fz-juelich.de

Shahbaz Memon
Juelich Supercomputing Centre, Germany
Email: m.memon@fz-juelich.de

Shiraz Memon
Juelich Supercomputing Centre, Germany
Email: a.memon@fz-juelich.de

Lisa Zangrando
National Institute of Nuclear Physics, Italy
Email: lisa.zangrando@pd.infn.it

Massimo Sgaravatto
National Institute of Nuclear Physics, Italy
Email: massimo.sgaravatto@pd.infn.it

Eric Frizziero
National Institute of Nuclear Physics, Italy
Email: eric.frizziero@pd.infn.it

13. Acknowledgments

We are grateful to numerous colleagues for discussions on the topics covered in this document, and to the people who provided comments on the public drafts.

14. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

15. Full Copyright Notice

16. References

[1] Strawman of the AGU Execution Service: Functionality Description, http://forge.gridforum.org/sf/go/doc15736

[2] Job Submission Description Language (JSDL) Specification, Version 1.0 www.gridforum.org/documents/GFD.136.pdf

[3] GLUE Specification v. 2.0, www.gridforum.org/documents/GFD.147.pdf

[4] JSDL SPMD Application Extension, Version 1.0, www.gridforum.org/documents/GFD.115.pdf