Editors:
Morris Riedel, JSC/FZJ
Aleksandr Konstantinov, UoLund

31 March 2013

OGF – Production Grid Infrastructure

European Middleware Initiative Execution Service

Version 2.0

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 1.16.

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.

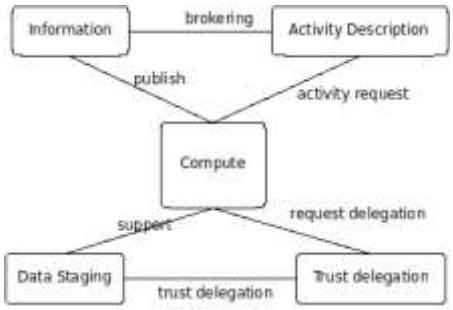

Figure 1: Relationship between execution service elements

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 61 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 five main modules. These modules offer different functionalities realized as independent port-types, and can be grouped and offered via independent services, usually on the same machine, eventually running separately on different machines.

Each of them implements a set of operations. The following describes each module's purpose and the operations corresponding to each port-type.

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

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

It must be stressed that the ResourceInfo port-type refers **only to information related** to the Computing Element and it does not contain information about activities. Activities information can be retrieved using the ActivityInfo port-type.

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:

1. **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.
2. **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 55). 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:

1. **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.

2. **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.

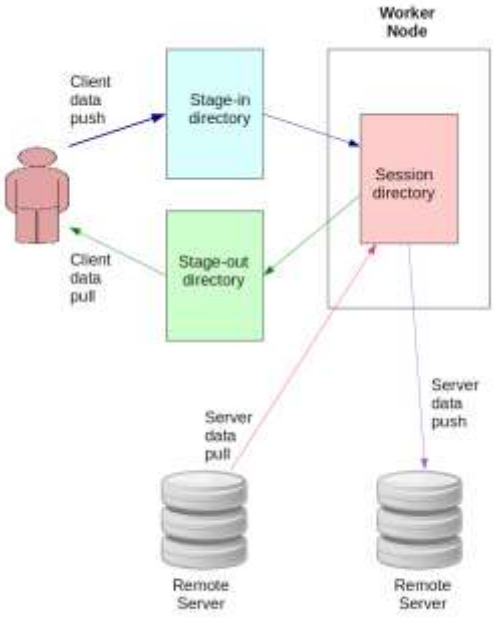

Figure 2: Data Staging directories and actors

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 **terminal** state with one of ***-failure** attributes set 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 data object whether stage-out should be performed, see the UseIfFailure attribute (section 57).
- If there is a failure during the stage out of a file, activity will go into the **terminal** with **postprocessing-failure** attribute. 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 activity is cancelled during user job execution, the ES can optionally (configurable in the activity description document, see attribute UseIfCancel, section 57) 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 those tokens 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 considered:

- X.509 proxy credentials (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 solutions to “transfer” proxies are in use:
- 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 servers. 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 multiple activities where each activity is described by an activity description document. The service MUST perform a certain amount of validation on each activity. 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 unique identifier, the activityID assigned by the service. The activityID should not be assumed by client applications to contain any meaningful information like service address, i.e. a activity is NOT directly globally 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 After retrieving the URL for stage-in directory from service, the client is able to push the data to the execution service as long as service advertises **client-stagein-possible** attribute.

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

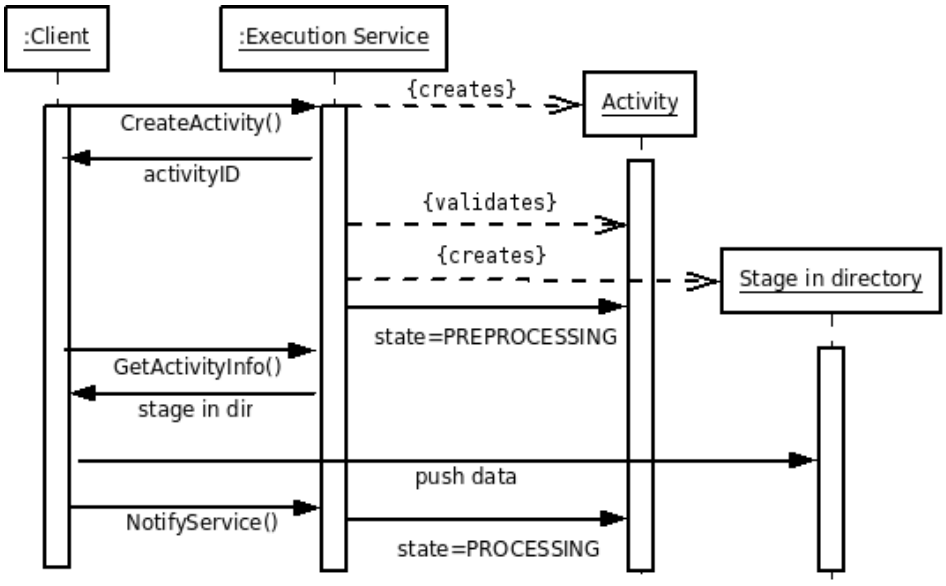

Figure 3: Activity creation, validation and client data push

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 an activityID assigned to the activity and uniquely identifying it inside the service
- It MUST return the ActivityManagement Endpoint URL that MUST be contacted to manage the activity.
- It MUST return the ResourceInfo Endpoint URL that MAY be contacted to retrieve CE information.
- 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 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.
    - For those activities which can't be processes due internal access control decisions specific to those activities the service MUST return a response containing a message error AccessControlFault.
    - In all other cases where activity could not be created error message InternalBaseFault is used.

In case the number of submitted activities is too large, instead of CreateActivity response the service MUST return SOAP fault VectorLimitExceededFault, which contains the maximal allowed number of activity descriptions per single CreateActivity request. In this case the service MUST NOT attempt to create any activities.

Faults

All faults returned by all EMI ES operations are derived from InternalBaseFault. This base fault provides numerical FailureCode of fault, short textual Message and longer Description. All these values are implementation specific and are not defined in this document. The base fault also provides Timestamp of when failure happened. Other kinds of faults may also provide additional information.

AccessControlFault – client is not allowed to perform CreateActivity operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

Activity Description Document Validation

The service takes an 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 **validation-failure** secondary state of **terminal** state. The list of mandatory validation steps is as follows:

0. 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 ResourceInfo port-type offers two operation to obtain information about the EMI-ES resources. The GetResourceInfo operation returns all the information wrapped as a single “resource document” while the QueryResourceInfo operation enables server-side queries on the same document.

The resource information served by this port-type refers only to information related to the Computing Element, represented via a “resource document” composed of GLUE2 entities and attributes. It MUST contain information related to all EMI-ES port-types regardless whether these are hosted by a single service or deployed in a distributed fashion. The document MAY contain other non EMI-ES endpoints and resource information in case these are hosted by the same services running EMI-ES. The documents MUST NOT 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 model ComputingService entity and all its related entities.. Activity records are not part of the resource information.

The operation is used to obtain information about EMI-ES Endpoints and their optional distribution across various services and the resources behind those, as a single document composed of GLUE2 XML elements that abides the GLUE2 computing service model.

The resource document is described in Section 8.1

Data-Staging Implications

The resource document MUST contain data staging capabilities expressed as GLUE2 Endpoint Capability. Actual Capability attribute values are in Section 8.1.

State Model Implications

None

Request

None

Response

- The response MUST include the resource document as described in Section 8.1.
- In case of no possibility to return a complete resource document, an appropriate fault MUST be returned.

Faults

InternalResourceInfoFault – service failed to generate/retrieve requested information
ResourceInfoNotFoundFault – service has no information to report and this is not information generation fault
AccessControlFault – client is not allowed to perform GetResorceInfo operation
InternalBaseFault – any other failure preventing service from performing request

A conceptual sketch of the response document is shown below:

```
<GetResourceInfoResponse>
  <Services>
    <ComputingService CE>
      <Endpoint1 ActivityCreation>
        <URL>https://somehost.somedomain:8000/ActivityCreation</URL>
      </Endpoint1>
      <Endpoint2 ResourceInfo>
        <URL>https://somehost.somedomain:8000/ResourceInfo</URL>
      </Endpoint2>
      <Endpoint3 ActivityManagement>
```

```
        <URL>https://somehost.somedomain:8000/ActivityManagement</URL>
      </Endpoint3>
      <Endpoint4 Delegation>
        <URL>https://somehost.somedomain:8000/Delegation</URL>
      </Endpoint4>
    </ComputingService>
    <Service JobInfo>
      <Endpoint ActivityInfo >
        <URL>https://otherhost.otherdomain:8111/ActivityInfo</URL>
      </Endpoint>
    </Service>
  </Services>
</GetResourceInfoResponse>
```

Full example documents are presented in section 13.4.

3.2 QueryResourceInfo Operation

Functionality

This operation provides flexible access to query the full resource document (see Section 8.1). 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 a GLUE2 Capability element in the resource document.

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). The XPath queries must be processed by service as if GLUE2 document has no namespace defined. The reason is that GLUE2 document can contain only elements of one namespace and stripping it allows for simpler XPath expressions and no need to handle namespace definitions.

Response

The response includes the following information:

- Results of the query.
- In case of a failed query, an appropriate FAULT is returned

Faults

NotSupportedQueryDialectFault – the requested query dialect is not supported by service

NotValidQueryStatementFault – the query expression is not recognized as valid query for specified query dialect
InternalResourceInfoFault – service failed to generate/retrieve requested information

ResourceInfoNotFoundFault – service has no information to report and this is not information generation fault
AccessControlFault – client is not allowed to perform QueryResourceInfo operation

InternalBaseFault – any other failure preventing service from performing request

4. Interface: ActivityManagement Port-Type

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

4.1 PauseActivity

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 activity **client-paused** attribute. Note: it is highly likely that due to short lifetime of this state service will choose to stop activity 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 activity 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 activity the **client-paused** attribute.

postprocessing – any de-provisioning and data staging activities are stopped. Once processing is paused the service assigns the activity the **client-paused** attribute.

terminal – not applicable.

Request

vector of activityID elements

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, an appropriate error element is returned
  - ActivityNotFoundFault – specified activityID does not match any known activity
  - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
  - OperationNotAllowedFault - operation can't be performed because activity is not in suitable

state
- AccessControlFault - operation can't be performed due to internal access control decisions specific to this activity
- InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

AccessControlFault – client is not allowed to perform PauseActivity operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

4.2 ResumeActivity

Functionality

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 activity was paused or not.

Request

vector of activityID elements

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, an appropriate error element is returned
  - ActivityNotFoundFault – specified activityID does not match any known activity
  - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
  - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state
  - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
  - InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

AccessControlFault – client is not allowed to perform ResumeActivity operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

4.3 NotifyService

Functionality
The 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:

- The activityID
- A string, which specifies what the client wants to notify service about. This string allows for one of the possible values:
- **client-datapull-done**: this is used to notify the service that the client has completed the client data pull action
- **client-datapush-done**: this is used to notify the service that the client has completed the client data push action

Response:

The response of the NotifyService operation returns a vector of values; one per input element with each value 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 error element is returned
  - ActivityNotFoundFault – specified activityID does not match any known activity
  - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
  - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state (like **client-datapush-done** notification while activity is in **postprocessing** state)
  - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
  - InternalNotificationFault - any other failure preventing service to perform operation on specific activity and related to notification functionality
  - InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

- VectorLimitExceededFault – request contains vector with too many elements (see CreateActivity)
- AccessControlFault – client is not allowed to perform NotifyService operation
- InternalNotificationFault - any other failure preventing service from processing all activities and related to notification functionality
- InternalBaseFault – any other failure preventing service from processing all activities

4.4 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 **terminal** with ***-cancel** attribute 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 the user 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 57.

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 **terminal** state with one the ***-cancel** attributes, 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, an appropriate error element is returned
    - ActivityNotFoundFault – specified activityID does not match any known activity
    - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
    - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state
    - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
    - InternalBaseFault – any other error preventing service to perform operation on specific activity
-

Faults

AccessControlFault – client is not allowed to perform CanelActivity operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

4.5 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 return error element OperationNotAllowedFault

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, an appropriate error element is returned
    - ActivityNotFoundFault – specified activityID does not match any known activity
    - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
    - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state (one of terminal states)
    - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
    - InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

AccessControlFault – client is not allowed to perform WipeActivity operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

4.6 RestartActivity Operation (Optional)

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). Activity is still considered as same activity and its activityID is unchanged.

Data-Staging Implications

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

State Model Implications

The RestartActivity 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 activityID elements.

Response

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

- For those activities which can be restarted, the execution service should return time estimation in which the restart will be done;
  - EstimatedTime: estimated time restart the activity , special values: 0 means already restarted (i.e. immediately), undefined if the service is not able to perform an estimation
- For those activities which cannot be restarted, an appropriate error element is returned
  - ActivityNotFoundFault – specified activityID does not match any known activity
  - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
  - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state
  - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
  - InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

OperationNotPossibleFault – if service does not support this operation
AccessControlFault – client is not allowed to perform RestartActivity operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

4.7 GetActivityStatus Operation

Functionality

This operation provides the state information according to the state model (see Section **Error! Bookmark not defined.**) 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 a vector of activityID elements

Response

The response includes the following information:

- For those activities for which state can be obtained the following information is provided for each activity:
  - activity identifier
  - the corresponding state (see section 7.1)
  - the corresponding state attributes (optional) (see section 7.1)
  - a timestamp of last state change
  - an optional description (or message)
- For those activities where the state cannot be provided (for example, because the activity identifier does not exist), an appropriate error structure is returned
  - ActivityNotFoundFault – specified activityID does not match any known activity
  - UnableToRetrieveStatusFault – although activity does exist retrieving its status failed
    - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
    - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state
  - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
  - InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

AccessControlFault – client is not allowed to perform GetActivityStatus operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

A conceptual sketch of an example response is as follows:

```
<GetActivityStatusResponse>
  <ActivityStatusItem>
    <ActivityID>556a7fbc-28ff-414e-bcee-6d1116c7d399</ActivityID>
    <ActivityStatus>
      <State>accepted</State>
      <StateAttribute>validating</StateAttribute>
      <StateAttribute>client-stagein-possible</StateAttribute>
      <StateAttribute>client-stageout-possible</StateAttribute>
      <Timestamp>2012-07-11T10:23:40.559+02:00</Timestamp>
      <Description>Running through validation</Description>
    </ActivityStatus>
  </ActivityStatusItem>
  <ActivityStatusItem>
    <ActivityID>20503220-d0bb-11e1-9c3f-001e331f6e47</ActivityID>
    <ActivityStatus>
      <State>terminal</State>
      <Timestamp>2012-07-11T10:23:40.559+02:00</Timestamp>
      <Description>Failure: not restartable</Description>
    </ActivityStatus>
  </ActivityStatusItem>
</GetActivityStatusResponse>
```

4.8 GetActivityInfo Operation

Functionality

This vector operation provides the activity information according to the GLUE2 activity model. The set of activities for which the information is requested MUST be explicitly provided in the request and the operation will return either a full EMI-ES activity document or a selection of EMI-ES activity elements.

The latter can be achieved additionally specifying a set of requested elements.

The EMI-ES activity document uses attributes from the GLUE2 ComputingActivity entity and it extends it with EMI-ES attributes for capturing key concepts such as data staging..

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 as described in Section 8.2. 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 GetActivityInfo response MUST publish the activity state according to the EMI ES state model and MAY publish additional state model values as well. EMI-ES states and state attributes MUST be published following a format defined in Section 8.2.

Request

The request includes the following information:

- A vector of activityID elements
- A vector of EMI-ES activity attributes. In case this parameter is not specified then the full activity record MUST be returned.

1. **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 either a vector of requested EMI-ES attributes or the full EMI-ES activity document if no attribute is 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 attribute was requested), an appropriate error structure is returned
  - ActivityNotFoundFault – specified activityID does not match any known activity
  - UnableToRetrieveStatusFault – although activity does exist retrieving its information failed
    - OperationNotPossibleFault – operation can't be performed because properties of activity somehow prevent it
    - OperationNotAllowedFault - operation can't be performed because activity is not in suitable state
  - AccessControlFault - operation can't be performed due internal access control decisions specific to this activity
  - InternalBaseFault – any other error preventing service to perform operation on specific activity

Faults

2. UnknownAttributeFault – at least one of specified activity attributes is not recognized
   AccessControlFault – client is not allowed to perform GetActivityInfo operation.

VectorLimitExceededFault – number of elements in request exceeds allowed one.

InternalBaseFault – any other failure preventing service from processing all activities.

A conceptual sketch of the two possible activity responses is shown below:

1) Full activity documents are returned. Note that the mandatory elements are always present in the full document response.

```
<GetActivityInfoResponse>
  <ActivityInfoItem>
    <ActivityID>15f4a518</ActivityID>
    <ActivityInfoDocument>
      <ID>15f4a518<ID>
```

```
            <IDFromEndpoint>https://cream-10.pd.infn.it:8443/Activity/15f4a518</IDFromEndpoint>
            <State>emies:accepted</State>
            <State>emiesattr:validating</State>
            <Owner>CONFIDENTIAL</Owner>

            …

          </ActivityInfoDocument>
        <ActivityInfoItem>
        <ActivityInfoItem>
          <ActivityID>798ac354</ActivityID>
          <ActivityInfoDocument>
            <ID>798ac354<ID>
            <IDFromEndpoint>https://cream-10.pd.infn.it:8443/Activity?798ac354</IDFromEndpoint>
            <State>emies:preprocessing</State>
            <State>emiesattr:server-stagein</State>
            <Owner>/DC=eu/DC=KnowARC/O=Lund University/CN=demo1</Owner>

             …

          </ActivityInfoDocument>
        <ActivityInfoItem>
      <GetActivityInfoResponse>
```

2) Filtered documents are returned; in this case the document contains only the requested elements, namely Error and StageOutDirectory:

```
<GetActivityInfoResponse>
  <ActivityInfoItem>
    <ActivityID>15f4a518</ActivityID>
    <ActivityInfoDocument>
      <Error>Error code 476259</Error>
      <StageOutDirectory>gsiftp://localhost:2811/FILESPACE/15f4a518</StageOutDirectory>
    </ActivityInfoDocument>
  </ActivityInfoItem>
  <ActivityInfoItem>
    <ActivityID>798ac354</ActivityID>
    <ActivityInfoDocument>
      <Error>Error code 421223</Error>
      <StageOutDirectory>gsiftp://localhost:2811/FILESPACE/798ac354/</StageOutDirectory>
    </ActivityInfoDocument>
  </ActivityInfoItem>
<GetActivityInfoResponse>
```

- Complete XML documents examples are provided in section 13.5.

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 user identity is not an explicit parameter of the ListActivities operation, but is considered to be prerequisite, already obtained through the authentication process.

The association between the user identity and his/her activities is managed by the server.

The operation returns just a list of the activityIDs while detailed information about those Activities can be obtained by invoking the getActivityInfo. Moreover the ListActivities operation 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 ListActivities operation accepts as input the following NOT mandatory parameters:

- FromDate, ToDate: the activity creation time window
- ActivityStatus elements: the list of possible statuses, that is, each item of the list is a state with all its attributes.
- 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.

A conceptual sketch of the ListActivities request is shown below:

```
<ListActivitiesRequest>
  <FromDate />
  <ToDate />
  <Limit />
  <ActivityStatusList>
    <ActivityStatus>
      <State>accepted</State>
      <StateAttribute>validating</StateAttribute>
      <StateAttribute>client-stagein-possible</StateAttribute>
      <StateAttribute>client-stageout-possible</StateAttribute>
    </ActivityStatus>
    <ActivityStatus>
      <State>terminal</State>
      <StateAttribute>validation-failure</StateAttribute>
      <StateAttribute>app-failure</StateAttribute>
    </ActivityStatus>
    <ActivityStatus>
      <State>processing-queued</State>
      <StateAttribute>server-paused</StateAttribute>
      <StateAttribute>client-paused</StateAttribute>
    </ActivityStatus>
  </ActivityStatusList>
</ListActivitiesRequest>
```

Response

The response of the ListActivity operation returns a list of ActivityIds or an appropriate fault.

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

Faults

InvalidParameterFault - some input parameter is illegal (for example FromDate is older than ToDate)

AccessControlFault – client is not allowed to perform ListActivities operation.

InternalBaseFault – any other failure preventing service from processing all activities.

5.2 GetActivityStatus Operation

This operation is as described in section 4.7 above.

5.3 GetActivityInfo Operation

This operation is as described in section 4.8 above.

6. Interface: Delegation Port-Type

Delegation is a critical ingredient for the Execution Service, since data staging often must be performed on behalf of clients. This 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. Since delegation is critical for other services as well (such as data management services), a common solution for all the EMI services is desirable. Currently EMI ES adopts not yet released version 2.1 of GridSite delegation service. Version 2.1 is modified version 2.0 adapted to document/literal WSDL style. Below GridSite delegation operations are described as applicable to EMI Execution service.

6.1 getVersion Operation

Functionality

The *getVersion* operation provides version of service implementing Delegation capability. This version is specific to implementing service and can be any.

Data-Staging Implications

None.

State model Implications

None.

- **Request**

The request is empty.

- **Response**
- The response contains *getVersionReturn* string element with version information.

Faults

- DelegationException – generic error

6.2 getInterfaceVersion Operation

Functionality

The *getInterfaceVersion* operation provides version of this interface. Its value must be 2.1.

Data-Staging Implications

None.

State model Implications

None.

Request

The request is empty.

Response
The response contains *getInterfaceVersionReturn* string element containing “2.1” value.

Faults

- DelegationException – generic error

6.3 GetServiceMetaData Operation

Functionality

The *getServiceMetadata* operation provides access to meta-data describing this service. Meta-data is organised in key-value pairs. Service is free to provide any meta-data. For EMI ES no meta-data is required.

Data-Staging Implications

None.

State model Implications

None.

Request

The request contains *key* of requested meta-data.

Response

The response contains corresponding value of requested meta-data.

Faults

DelegationException – generic error. Also returned if meta-data for requested key does not exist.

6.4 GetProxyReq Operation

Functionality

The *getProxyReq* 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 *PutProxy* operation is used to complete the delegation procedure.

Data-Staging Implications

The delegationID supplied in request 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 data-staging 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. The delegationID MAY be (re-)used in multiple CreateActivity operation invocations.

Request

- delegationID: the ID to be assigned to be used in *putProxy* and assigned to stored delegation. GridSite allows for empty delegationID which is then replaced by hash of certificate subject and VOMS(7) attributes. But because neither hashing algorithm nor way to combine VOMS attributes are defined delegationID MUST be not empty in EMI ES implementation.

Response

The response includes getProxyReqReturn with an X.509 certificate signing request in PEM format.

Faults

DelegationException – generic error.

6.5 getNewProxyReq Operation

Functionality

The *getProxyReq* 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 *PutProxy* operation is used to complete the delegation procedure.

Data-Staging Implications

The delegationID provided in 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 data-staging 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. The delegationID MAY be (re-)used in multiple CreateActivity operation invocations.

Request

Request is empty.

Response
The response includes proxyRequest with an X.509 certificate signing request in PEM format and delegationID assigned by service.

Faults

DelegationException – generic error. Also returned if credentials with specified delegationID already exists.

6.6 renewProxyReq Operation

Functionality

The *renewProxyReq* operation requests to replace already existing delegation with new one. The server answers with a certificate signing request which includes the public key for the renewed delegated credentials. The *PutProxy* operation is used to complete the delegation procedure.

Data-Staging Implications

The credentails renewal operation may happen after CreateActivity. In this case after PutProxy is performed renewed credentials are to be passed to already existing activity. If activity is in data-staging state it is up to implementation how soon new credentials become active.

State model Implications

The two-step delegation process MUST be performed before credentials renewal operation is complete.

Request

- delegationID: the ID of delegation to be replaced.

Response

The response includes renewProxyReqReturn with an X.509 certificate signing request in PEM format.

Faults

DelegationException – generic error. Also returned if delegationID does not match any of client's delegated credentials.

6.7 putProxy Operation

Functionality

The p*utProxy* operation completes the delegation procedure by sending the signed proxy certificate along with the d*elegationID* to the server. The signed certificate is based on the certificate signing request previously retrieved together with the d*elegationID* via an *getProxyReq, getNewProxyRequest* or *renewProxyReq* operation invocation.

Data-Staging Implications

The delegated credential MAY be used by the Execution Service to carry out data-staging 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:

- the delegationID identifying delegation session as assigned in getProxyReq/*renewProxyReq* or obtained by getNewProxyRequest
- the proxy element which contains RFC 3280 style proxy certificate, signed by the client

Response

The response includes the SUCCESS string in case of successful putProxy operation.

Faults

DelegationException – generic error. This is also returned if session for delegationID was not started using getProxyReq/getNewProxyRequest.

6.8 getTerminationTime Operation

Functionality

The *getTerminationTime* operation returns expiration time of specified delegation.

Data-Staging Implications

None.

State model implications

The delegation is two-step process. Hence *getTerminationTime* can succeed only after second step delegation process is complete.

Request
The request includes the delegationID identifying delegated credentials for which information is requested.

Response
The response includes the getTerminationTimeReturn element which contains expiration time of delegated credentials.

Faults

DelegationException – generic error. This is also returned of no delegated credentials found matching delegationID for requesting client.

6.9 destroy Operation

Functionality

The *destroy* operation erases delegated credentials or open delegated session from service.

Data-Staging Implications

None.

State model implications

None.

Request
The request includes the delegationID identifying delegated credentials or delegated session started by getProxyReq/getNewProxyRequest/renewProxyReq operations.

Response
Successful response is empty.

Faults

DelegationException – generic error. This is also returned if no delegated credentials or session found matching delegationID for requesting client.

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. A state and all its assigned attributes together defines an activity **status**.

The **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 activity is doing stage-out, releasing resources (de-provisioning). This phase is represented by state **postrpocessing**.
- 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 by 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 **state attributes** are defined:

- **validating** informs that service is performing validation of activity request.
- **server-paused** means server stopped activity 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. The activity 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 activity 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 user 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 the activity description – the user job. 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 [**preprocessing-cancel**, **processing-cancel**, **postprocessing-cancel**] inform that activity was cancelled on client request. First part of name refers to phase of activity when cancellation request was executed.
- ***-failure** attributes [**validation-failure**, **app-failure**, **preprocessing-failure**, **processing-failure**, **postprocessing-failure**] inform that activity processing failed. First part of name refers to phase of activity where failure was detected. Additionally **validation-failure** refers to job failed due to failure to validate activity request. **app-failure** means failure of specified payload.
- **expired** indicates that the activity was cancelled because its expiration time has been exceeded. The expiration time is optionally set in the activity description, see section **Error! Bookmark not defined.**.

The following table 1 shows the applicability of attributes to states.

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

Initial state	**accepted**	**preprocessing**	**processing-accepting**	**processing-queued**	**processing-running**	**postprocessing**	**terminal**
Allowed transitions	**preprocessing**	**processing-accepting**	**processing-queued**	**processing-running**	**processing-queued**	**terminal**	
	terminal	**postprocessing**	**processing-running**	**postprocessing**	**postprocessing**		
		terminal	**postprocessing**	**terminal**	**terminal**		
			terminal				

Table 2. Allowed state transitions

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. In this section a normative definition of the information to be published through the information port-types will be given based on GLUE2 model.

It is important to note that notion of Activity as defined in this document corresponds to a job in GLUE2 terms, that is, information related to an EMI-ES Activity can be published as a GLUE2 ComputingActivity element.

8.1 Resource information: the resource document

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 model [3].

The resource information is contained in a "resource document" composed of GLUE2 XML elements that abides the GLUE2 computing service model. The main entity of the model being endorsed is the ComputingService, which further aggregates the ComputingManager, ComputingEndpoint, ComputingShare, ExecutionEnvironment, and ApplicationEnvironment.

The resource document MUST contain information related to all EMI-ES port-types represented as GLUE2 Endpoints or ComputingEndpoints. These Endpoints MAY be deployed on separate services: in this case, the document MUST contain information about these services as well.

The resource document MAY contain other non EMI-ES endpoints and resource information in case these are hosted by the same services running EMI-ES.

The documents MUST NOT include information about activities (not even the list of activity Ids).

8.1.1 Structure of the resource document

The mandated structure follows the hierarchical XML rendering of the GLUE2 model.

The structure is presented through two examples based on different deployment scenarios. In the first scenario (A), all the EMI-ES port-types are implemented in the same service. In the second case (B), EMI-ES port-types are distributed over multiple services. The two examples should be read as normative definitions of the resource document. The first occurrence of an element identifies the mandatory and optional attributes of an entity. Items must be rendered as XML elements, and they are presented here as simple strings for ease of reading.
There is no limitation on the number of endpoints per port-type. For example, there can be more than one ActivityCreation endpoints, provided that their GLUE2 ID is different.

Attribute values and their semantics are given in Section 8.1.2.

8.1.1.1 Response document for Scenario A

```
<GetResourceInfoResponse>
 <Services>
  <ComputingService>
    <!-- Mandatory -->
    ID
    Type
    HealthState
    Capability
    QualityLevel

    <!-- Optional -->
    CreationTime
    Validity
    Name
    OtherInfo
    StatusInfo
    Complexity
    TotalJobs
    RunningJobs
    WaitingJobs
    StagingJobs
    SuspendedJobs
    PreLRMSWaitingJobs
    Associations

   <ComputingEndpoint>
```

```
        <!-- Mandatory -->
        InterfaceName = org.ogf.glue.emies.activitycreation
        Capability
        URL
        ID
        ImplementationName
        ImplementationVersion
        QualityLevel
        HealthState
        ServingState
        Staging
        JobDescription

        <!-- Optional -->
        CreationTime
        Validity
        Name
        OtherInfo
        Technology
        InterfaceVersion
        InterfaceExtension
        WSDL
        SupportedProfile
        Semantics
        Implementor
        HealthStateInfo
        StartTime
        IssuerCA
        TrustedCA
        DowntimeAnnounce
        DowntimeStart
        DowntimeEnd
        DowntimeInfo
        TotalJobs
        RunningJobs
        WaitingJobs
        StagingJobs
        SuspendedJobs
        PreLRMSWaitingJobs
        Associations
        <AccessPolicy></AccessPolicy>

    </ComputingEndpoint>
    <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.activitymanagement
        Capability
        ...
    </ComputingEndpoint>
    <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.activityinfo
        Capability
        ...
    </ComputingEndpoint>
    <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.resourceinfo
        Capability
        ...
    </ComputingEndpoint>
    <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.activityinfo
        Capability
        ...
    </ComputingEndpoint>
```

```
    <ComputingManager></ComputingManager>
    <ComputingShare></ComputingShare>
    <Location></Location>
    <Contacts></Contacts>
    <!-- non EMI-ES endpoints -->
    <ComputingEndpoint>
        InterfaceName = org.nordugrid.gridftpjob
        Capability
        ...
    </ComputingEndpoint>
    <ComputingEndpoint>
        InterfaceName = org.nordugrid.ldapng
        Capability
        ...
    </ComputingEndpoint>
  </ComputingService>
 <Services>
</GetResourceInfoResponse>
```

8.1.1.2 Response document for Scenario B

```
<GetResourceInfoResponse>
 <Services>
  <ComputingService>
     ID = bd9aa2ca-ccf3-11e1-be88-001e331f6e47
     Type = org.distributed.CE
     Name = jobmanager service
     OtherInfo = This service hosts two EMI-ES port-types
     OtherInfo = and one non EMI-ES Endpoint

    <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.activitycreation
        Capability
        ...
    </ComputingEndpoint>
    <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.activitymanagement
        Capability
        ...
    </ComputingEndpoint>
    <ComputingManager></ComputingManager>
    <ComputingShare></ComputingShare>
    <Location></Location>
    <Contacts></Contacts>
    <!-- non EMI-ES endpoints -->
    <ComputingEndpoint>
        InterfaceName = org.nordugrid.gridftpjob
        Capability
        ...
    </ComputingEndpoint>
  </ComputingService>

  <ComputingService>
     ID = 17e5f658-ccf4-11e1-970e-001e331f6e47
     Type = org.distributed.CE
     Name = jobinfo service
     OtherInfo = This service hosts one EMI-ES port-type
     <ComputingEndpoint>
        InterfaceName = org.ogf.glue.emies.activityinfo
        Capability
        ...
```

```
      </ComputingEndpoint>
   </ComputingService>

   <Service>
      ID = 9ddec5b4-ccf4-11e1-817a-001e331f6e47
      Type = org.distributed.CE
      Name = resourceinfo service
      OtherInfo = This service hosts one EMI-ES port-type
      OtherInfo = and one non EMI-ES Endpoint
      <Endpoint>
         InterfaceName = org.ogf.glue.emies.resourceinfo
         Capability
         ...
      </Endpoint>
      <!-- non EMI-ES endpoints -->
      <Endpoint>
         InterfaceName = org.nordugrid.ldapng
         Capability
         ...
      </Endpoint>
   </Service>

   <Service>
      ID = cba592a2-ccf4-11e1-a51f-001e331f6e47
      Type = org.distributed.CE
      Name = delegation service
      OtherInfo = This service hosts one EMI-ES port-type
      <Endpoint>
         InterfaceName = org.ogf.glue.emies.delegation
         Capability
         ...
      </Endpoint>
   </Service>
  <Services>
</GetResourceInfoResponse>
```

8.1.2 GLUE2 resource document attribute values

In this section a detailed explanation of the GLUE2 attribute values is given, The mandatory/optional nature of the element is also repeated for ease of reading. Section 13.4 provides the detailed schema-level example of the resource information model used within the ResourceInfo port type.

8.1.2.1 ID

Each GLUE2 ID element of the resource document MUST have a universally unique valid URI as a value; The IDs must not be interpreted by the user or the system as having any meaning other than identifiers. In particular, there is no predefined relationship between an ID and a network endpoint or activityID. IDs MUST be persistent, as long as the service is considered to be the same. For example, a service restart or configuration changes should not result in a new ID unless it is deliberately requested by operational needs. It is up to implementation how to generate suitable IDs.

8.1.2.2 Service Type

The value of this element should be the Type of the service hosting the EMI-ES Endpoints. These values SHOULD be taken from the official GLUE2 enumeration list of ServiceType_t [5].

8.1.2.3 Service Capability

The value of this mandatory element MUST be the union of the capabilities of the endpoints hosted by the service.

8.1.2.4 ComputingService TotalJobs, RunningJobs, WaitingJobs, StagingJobs, SuspendedJobs and PreLRMSWaitingJobs

These element values, if published, SHOULD reflect all the activities managed by the service through any of

its endpoints, including non EMI-ES ones.

8.1.2.5 Endpoint InterfaceName

For EMI-ES endpoints, the value of this element MUST be the InterfaceName of the port-type offered by the endpoint. The mapping between port type and InterfaceName is defined in the table 3.

If the service contains non EMI-ES endpoints, then their InterfaceName must be published according to the official GLUE2 Enumeration list for InterfaceName_t.

EMI-ES Port Type	*GLUE2 InterfaceName_t*	*GLUE2 Capability_t*
ActivityCreation	org.ogf.glue.emies.activitycreation	executionmanagement.jobcreation* executionmanagement.jobdescription* data.access.sessiondir.<proto> data.access.stageindir.<proto> data.access.stageoutdir.<proto> data.transfer.cepull.<proto> data.transfer.cepush.<proto>
ActivityManagement	org.ogf.glue.emies.activitymanagement	executionmanagement.jobmanagement* information.lookup.job* data.access.sessiondir.<proto> data.access.stageindir.<proto> data.access.stageoutdir.<proto> data.transfer.cepull.<proto> data.transfer.cepush.<proto>
ResourceInfo	org.ogf.glue.emies.resourceinfo	information.discovery.resource* information.query.xpath1* information.query.<language>
ActivityInfo	org.ogf.glue.emies.activityinfo	information.discovery.job* information.lookup.job*
Delegation	org.ogf.glue.emies.delegation	security.delegation*

Table 3: Port-types and GLUE2 related open enumerations. A Capability with * means that the capability is mandatory: must be present in all instances of that endpoint. Capabilities and InterfaceNames in bold are newly defined in this document.

8.1.2.6 Endpoint InterfaceVersion

The value of this optional element should contain the version of the EMI-ES Specification document (that is, this document or other versions of this document) to which the implementation refers to.

8.1.2.7 Endpoint WSDL

The value of this optional element should point to the URL of the WSDL document describing EMI-ES port-type.

8.1.2.8 Endpoint Semantics

The value of this optional element MAY be a URL to a location where this EMI-ES Specification document is stored.

8.1.2.9 Endpoint Implementor

The value of this optional element is the name of the organization developing and maintaining the EMI-ES implementation.
Examples:

```
<Implementor>NorduGrid</Implementor>
<Implementor>gLite</Implementor>
```

8.1.2.10 Endpoint Capability

The values for this multivalued mandatory element MUST be taken as shown in table 3 for all the EMI-ES endpoints.
Please note the mandatory and non-mandatory values in the table.
The implementations should take care of publishing the values only if they are relevant, for example, a **data.transfer.cepull.<proto>** should be published if and only if the server is capable of doing server pull transfers. For the detailed definition of all capabilities refer to table 4.

Capability_t	*Description*
data.access.sessiondir.<protocol>	Capacity of providing access to the directory (eventually on a worker node) where the job is executed by means of <protocol> <protocol> is one of: ftp, https, gridftp
data.access.stageindir.<protocol>	Capacity of offering clients a place where to upload data by means of <protocol> <protocol> is one of: ftp, https, gridftp
data.access.stageoutdir.<protocol>	Capacity of offering clients a place from where to download output data by means of <protocol> <protocol> is one of: ftp, https, gridftp
data.transfer.cepull.<protocol>	Capacity of the Computing Element to fetch files from remote network location. <protocol> is one of: ftp, https, gridftp, srm
data.transfer.cepush.<protocol>	Capacity of the Computing Element to upload files to remote network location. <protocol> is one of: ftp, https, gridftp, srm
executionmanagement.jobcreation	Capacity of creating an activity or a set of activities
executionmanagement.jobmanagement	Capacity of managing an activity or a set of activities on a Computing Element
information.discovery.job	Capacity of locating activities, possibly satisfying a set of requirements
information.lookup.job	Capacity of providing information about an activity or a set of activities
information.query.<language>	Capacity of answering information system queries specified in a <language> <language> can be one of: **xpath1** (for XPath v 1.0), **xpath2** (for XPath v 2.0), **xquery1** (for Xquery v 1.0), **custom** (for custom queries)

Table 4: GLUE2 EMI-ES Capabilities and their description.

8.1.2.11 Endpoint URL

The element value as defined in the GLUE2 Specification is the network location of the endpoint to contact the related service. The URL MUST contain all the necessary information to contact that endpoint.

8.1.2.12 Endpoint ImplementationName and ImplementationVersion

The values of these mandatory elements should follow the agreement for product name and version; ImplementationName should contain the product name and ImplementationVersion the product version.

Example:
ImplementationName: CREAM
ImplementationVersion: 1.14.0

8.1.2.13 Endpoint Staging

The closed enumeration values of this mandatory element for an EMI-ES endpoint corresponds to the following EMI-ES data-staging functionalities (see Section 1.3):

- none: the server MAY only provide support for client data push or client data pull
- stagingin: the server MUST provide server data pull
- stagingout: the server MUST provide server data push
- staginginout: the server MUST provide both server data pull and server data push

8.1.2.14 Endpoint JobDescription

The value of this multivalued mandatory element should contain at least the string **emies:adl**, a value that identifies the EMI-ES Activity Description Language in GLUE2 context.

Example:

```
<JobDescription>emies:adl</JobDescription>
<JobDescription>nordugrid:xrsl</JobDescription>
```

8.1.2.15 Endpoint OtherInfo

According to the common agreement among services constituting EMI [9] middleware this field MUST contain strings “MiddlewareName=EMI” and “MiddlewareVersion=[EMI version]”. Here [EMI version] is substituted with official EMI version of released middleware.

Example:

```
<OtherInfo>MiddlewareName=EMI</OtherInfo>
<OtherInfo>MiddlewareVersion=2.0.0_1</OtherInfo>
```

8.1.2.16 AccessPolicy

This optional element represents rights to the endpoint based on the basic PolicyScheme.

8.2 Activity information: the EMI-ES activity document

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 GetActivityInfo Response document.

The EMI-ES activity document includes all the GLUE2 ComputingActivity elements plus some EMI-ES specific elements to form a deliberately flat structure.

See also section 13.4 which provides the detailed schema-level example of the activity information model used within the GetActivityInfo operation.

8.2.1 Structure of the EMI-ES activity document

The mandated structure follows a flat XML representation of all the activity attributes.
All the items in the following representation must be considered as XML tags; this representation is used for ease of reading. The structure identifies the mandatory and optional elements.

```
<ActivityInfoDocument>
  <!-- Mandatory -->
  ID:
  State:
  IDFromEndpoint:
  Owner:

  <!-- Optional -->
  CreationTime:
  Validity:
  LocalIDFromManager
```

```
  JobDescription:
  RestartState:
  ExitCode:
  ComputingManagerExitCode:
  Error:
  WaitingPosition:
  UserDomain:
  LocalOwner:
  RequestedTotalWallTime:
  RequestedTotalCPUTime:
  RequestedSlots:
  RequestedApplicationEnvironment:
  Sdtin:
  Stdout:
  StdErr:
  LogDir:
  ExecutionNode:
  Queue:
  UsedTotalWallTime:
  UsedTotalCPUTime:
  UsedMainMemory:
  SubmissionTime:
  ComputingManagerSubmissionTime:
  StartTime:
  ComputingManagerEndTime:
  EndTime:
  WorkingAreaEraseTime:
  ProxyExpirationTime:
  SubmissionHost:
  SubmissionClient:
  OtherMessages:
  StageInDirectory:
  StageOutDirectory:
  SessionDirectory:
  ComputingActivityHistory:
<ActivityInfoDocument>
```

8.2.2 EMI-ES activity document attributes

8.2.2.1 ID

This mandatory element is the universally unique ID of the activity document, as mandated in GLUE2: each GLUE2 ID element MUST have a universally unique valid URI as a value; The IDs must not be interpreted by the user or the system as having any meaning other than identifiers. In particular, there is no relationship between an ID and a network endpoint. IDs MUST be persistent during the lifetime of the activity. The activity document ID MAY be the same as the ActivityID (the one returned by the CreateActivity operation).

8.2.2.2 State

This mandatory element values are taken from a special set of ComputingActivity_t strings with the syntax mandated by GLUE2. Values are defined in this specification for both EMI-ES states and state attributes as follows:

- For EMI-ES states, the state name is prefixed with the string "**emies:"** followed by the EMI-ES state name.

Example: EMI-ES state **processing-queued** is represented as *emies:processing-queued*

- For EMI-ES state attributes, the state attribute name is prefixed with the string "**emiesattr:"** followed by the EMI-ES state attribute name.

Example: EMI-ES state attribute **client-stagein-possible** is represented as *emiesattr:client-stagein-possible*

In case of a state with multiple state attributes, all the state attributes MUST be published together with the state. In addition to EMI-ES states, the state element of the activity document MAY contain also other GLUE2 states belonging to different state models.

Example:

```
        <State>emies:terminal</State>
        <State>emiesattr:app-failure</State>
        <State>emiesattr:processing-failure</State>
        <State>nordugrid:failed</State>
        <State>bes:terminated</State>
```

Several state attributes trigger the publication of other elements of the activity document:
StageInDirectory element MUST be published IF activity state includes **client-stagein-possible** state attribute.
StageOutDirectory element Must be published IF activity state includes **client-stageout-possible** state attribute.

8.2.2.3 IDFromEndpoint

This mandatory element value is a URI that SHOULD contain the EMI-ES activityID obtained from the CreateActivity operation prepended with urn:idfe: to fit requirement for this element to be URI.

8.2.2.4 Owner

This mandatory element value as defined in the GLUE2 model is a String containing the Grid identity of the activitie's owner. It is RECOMMENDED that the Grid identity is specified as the subject name of client's credentials presented during authentication process. If anonymity is requested, the reserved value CONFIDENTIAL should be used.

8.2.2.5 CreationTime

This optional element value contains the creation time of the activity document, and not of the activity.

8.2.2.6 JobDescription

This optional element value as defined in GLUE2 model should be the activity description language used to specify the activity request. Currently this value can only be set to **emies:adl**.

8.2.2.7 StageInDirectory

This optional element is defined as part of the EMI-ES specification. If published, the value MUST contain a URL which can be used by the client to stage in data as described in section **Error! Bookmark not defined.**.
This element MUST be published IF activity state includes **client-stagein-possible** state attribute.

8.2.2.8 StageOutDirectory

This optional element is defined as part of the EMI-ES specification. If published, the value MUST contain a URL which can be used by the client to stage out data as described in section **Error! Bookmark not defined.**. This element MUST be published IF activity state includes **client-stageout-possible** state attribute.

8.2.2.9 SessionDirectory

This optional element is defined as part of the EMI-ES specification. If published, the value MUST contain a URL which can be used by the client to access activity execution directory as described in section **Error! Bookmark not defined.**.

8.2.2.10 ComputingActivityHistory

This optional element is defined as part of the EMI-ES specification, represents historical record about activity state changes and operation requests.

The state changes are represented by ActivityStatus elements which contain information about state with attributes and timestamp representing when state was reached.

The operation requests are presented in Operation elements. Each element contains name of operation with Timestamp when it was requested and either operation succeeded. Operations names are implementation specific and represent internal activity management flow inside service. Names MUST be composed of lowercase ASCII symbols.

Below is an example of such history records.

```
<ComputingActivityHistory>
  <ActivityStatus>
    <State>accepted</State>
```

```
    <Timestamp>2012-07-13T09:20:43.000+02:00</Timestamp>
  </ActivityStatus>
  <ActivityStatus>
    <State>preprocessing</State>
    <Timestamp>2012-07-13T09:20:43.000+02:00</Timestamp>
  </ActivityStatus>
  <ActivityStatus>
    <State>processing-accepting</State>
    <Timestamp>2012-07-13T09:20:43.000+02:00</Timestamp>
  </ActivityStatus>
  <Operation>
    <RequestedOperation>create_activity</RequestedOperation>
    <Timestamp>2012-07-13T09:20:42.000+02:00</Timestamp>
    <Success>true</Success>
  </Operation>
  <Operation>
    <RequestedOperation>start_activity</RequestedOperation>
    <Timestamp>2012-07-13T09:20:47.000+02:00</Timestamp>
    <Success>true</Success>
  </Operation>
</ComputingActivityHistory>
```

9. Activity Description

This section defines the activity description, an XML document that serves as a request description for creating activities on an execution service.

There are several possible use cases of activity description instances:

a) for describing an activity to a end-user client

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

c) to pass activity 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 activity description for direct consumption by the execution service* (case c). Additional information and structures to be used for match making can be added, but will be ignored by the execution service. Activity description instances for clients/brokers are at different level, and outside the scope of this specification.

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.

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 activity 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 honour. 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 honoured 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 EMI ActivityDescription language

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 described in details 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 activity 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 activity 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 activity. 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) [7] which accepts Usage Records (UR) [8].

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 the activity MUST be cancelled. The activities not completed before the expiration time – if defined - 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 activity 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 activity 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 activity (**accepted**, etc). The default value of this element is “terminal”, i.e. by default, notifications will be sent when the activity 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 zero 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 exclusive 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 using ApplicationEnvironment element.

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 61 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. Exact set of allowed options is 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 file system 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 activity 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 activity 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 activity reached POSTPROCESSING phase without failures. Default value is true.

Note: if all default values of UseIf* elements are applied in case of failure or cancelation of the activity, 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 6) 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. Outlook and list of deferred issues

This section lists a number of items that were considered out of scope for the first version of the EMI ES specification, but are of considerable interest for an updated version.

Delegation:

- support credential service (similar to MyProxy)

Activity Management

- more options for filtering in ListActivities

Data staging:

- support file sets
- In order to inform users about the progress of data staging, the activity information SHOULD include a progress value (in percent from 0-99) that indicates how much of the processing in the present state has been already completed. Implementations are free in how they calculate the progress.

Activity Description:

- allow to specify activities and their priorities (QoS attributes)
- "scalable time", i.e. specification of requested time values in relation to some benchmark value.

Information:

- 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.
- AccessPolicy – possible values are undefined. This needs a separate agreement.

Activity Information:

EMI-ES GLUE2 XML rendering is currently based on an XSD schema that features a hierarchical representation of GLUE2 objects. The GLUE2 Working Group is currently working on a non-hierarchical XSD schema that represent the model objects as a flat list. The EMI-ES specification may be eventually updated in future to be compliant with the GLUE2 Group recommendations regarding the Resource and Activity Documents.

12. Editor Information

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

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

13. 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: oxana.smirnova@hep.lu.se

Florida Paganelli
University of Lund, Sweden
Florido.paganelli@hep.lu.se

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

14. 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.

15. 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.

16. Full Copyright Notice

17. 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