AGU Execution Service strawman rendering

(draft version 0.36, 2009/10/05)

Editors:
Balázs Kónya (Lund University/ARC)
Moreno Marzolla (INFN/gLite)
Morris Riedel (FZJ/UNICORE)
Contributors:
Etienne Urbah (CNRS IN2P3 LAL/EDGeS)
to be listed

Contents

1 Introduction

This document provides a realization of an execution service which would match the needs of the ARC, gLite and UNICORE production middleware stacks (the envisioned service therefore is referred to as the “AGU” execution service). The requirements have been collected and described in detail in the "Strawman functionality" document[1]. The strawman identified the following key functionality of an execution service:

1. Job Management
2. Job Monitoring
3. Resource monitoring
4. Access to Session-Directory
5. Data Staging
6. Runtime Environment Support
7. Request Routing
8. Delegation

This rendering document makes an attempt to translate the requirements into concrete data models and operations.

The execution, information and delegation elements of the AGU-ES are separated into three portTypes, because it enables more flexibility for services that may want to re-use a particular functionality provided by a specific portType. The execution and information portTypes are MANDATORY, while the delegation portType is optional.

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.

We consider an activity within this document as being a computational job. As a result an ‘activity’ is equivalent to a single ‘job’.

2 Interface: Execution Port-Type

All operations of this portType are vector operations while each element of this vector is independent. Thus the vector operation does neither define a ‘grouping of activities’ nor ‘transaction of activities’. Hence, the service satisfies each vector element as best as possible while failed independent vector elements will be reported in a fault as response to the vector operation.

The service supports vector operations and may have limits that are based on implementation aspects, but if requests exceed the implementation limits the service communicates this via a fault. There are two options in this aspect

- The service just rejects the request with one single fault specifying the limit that was exceeded.
- The service returns the same length of the vector with responses that may include faults per vector element.

TBD: Where/how is the service vector limit information exposed beforehand? Is there a GLUE2 element? NOT COVERED IN GLUE, meta-scheduler require this information. PGI GLUE2-TUNED field → limit of vector operation invocations…. (but per type of vector operation, createActivity, ChangeActivityStatuses)

TBD: AGREE ON FAULT MECHANISM related to SERVICE VECTOR LIMITs

TBD: Transaction approach (maybe grouping), what happens if one out of this 100 fails? 99 are started or all fail? Problem with LRMS: If 9 have been submitted and the 10th fails – how do you get the 9 directly terminated also…in order to satisfy a potentially specified atomic transaction.

The Execution portType operates on activity IDs in order to ensure a good performance. The execution port-type offers capabilities to manage and monitor activities which are explicitly addressed by their activity Ids. On the other hand, the Information portType is to be used to monitor resources behind the execution service and to perform information queries over activities not explicitly specified by activity Ids.

This port-type is the source for data staging-related information through the activity information.

2.1 CreateActivity operation

Functionality

The operation is used to request the creation of a vector of activities while each activity is described by a job description document. After a certain amount of validation steps, the service creates an instance of each activity that is identified by a global unique ID assigned by the service. The activities are created having the initial state according to the state model and step through the transitions of the state model until they reached the optionally provided holdpoints for each activity within the job description.

Data-Staging Implications

One feature of the execution service is the optional support for client initiated data transfer (i.e. data-push) to the data-stage-in location.

There are different ways to perform client initiated data-staging-ins that are as follows:

- The job description language provides elements (File/Source) that enable the request of client initiated data-staging-in of specified data (e.g. files, directories, etc.). The information might be used by the service in order to track the progress of the client-initiated data-staging-ins.
    - If a service supports this capability, then
        - The service MAY immediately return a data-staging-in location as a response of the CreateActivity() operation, or
        - If the service cannot provide this location during the CreateActivity() operation, then the service MUST expose this information via the GetActivityInfo() operation before the activity enters the ‘pre-processing’ state.
    - If a service does not supports this capability, then
        - UNSUPPORTEDCAPABILITYFAULT
    - If the client wants to perform client initiated data-staging-in, the client MUST also specify in the job description via holdpoints that the job holds in the ‘pre-processing-hold-manual-stage-in’ state in order to perform the client initiated data-staging-in in this state. Using the known data-staging-in location, the client is able to push the data to the execution service while the activity remains in the ‘pre-processing-hold-manual-stage-in’.
- Users might also do ‘manual data-staging’ in order to transfer data to the known data-stage-in location without explicitly specifying these transfers in the job description document.

TBD: Figure of the process
TBD: (Shahbaz): Data URL is received… out of scope how the data is transported to

State Model Implications

The service MUST perform all mandatory and optional validation steps (see below) of the job description document until the ‘pre-processing’ state.
The activities are created having the initial state according to the state model and step through the transitions of the state model until they reached the optionally provided holdpoints (see below) for each activity within the job description.
The state model enables the feedback of ongoing staging activities via dedicated states.

Holdpoints

Holdpoints are used to specify a specific point in the state model where the activity is supposed to hold. A certain amount of these holdpoints are optionally defined within the job description of each activity. The ChangeActivityState() operation can be used to resume an activity that is in a hold state. The activity continues then until a potentially next holdpoint is reached.
The job description schema contains the list of allowed holdpoints that can be specified according to the state model. But this does NOT imply that the service must support all of these allowed holdpoints.

Request

The parameter for the request of the operation for each activity in the vector is as follows:

- One job description document that is encoded in XML and is compliant with AGU-JSDL

Response

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

- For those activities for which a validation of the job description document has been successful, the execution service should return the following information:
  - It MUST return a global unique ID assigned by the service to one activity
    - TBD: UNICORE Gateway issue, EPR → consists of a server identity, DN of the creator of the EPR → Find out why that is required… maybe the address has to be there or the extension mechanism ….
  - It MAY return a stage-in data location that might be used to manually stage-in data or might be helpful to check whether the correct data is staged. The data that was staged-in is made available in the session directory as soon as the activity enters the 'pre-processing' state.
    - Example: URI + 'addressing attributes' TBD:more concrete here
  - It MAY return a session directory, which refers to the directory on the computational resource where the activity is running (aka job sandbox, uspace). This location might be used for immediate interactive access once the activity is created (e.g. visualization, SSH, manual data-stagings). TBD: If this session directory is provided then it should remain the same (content and location) without cleaning the session directory until the job reached the 'finish' state, e.g. in order to monitor the execution.
    - Example: URI + 'addressing attributes' TBD: more concrete here
- For those activities where the job description is represented by a malformed XML document, the environment wherein the service exists MUST return a corresponding fault.
- For those activities where the job description is not compliant with the AGU-JSDL schema, the service MUST throw an → INVALIDJOBDESCRIPTIONFAULT
- For those activities where the job description contains a semantic error that occurs during the semantic validation step (see below), the service MUST throw an → INVALIDJOBDESCRIPTIONSEMANTICFAULT
- For those activities where the job description requests a capability that is not provided by the service, the service MUST throw an → UNSUPPORTEDCAPABILITYFAULT

Faults

INVALIDJOBDESCRIPTIONFAULT
INVALIDJOBDESCRIPTIONSEMANTICFAULT
UNSUPPORTEDCAPABILITYFAULT - Maybe convey which capability is not supported → e.g. not GridFTP here! We are secure!

Job Description Document Validation

The service takes a job description document as input that is encoded in XML. The service MUST perform a certain amount of validation steps and throws a fault if a validation step fails. The list of mandatory validation steps is as follows:

- XML Validation: Check whether XML is a valid XML document (i.e. well-formed and such like).
- Schema validation: Check against the schema of the service
- Semantic validation: This validation includes that the service is capable to understand the values of the XML elements of the job description. It also includes checks that cannot be proofed against the schema. This checks the job description dialect as well as the check whether the information provided for holdpoints actually match states of the state model that allow for hold. Another example, the number of CPUs is represented by a double, so fractional values are syntactically correct.
- Service capability validation: Data-staging is optional and thus a service MAY not be supporting some of the optional job description elements. This includes a check whether the service is actually capable of satisfy the holdpoint requests.
  - TBD: List what we mean by service-capabilites (maybe examples)
  - Examples of optional features (all approved by schema, but MAY not supported by service, we don't force each service to have GridFTP and SRM):
  - GridFTP??? Elements digested while we don't care about the gsiftp url in this point because its resource-specific - also data-stagings → Service Feature
  - SRM-staging Elements – also data-stagings → Service Feature
  - Credential Service

The service MAY also performs 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 optional job description elements MUST NOT be ignored by the service and rather throws a corresponding fault in the case that the service does not support this optional client request.
TBD: The Submit state might need a Hold substate. To be communicated to the PGI group, might be hold to observe the execution → Motivation – depends on the allocation… hold in gLite was introduced because there could be an error during the creation of the activity (bug-isolation)
TBD: LEASE??? Use case –> allocation, EGEE experience! Discussion required

2.2 ChangeActivityStatus operation

Functionality

This operation is used to request the status change of a set of activities. The execution service SHOULD change the state of the requested activities to the desired states from each of the current state or from an optionally defined initial state. In that context only state transitions that are part of the state model are valid while the transition itself MUST NOT passes through other states (i.e. one-step transition only).

Motivation for Change Status

There are a couple of motivations for the ChangeActivityStatus() operation:

- Extension Points: The ChangeActivity operation acts as a hook for service extensions that require state changes.
- Controlling manual data-staging's: After the manual data-staging's have been performed a change of activity status becomes necessary to continue with the processing of the activity.
- Hold/resume: The service MAY offer the hold and corresponding resume functionality of job processing during either pre-processing, delegated, or post-processing states.
- Recovery: The operation can be used for initiating the processing of failed activities again.

Data-Staging Implications

State change operations MUST NOT result in additional data-staging activities besides the ones that are specified in the state model.

State Model Implications

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

Request

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

- An activity identifier;
- The initial status of the activity (optional);
- The desired new status of the activity

Specifying Initial States versus using Current Status

The motivation for optionally specifying an initial state is as follows:
The activity in question is in status B at the time when the client gets this information and makes the decision to request a status change in order to transition from B to D. Over time, the activity transitions to status C while the client still assumes that the activity is in status B. Without the defined initial state (here status B), the service would change the status from C to D while the intention of the transition was from status B to status D.
So the optional parameter with the initial state ensures that actually only transitions are performed that have been requested by the client.
TBD: Figure about the timings with B,C,D form the meeting (Morris)
TBD: what to do with state changes when the service do not understand states like sub-states or states of new extensions (Postponed, Shahbaz issue)

Response

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

- For those activities for which a valid status change (according to the state model) is requested, the execution service should return information on the timing in which the status change will be done
- Time estimation: ETSC - Estimated time to state change, special values: 0 means already transition performed (i.e. immediately), undefined if the service is not able to perform an estimation
- For those activities for which the status change cannot be done (for example, because the activity identifier does not exist, or for which the status change is not legal according to the state model), an appropriate fault structure should be returned. → TBD: FAULTNOTLEGAL
- If the current state is different from the optionally provided initial state the service should throw a fault with information about the current state → INITIALSTATENOTMATCHFAULT

Faults

TBD: FAULTNOTLEGAL
→ INITIALSTATENOTMATCHFAULT

2.3 CancelActivity operation

Functionality

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

State Model Implications

The cancel operation is applicable to any state of the state model except the terminal states 'Failed', 'Finished (with success/error)', 'Cancelled', and 'Purged'.

Request

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

Response

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

- For those activities which can be cancelled, the execution service should return information on the timing in which the cancellation will be done (i.e., whether the activity has already been cancelled, or cancellation will be attempted in the future);
    - Time estimation: ETC - Estimated time to cancellation, special values: 0 means already cancelled (i.e. immediately), undefined if the service is not able to perform an estimation
- For those activities which cannot be cancelled (for example, because the activity identifier does not exist), an appropriate fault structure should be returned. →NOTPOSSIBLEFAULT
- If the cancel operation request is performed on states that do not allow for a 'cancel' transition then a fault should be thrown. → NOTALLOWEDFAULT

Faults

TBD: Faults more exactly…as list
TBD: NOTPOSSIBLEFAULT
TBD: NOTALLLOWEDFAULT

Cancellation of All Activities

A cancellation of all activities of one particular owner is only supported by explicitely specifying all the activity IDs, because an operation such as CancelAllActivities() require the identification of an owner of these activities w/o providing the activity IDs (e.g. certificate identity or such like). This implies an external relationship of the service with another entity and therefore such an operation is considered to be out of scope.

In the cases of where more activities are requested than handled by the service, it is expected that clients are adjusted to the capabilities of the service in terms of choosing the right size of the request vector and maybe consider more than one request to achieve the cancellation of all activities.

2.4 WipeActivity operation

Functionality

This operation can be used to request the complete removal of a set of activities, including the job-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, stage-out) 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 state according to the state model. If the job is not in any final state yet, the service must throw a fault → TBD: JOBNOTFINALSTATEFAULT

Request

The input parameter is a vector of activity identifiers. Another mechanism of specifying a subset of the affected activity identifiers (e.g. filtering, grouping) is 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 out (i.e. immediately), undefined if the service is not able to perform an estimation
- For those activities which cannot be wiped out (for example, because the activity identifier does not exist), an appropriate fault structure should be returned. →NOTPOSSIBLEFAULT
- If the wipe operation request is performed on states that do not allow for an invocation of the WipeActivity() operation then a fault should be thrown. → NOTALLOWEDFAULT

Faults

TBD: Faults more exactly…as list
TBD: JOBNOTFINALSTATEFAULT
TBD →NOTPOSSIBLEFAULT

2.5 GetActivityStatus operation

Functionality

This operation provides the state information according to the state model of a vector of activities with different levels of verbosity. Since the state model comprises of hierarchical states the verbosity is used to define the requested level of depth of the returned information about the state.

Different state models

This operation only provides state information according to the state model within this document, but the more detailed information retrieved by the GetActivityInfo() MAY contain status of other state models.

Data-Staging Implications

The operation returns detailed information, depending on the verbosity level, about every state including the data staging ones as well.

Request

The request includes the following information

- A vector of activity identifiers
- Optionally one verbosity level of details encoded as Integer (1) main state (2) sub-state (3) sub-sub-state and so forth. If the verbosity level is not defined the maximum information about the states is requested.

Response

The response includes the following information:

- The information about the states of each of the activity is provided as a pair of activity identifiers and the corresponding state. The granularity of the state information depends on the requested verbosity level.
- For those activities where the status cannot be requested (for example, because the activity identifier does not exist), an appropriate fault structure should be returned. →NOTPOSSIBLEFAULT

Faults

INVALIDACTIVITYID
UNABLETORETRIEVESTATUS
TBD: ACCESS Control, at which level to do control?

2.6 GetActivityInfo operation

Functionality

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

Data-Staging Implications

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

be used to retrieve that information.

State Model Implications

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

Request

The request includes the following information:

- A vector of activity identifiers
- A vector of optional GLUE2 activity attributes which are requested in addition to the mandatory attributes. In case if this parameter is not specified then the full activity record MUST be returned.

Response

The response includes the following information:

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

Faults

UNKNOWNACTIVITYID
UNKNOWNGLUE2ACTIVITYATTRIBUTE
TBD: extend the GLUE2 activity with sessiondir/stagein info (as part of the DATA model)
TBD: ACCESS Control, at which level do control?

3 Interface: Information Port-Type

The Information portType enables more flexibility due to complex queries on resource and activity information expressed according to the GLUE2 model. Unlike in case of the information-related operations of the Execution port type here the explicit specification of the activity IDs as a request parameter is not required.

This portType might be re-used by information systems that publish information about the resources and its activities.

3.1 QueryResourceInfo operation

Functionality

This operation provides flexible access to query the full resource description formatted according to the XML rendering of GLUE2 information model. The operation can be used to obtain any kind of resource characteristics through a query expressed in one of the supported query languages.

Query Languages

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

Resource Model

The GLUE2 information model provides an XML rendering of the computing service concept. The QueryResource operation MUST be applied on the full GLUE2 document excluding the ComputingActivity elements (see Data model: Resource section for details).

Data-Staging Implications

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

Request

The request includes the following information:

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

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

Response

The response includes the following information:

- The result of the query returned as a plain string.
- Or in case of failed query an appropriate FAULT.

Faults

NOTSUPPORTEDQUERYDIALECT
NOTVALIDQUERYSTATEMENT
QUERYFAILURE
TBD: access control

3.2 QueryActivityInfo operation

Functionality

This operation provides flexible access to query the full set of activities being managed by the Execution Service. The activity information is formatted according to the XML rendering of the GLUE2 information model. This operation can be used to obtain information about a dynamically defined set of activities.

Query Languages

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

Resource Model

The GLUE2 information model provides an XML rendering of computing activities. The computing activity elements MAY appear grouped under different XML nodes within the full GLUE2 XML document which describes a complete Execution Service (see Data model: Resource). Clients MUST ensure that queries are applied on the complete set of activities regardless their position within the GLUE2 document representing a particular Execution Service.

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 or stage-out directories related information. This operation MAY be used to retrieve that information.

State Model Implications

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

Request

The request includes the following information:

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

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

Response

The response includes the following information:

- The result of the query returned as a vector of data structure composed of IDs of matching activities and the corresponding query result strings.
- Or in case of failed query an appropriate FAULT.

Faults

NOTSUPPORTEDQUERYDIALECT
NOTVALIDQUERYSTATEMENT
QUERYFAILURE
TBD: access control
TBD: It may be worth merging the resource & activity queries into a single QueryInfo operation due to the complex nature of GLUE2 document.

4 Interface: Delegation Port-Type

The delegation port-type provides operations by which the clients can temporarily convey their X509 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.
TBD: is this a static delegation? Implications?

TBD: support for other credential format???
TBD: question of renewal?
TBD:Extend the data staging block of JSDL with delegation information (e.g DelegationID)

4.1 InitDelegation operation

Functionality
The *InitDelegation* operation starts the delegation procedure by asking for a certificate signing request from the server. The server answers with a certificate signing request which includes the public key for the new delegated credentials. The *PutDelegation*() method of the Delegation portType has to be called to complete the delegation procedure.
Data-Staging Implications
The DelegationID response MAY be used in the DataStaging element of the job description in order to assign a delegated credential (once the delegation process was completed) to a datastaging operation to be performed by the Execution Service on behalf of the client.
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 of this operation contains no input parameters.
Response
The response includes the following information:
- DelegationID. This string is used to uniquely refer to this delegation session.
- Certificate signing request. A string which contains an X509 certificate signing request.
- A FAULT in case of an internal service error.

Faults
INTERNALSERVICEDELEGATIONFAULT

4.2 PutDelegation operation

Functionality
The *PutDelegation* operation completes the delegation procedure by sending the signed proxy certificate along with the *DelegationID* to the server. The signed certificate is based on the certificate signing request previously retrieved together with the *DelegationID* via an *InitDelegation* operation invocation.
Data-Staging Implications
The delegated credential MAY be used by the Execution Service to carry out datastaging operations on behalf of the user. The delegated credential is assigned to the data transfer via a dedicated job description element of the DataStaging block of the AGU-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
- the RFC 3280 style proxy certificate, signed by the client

Response
The response includes the following information:
- a SUCCESS string in case of successful Delegationput operation.
- or if a DelegationID is unknown to the service an UNKNOWNDELEGATIONID fault to be thrown

TBD: any useful information for RESPONSE?
Faults
UNKNOWNDELEGATIONID

5 Data Model: Activity States

5.1 Hierarchical state model

Scope

The scope of this 'PGI Single Job State Model' is only vector operations on Single Jobs.
Each element of this vector is independent. Thus the vector operation does neither define a 'grouping of activities' nor 'transaction of activities'. Hence, the service satisfies each vector element as best as possible while failed independent vector elements will be reported in a fault as response to the vector operation.
This excludes Collection Jobs, Parameter Sweep Jobs, DAG Jobs, ... described at http://forge.gridforum.org/sf/go/doc15735?nav=1 (for which the Execution Service automatically submits several Single Jobs).
~~In particular, 'Create multiple Activities' as defined in chapter 3.1.2 of 'Strawman of the Geneva grid job Execution Service (GES)' at http://forge.gridforum.org/sf/go/doc15590?nav=1 is NOT covered by the 'Single Job State Model'.~~

Any implementation of the PGI Execution Service MUST implement all first level Job states, exactly as described in this document. It MAY implement substates. If these substates correspond to substates described in this document, then these substates MUST be implemented exactly as described in this document.

5.1.1 User Job Submission

This initial state requires that the Job Submitter (which can be a human User, a software client, or the Execution Service itself) has prepared a Job description (standard is JSDL) for a Single Job.
The Job description must contain a request to run a script or an executable. The Job description can contain:
– Requests of automatic staging (performed by the Execution Service without action of the Submitter),
– Requests of manual staging (where the Execution Service notifies the Submitter the location of the file(s) to be staged).

5.1.2 Submitted

The Job goes into the 'Submitted' state in the 2 following cases :
– When the User has submitted a readable Job description,
– When the Execution Service has triggered the 'Automatic Resubmission' transition for a Job in the 'Failed' state, in which case the Job retains its original Jobid (or EPR).
In case of a new submitted job, the Execution Service immediately allocates a Jobid (or an EPR) to the Job and sends it back to the Submitter.
That permits the Submitter, at any time, to request a Job cancellation, which sends the Job to the 'Cancelled' state.
If the Execution Service finds that the Job can NOT be executed at all (for example: bad Job description, Requirements exceeding quotas), then it triggers the 'Submitted Failure' transition which sends the Job to the 'Failed' state.
'Cancellation' and 'Failure' can happen in any substate of the 'Submitted' state.
When the Execution Service has found a set of resources adequate for the execution of the Job :
– It decides the storage resources which are necessary for Stage-in, Job execution and Stage-out, but does NOT allocate them yet,
– It decides which computing resources will execute the Job, but does NOT allocate them yet,
– It triggers the 'Goes to Pre-processing' transition which sends the Job to the 'Pre-processing' state.
The Execution Service updates the Job status, but does NOT specifically notify the Submitter.
The Job can stay in this 'Submitted' state a long time, for various reasons, for example :
Submitted:Incoming
In this substate, the Execution Service has previous submitted Jobs to process first.
Submitted:Waiting
In this substate, the resources required by the Job are NOT simultaneously available yet.

Submitted:Outgoing
In this substate, the Execution Service has previous Jobs to send to the 'Pre-processing' state first.

5.1.3 Pre-processing

The Job goes into the 'Pre-processing' state from the 'Submitted' state when the Execution Service has triggered the 'Submitted goes to Pre-processing' transition.
At the beginning of the 'Pre-processing' state, the Execution Service allocates the storage resources which are necessary for Stage-in.
Inside the 'Pre-processing' state, the computing resources are decided, but NOT allocated yet. This permits the Job to stay inside the 'Pre-processing' state as long as necessary to perform automatic Stage-in, without blocking computing resources.
In case of an unrecoverable error, the Execution Service triggers the 'Pre-processing Failure' transition which sends the Job to the 'Failed' state.
'Cancellation' and 'Failure' can happen in any substate of the 'Pre-processing' state.
When the Execution Service has verified that all Stage-in operations have been successfully performed, it triggers the 'Pre-processing goes to Delegated' transition which sends the Job to the 'Delegated' state.
The Execution Service updates the Job status, but does NOT specifically notify the Submitter.
The Job can stay in this 'Pre-processing' state a long time, for various reasons, for example :
Pre-processing:Incoming
In this substate, the Execution Service has previous pre-processing Jobs to process first.
Pre-processing:Automatic-Stage-In
In this substate, the Execution Service automatically performs Stage-in.
Pre-processing:Hold
The Job can go into the 'Hold' substate of the 'Pre-processing' state, for example in the 3 following cases :
– Manual Stage-in is necessary,
– The Submitter has voluntarily suspended the Job,
– Pre-processing has failed in a manner that can perhaps be corrected by the Submitter.
Inside this 'Hold' substate :
– The storage resources which are necessary for Stage-in are already allocated,
– The computing resources are decided, but NOT allocated yet.
This permits the Job to stay inside this 'Hold' substate as long as necessary to perform manual Stage-in or any other User action, without blocking computing resources.
The Execution Service DOES specifically notify the Submitter that the Job requires User action.
The Execution Service updates the Job status.
The Execution Service can send the Job back to a normal substate of 'Pre-processing' when the Submitter has notified the Execution Service that automatic processing can be resumed.
The Job can stay in this 'Hold' substate a long time, for example inside following third level substates :
Pre-processing:Manual-Stage-In
In this substate of 'Hold', the Submitter has to manually perform Stage-in.
Pre-processing:Suspended
In this substate of 'Hold', the Job has been voluntarily suspended by the Submitter.
Pre-processing:Failed-Recoverable
In this substate of 'Hold', the Submitter has to perform corrective actions.
Pre-processing:Outgoing
In this substate, the Execution Service has previous Jobs to send to the 'Delegated' state first.

5.1.4 Delegated

The Job goes into the 'Delegated' state from the 'Pre-Processing' state when the Execution Service has triggered the 'Pre-Processing goes to Delegated' transition.
At the beginning of the 'Delegated' state :
– The storage resources which were necessary for Stage-in are already allocated, and Stage-in has already been successfully completed,
– The Execution Service allocates the storage resources which are necessary for Job execution (including Stage-out),
– The Execution Service allocates all necessary computing resources to the Job.
So, the Job should make best use of the allocated computing resources : It should NOT sleep or halt. The Execution Service should consider most Job failures as unrecoverable.
The 'Delegated' state covers both :

– Execution by a batch system belonging to the original grid infrastructure; In that case, the Execution Service must be able to give to the Submitter the name of the batch queue (+ perhaps local job id),
– Forwarding to the Execution Service of another grid infrastructure; In that case, the Execution Service must be able to give to the Submitter the corresponding EPR; Detailed follow-up of the Job status inside the other grid infrastructure is NOT performed at all by the original Execution Service, but optionally by the Submitter using the returned EPR.
In case of an unrecoverable error, the Execution Service triggers the 'Delegated Failure' transition which sends the Job to the 'Failed' state.
'Cancellation' and 'Failure' can happen in any substate of the 'Delegated' state.
At the end of the 'Delegated' state, the Execution Service deallocates :
– All computing resources from the Job,
– The storage resources which where required only for Job execution,
– The storage resources which where required only for Stage-in.
When the Execution Service has verified that Job execution has been entirely performed (successfully or with a non-zero return code), it triggers the 'Delegated goes to Post-Processing' transition which sends the Job to the 'Post-Processing' state.
The Execution Service updates the Job status, but does NOT specifically notify the Submitter.
The Job can stay in this 'Delegated' state a long time, for various reasons, for example:
Delegated:Incoming
In this substate, the Execution Service has previous delegated Jobs to process first, for example allocate computing resources to them.
Delegated:Running
In this substate, the Job is in execution inside a batch system belonging to the original grid infrastructure.
Delegated:Forwarded
In this substate, the original Execution Service has forwarded the Job to the Execution Service of another grid infrastructure.
Delegated:Hold
The Job can go into the 'Hold' substate of the 'Delegated' state in rare cases, for example when :
– The Submitter has voluntarily suspended the Job,
– Job execution has failed in a manner that can perhaps be corrected by the Submitter.
Inside this 'Hold' substate, all storage and computing resources which are necessary for Job execution are already allocated.
So the Job should stay inside this 'Hold' substate as shortly as possible, and the Execution Service should trigger a 'Delegated Failure' transition on expiration of a short time-out.
The Execution Service DOES specifically notify the Submitter that the Job requires urgent User action.
The Execution Service updates the Job status.
The Execution Service can send the Job back to a normal substate of 'Delegated' when the Submitter has notified the Execution Service that automatic processing can be resumed.
The Job should stay in this 'Hold' substate as shortly as possible, for example inside following third level substates :
Delegated:Hold:Suspended
In this substate of 'Hold', the Submitter has voluntarily suspended the Job, and has to urgently perform some actions.
Delegated.Hold:Failed-Recoverable
In this substate of 'Hold', the Submitter has to urgently perform corrective actions.
Delegated:Outgoing
In this substate, the Execution Service has previous Jobs to process first, for example deallocate computing resources from them, then send them to the 'Post-Processing' state.

5.1.5 Post-processing

The Job goes into the 'Post-processing' state from the 'Delegated' state when the Execution Service has triggered the 'Delegated goes to Post-processing' transition.
At the beginning of the 'Post-processing' state :
– The storage resources which are necessary for Stage-out are already allocated,
– The computing resources are already deallocated.
This permits the Job to stay inside the 'Post-processing' state as long as necessary to perform automatic Stage-out, without blocking computing resources.
In case of an unrecoverable error, the Execution Service triggers the 'Post-processing Failure' transition which sends the Job to the 'Failed' state.

'Cancellation' and 'Failure' can happen in any substate of the 'Post-processing' state.
At the end of the 'Post-processing' state, the Execution Service deallocates the storage resources which where required only for Stage-out.
When the Execution Service has verified that all Stage-out operations have been successfully performed, it triggers the 'Finishes with Success or Error' transition which sends the Job to the 'Finished with Success or Error' state.
The Execution Service updates the Job status, but does NOT specifically notify the Submitter.
The Job can stay in this 'Post-processing' state a long time, for various reasons, for example :
Post-processing:Incoming
In this substate, the Execution Service has previous post-processing Jobs to process first.
Post-processing:Automatic-Stage-Out
In this substate, the Execution Service automatically performs Stage-out.
Post-processing:Hold
The Job can go into the 'Hold' substate of the 'Post-processing' state, for example in the 3 following cases :
– Manual Stage-out is necessary,
– The Submitter has voluntarily suspended the Job,
– Post-processing has failed in a manner that can perhaps be corrected by the Submitter.
Inside this 'Hold' substate :
– The storage resources which are necessary for Stage-out are already allocated,
– The computing resources are already deallocated.
This permits the Job to stay inside this 'Hold' substate as long as necessary to perform manual Stage-out or any other User action, without blocking computing resources.
The Execution Service DOES specifically notify the Submitter that the Job requires User action.
The Execution Service updates the Job status.
The Execution Service can send the Job back to a normal substate of 'Post-processing' when the Submitter has notified the Execution Service that automatic processing can be resumed.
The Job can stay in this 'Hold' substate a long time, for example inside following third level substates :
Post-processing:Hold:Manual-Stage-Out
In this substate of 'Hold', the Submitter has to manually perform Stage-out.
Post-processing:Hold:Suspended
In this substate of 'Hold', the Job has been voluntarily suspended by the Submitter.
Post-processing:Hold:Failed-Recoverable
In this substate of 'Hold', the Submitter has to perform corrective actions.
Post-processing:Outgoing
In this substate, the Execution Service has previous Jobs to send to the 'Finished with Success or Error' state first.

5.1.6 Finished with Success or Error

The Job goes into the 'Finished with Success or Error' state from the 'Post-processing' state when the Execution Service has triggered the 'Finishes with Success or Error' transition.
At the beginning of the 'Finished with Success or Error' state, the Execution Service deallocates all storage and computing resources which would still be allocated to the Job.
Inside the 'Finished with Success or Error' state, the Submitter can only retrieve the Job log and the Output Sandbox.
The Execution Service DOES specifically notify the Submitter that the Job is finished and that the Output Sandbox is available for retrieval.
The Execution Service can purge the Output Sandbox :
– when the Submitter has successfully retrieved it,
– or after a long defined period (several weeks).
The Execution Service then triggers the 'Purge after Finished' transition to send the Job to the 'Purged' state.

5.1.7 Failed

The Job goes into the 'Failed' state from any state when the Execution Service has detected an unrecoverable Job Failure.
At the beginning of the 'Failed' state, the Execution Service deallocates all storage and computing resources which would still be allocated to the Job.
The Execution Service can trigger the 'Automatic Resubmission' transition to send the Job back to the

'Submitted' state. This is possible only from this 'Failed' state, where we are sure that the Execution Service has completely deallocated all storage and computing resources from the Job.
Inside the 'Failed' state, the Submitter can only retrieve the Job log, and perhaps bits of the Output Sandbox which the Job could generate before failure.
The Execution Service DOES specifically notify the Submitter that the Job is 'Failed' and if bits of the Output Sandbox are available for retrieval.
The Execution Service can purge the Output Sandbox :
– if it is empty,
– or when the Submitter has successfully retrieved the available bits,
– or after a long defined period (several weeks).
The Execution Service then triggers the 'Purge after Failure' transition to send the Job to the 'Purged' state.

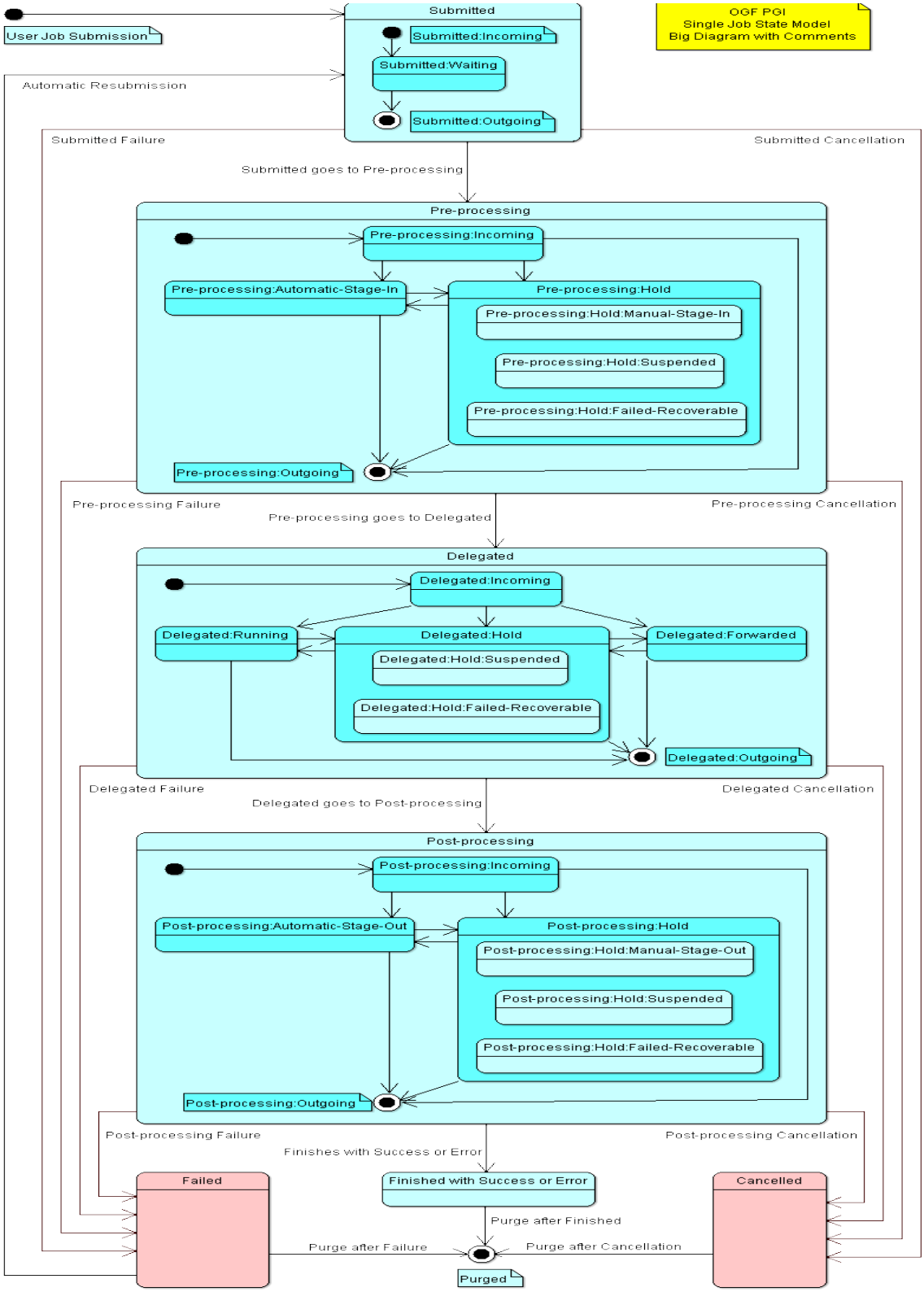
5.1.8 Cancelled

The Job goes into the 'Cancelled' state from any state when the Submitter has cancelled the Job.
At the beginning of the 'Cancelled' state, the Execution Service deallocates all storage and computing resources which would still be allocated to the Job.
Inside the 'Cancelled' state, the Submitter can only retrieve the Job log, and perhaps bits of the Output Sandbox which the Job could generate before cancellation.
The Execution Service DOES specifically notify the Submitter that the Job is 'Cancelled' and if bits of the Output Sandbox are available for retrieval.
The Execution Service can purge the Output Sandbox :
– if it is empty,
– or when the Submitter has successfully retrieved the available bits,
– or after a long defined period (several weeks).
The Execution Service then triggers the 'Purge after Cancellation' transition to send the Job to the 'Purged' state.

5.1.9 Purged

The Job goes into the terminal 'Purged' state from 3 states :
– From the 'Finished' state when the Execution Service has triggered the 'Purge after Finished' transition,
– From the 'Failed' state when the Execution Service has triggered the 'Purge after Failure' transition.
– From the 'Cancelled' state when the Execution Service has triggered the 'Purge after Cancellation' transition.
Inside the 'Purged' state :
– All storage and computing resources are already deallocated from the Job.
– The Output Sandbox has already been purged.
- The Submitter should still be able to retrieve the Job log (For example, inside EGEE, if the Job log has been purged from the LB, it must still be available in the JP).
TBD: synchronize the entire section with the rest of the document!!!

5.2 State diagram



5.3 Relation to data staging

Matrix of states vs. data actions
Meta: From interface usage perspective – what consumers can assume from the functionality of the interface.

States / Operations	Grid Job id	Data Req. for stage-in	Temporarily Data generated	data Req. for stage-out
Cancel Operation	Available	Not available	Not available	Available
Wipe Operation	Not available	Not available	Not available	Not available
Finished w success	Available	Not available	Not available	Available
Finished w error	Available	Available	Available	Available
Failed	Available	Not available	Not available	Available
Hold	Available	Available	Available	Available

Important sentences for the understanding

- Valuable temp. created data should be in any case marked for data-staging out
- The analysis if stage-out makes sense can be satisfied by using manual data staging in this context, for instance when cancelling a job.
- Data indicated for data-staging out can be staged automatically and manually.
- Elements that should be staged out manually also have to be marked for data-staging since otherwise they are gone and thus not accessible with tools such as SSH.
- Users may mark the complete session-directory as stage-out and thus have the chance to keep the maximum flexibility in terms of data-stagings with tools like SSH (i.e. SCP) and such like.
- Data that has been already staged by the service to an external target (i.e. remote storage elements) are not affected by these operations.

TBD: The PGI state model is currently not consistent with the above table:

- The description of the Cancel for instance is not consistent of the cancel
- To be discussed is the Finished sub-state with error / success
- The PGI state model does not affect data

TBD: What about data-stagings that are in progress while cancel pressed

6 Data Model: Resource and Activity representation

For the definition of the execution service the following internal structures need to be defined: Job description, job state model and execution service resource model. This section describes the resource model based on the GLUE2 information model.

7 Data Model: Job description

The rendering of the AGU execution service requires the definition of the following data structures: Job description, activity state model, execution service and activity resource models. This section defines the AGU job description.
TBD: explicitly require/mention XML?

7.1 AGU-JSDL high-level structure

```
<JobSpecification>
  <JobIdentification... />?
  <Application .../>?
  <Resources .../>?
```

```
  <DataStaging .../>?
       <File... />*
       <Directory... />*
  </DataStaging>
  <Meta/>?
</JobSepcification>
```

7.2 AGU-JSDL Types

In addition to the atomic types such as string, integer, datetime and URL the AGU-JSDL utilizes the following reusable complex structures:

ExecutableType

In order to generalize the description of any executable a new complex *ExecutableType* composed of *Path* and *Argument* has been introduced.

Path

This mandatory string element specify the path of the executable relative to the session directory. Multiplicity is one.

Argument

This optional element is a string specifying an argument for the executable. The multiplicity is zero or more with strict ordering. There is no default value of this element.

Comment: introduced complex structure and semantic changes compared to the Argument element of the GFD 136.

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.

Comment: new. The proposed new structure replaces ApplicationName, ApplicationVersion, OperatingSystem, OperatingSystemType, OperatingSystemVersion, OperatingSystemName of GFD136.

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.

Comment: new. The proposed new structure replaces ApplicationName, ApplicationVersion, OperatingSystem, OperatingSystemType, OperatingSystemVersion, OperatingSystemName of GFD136.

ScalableTime

This optional complex structure is used to define scalable time request optionally depending on benchmark results. Multiplicity zero or one. There is no default value of this element. ScalableTime is composed of *Value, BenchmarkType* and *BenchmarkValue*.

Comment: new.

Value

Mandatory integer range. Multiplicity is one. There is no default value of this element.

BenchmarkType

Optional string element specify the type of the benchmark used to obtain the BenchmarkValue. Multiplicity is zero or one. There is no default value of this element. It is an open enumeration with values of the GLUE2 Benchmark_t type:

- bogomips
- cfp2006
- cint2006
- linpack
- specfp2000
- specint2000

BenchmarkValue

Optional integer element specify the result of the benchmark defined by BenchmarkType. Multiplicity zero or one. There is no default value of this element.

Range

The *Range* is a simplified structure to express required minimum and maximum values. It has *Min* and *Max*

sub-element and one of them is mandatory. In the XML realization it may be necessary to introduce different range types e.g. integer and float values.

7.3 AGU-JSDL elements

7.3.1 JobIdentification

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

7.3.1.1 JobName

This optional string element MAY be specified by a user to name the job. It may not be unique to a particular job description, which means that a user MAY specify the same JobName for multiple job descriptions. Some project defines their own format for the JobName in order to categorize and explicitly define the particular version of job they run. However the recommended way to attach user specified categories to the job is to use UserTag element.
This element has no default value. The multiplicity of this element is zero or one.
Comment: same as JobName of the GFD 136.

7.3.1.2 Description

This optional string element may contain a longer textual description of job. This element has no default value. The multiplicity of this element is zero or one.
Comment: new.

7.3.1.3 JobType

This optional element provides a classification of the job. 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*: a job submitted as part of a collection of individual jobs which do not
- communicate among them,
- *parallelelement*: a job submitted as part of a collection of individual jobs which communicate among them,
- *single*: an individual stand-alone job,
- *workflownode*: a job submitted as part of a workflow.

Comment: new.

7.3.1.4 UserTag

This optional element is for human readable comments, tags for free grouping or identifying different jobs.
This element has no default value. The multiplicity of this element is zero or more. The type of this element is string.
Comment: new.

7.3.1.5 JobVOName

This element should be used to indicate the Virtual Organization of the job. If this optional element is not present then it is not defined. The multiplicity of this element is zero or one. The type of this element is string. It is recommended not to use this information to access control because authenticity is not garanteed.
Comment: new similar information was incorrectly force into the JobProject of GFD 136.

7.3.2 Application

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

7.3.2.1 Executable

This mandatory complex element with ExecutableType type is specifying the main executable of the job. Multiplicity is one.
Comment: introduced complex structure and semantic changes in compare to the Executable element of the GFD 136.

7.3.2.2 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.
Comment: same as Input in the GFD 136.

7.3.2.3 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.
Comment: same as Output in the GFD 136.

7.3.2.4 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.
Comment: same as Error in the GFD 136.

7.3.2.5 Join

This is an optional boolean element. If it is true it specifies that standard output and error should be joined into the file specified by Output element. If it is not defined then the default value is false, and joining is not done. Multiplicity is zero or one.
Comment: new.

7.3.2.6 Environment

This optional element specifies 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.
Comment: introduced complex structure in compare to the Environment element of the GFD 136.

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.

7.3.2.7 Prologue

This optional ExecutionType element specifies the path of the executable and its arguments which will be run before the actual executable is run. If the prologue fails the job execution terminates. There is no default value of this element.
Comment: new.

7.3.2.8 Epilogue

This optional ExecutionType element specifies the path of the executable and its arguments which will be run after the actual executable is run. If the epilogue fails the job execution terminates. There is no default value of this element.

Comment: new.

7.3.2.9 LogDir

This optional string element defines the name of the directory relative to the session directory containing grid-specific diagnostics per job. This directory is kept in the session directory of the job to be available for retrieval. Multiplicity is zero or one. There is no default value of this element.
Comment: new.

7.3.2.10 RemoteLogging

The optional elements specifies an URL for a logging service to send reports about job. There is no default value of this element. 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. Multiplicity is zero or more.
Comment: new.

7.3.2.11 Rerun

An optional integer specifying the number of possible reruns the user can initiate in case of a system failure. If it is not defined, the default value is 0. The actual reruns can take place in different part of the executation chain. Multiplicity is zero or one.
Comment: new.

7.3.2.12 ExpiryTime

The ExpiryTime element is optional and specifies the date and time after which the processing of a passive job can be terminated. Multiplicity is zero or one. There is no default value of this element.
Comment: new.

7.3.2.13 HoldPoints

TBD: n HoldPoint re-write -→
This optional boolean element defines whether the created job should be put on hold and processing suspended until explicit state change request arrives. The default value of this element is false. Multiplicity zero, one or more
Comment: new.

7.3.2.14 ProcessingStartTime

This optional datetime element defines the requested start date and time of the job as passed through to the underlying LRMS. There is no default value of this element. Multiplicity is zero or one.
Comment: new.

7.3.2.15 Notification

This optional string element defines the request in custom format for e-mail notifications on job status change. Multiplicity is zero or more. There is no default value of this element.
Comment: new.

7.3.2.16 CredentialService

This optional URL elements specifies an endpoint which may be used to contact a server to renew/extend delegated proxy credential used by a submitted job. There is no default value of this element but unexpected behavior may occur because of expired proxies. Multiplicity is zero or more.
Comment: new.

7.3.2.17 AccessControl

Free format place holder for policy description.
Comment: new.

7.3.3 Resources

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

7.3.3.1 OperatingSystem

This optional complex element specifies the operating system requirement to be provided for the grid job. Its type is SoftwareRequirement. Multiplicity is zero or more. 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
- *suse*: SUSE
- *ubuntu*: Ubuntu
- *windowsvista*: Microsoft Windows Vista
- *windowsxp*: Microsoft Windows XP

Comment: new structure obsoletes OperatingSystem, OperatingSystemType, OperatingSystemVerion, OperatingSystemName of the GFD136.

7.3.3.2 Platform

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

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

Comment: new element obsoletes CPUArchitecture, CPUArchitectureName of the GFD136.

7.3.3.3 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
- *infiniband*: Network based on Infiniband technology
- *myrinet*: Network based Myrinet technology

Comment: new.

7.3.3.4 NodeAccess

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

- *inbound*: inbound network is required to grid job
- *outbound*: outbound network is required to grid job
- *inoutbound*: both direction is required to grid job

Comment: new.

7.3.3.5 IndividualPhysicalMemory

This optional element is a integer range value specifying the amount of physical memory required to be available on every node of the computing element used by (a multi slot) grid job. The amount is given in bytes. Multiplicity is zero or one.
Comment: semantically modified version of IndividualPhysicalMemory of the GFD136. This element together with the Slots is sufficient to request physical memory therefore TotalPhysicalMemory of the GFD136 is not used.

7.3.3.6 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) grid job. The amount is given in bytes. Multiplicity is zero or one.
Comment: semantically modified version of IndividualVirtualMemory of the GFD136. This element together with the Slots is sufficient to request physical memory therefore TotalVirtualMemory of the GFD136 is not used.

7.3.3.7 DiskSpaceRequirement

This optional complex element composed of *DiskSpace, CacheDiskSpace* and *SessionDiskSpace* describes the space requirements of the grid job in terms of total size and its distribution over cache and session area. There is no default value of this element. Multiplicity is zero or one.
Comment: new structure obsoletes IndividualDiskSpace, TotalDiskSpace, DiskSpace and Filesystem of the GFD136.

7.3.3.7.1 DiskSpace

This mandatory integer element specifies the total required disk space of the grid job in bytes. The optional CacheDiskSpace and SessionDiskSpace elements can refine the distribution of this disk space over cache and session areas. There is no default value of this element. Multiplicity is one.

7.3.3.7.2 CacheDiskSpace

This optional integer element contains the requested amount of disk space in bytes on a longer lifetime cache territory. There is no default value of this element. Multiplicity is zero or one.

7.3.3.7.3 SessionDiskSpace

This optional integer element contains the requested amount of disk space in bytes on a session directory of the job. There is no default value of this element. Multiplicity is zero or one.

7.3.3.8 SessionLifetime

An optional duration element specifying the maximal time to keep the session directory of the job on the computing service upon job completion. There is no default value of this element. Multiplicity is zero or one.
Comment: new.

7.3.3.9 SessionAccessMode

An optional element specifying the mode how the user may want to access the session directory on the Computing Service. Multiplicity is zero or one. If it is not defined that the user not interested to access session directory in any way. It is closed enumeration with the following values:

- *readwrite:*both read and write access is required
- *read*: only read access is required.

7.3.3.10 IndividualCPUTime

This is an optional scalable time element where the Value sub-element specifies the number of CPU seconds required individually on every node of the computing element used by (a multi slot) grid job. There is no default value of this element. Multiplicity is zero or one.
Comment: new structure obsoletes the IndividualCPUTime of the GFD136.

7.3.3.11 TotalCPUTime

This is an optional scalable time element where the Value sub-element specifies the total number of CPU seconds required, across all CPUs of the computing element used by (a multi slot) grid job. There is no default value of this element. Multiplicity is zero or one. For singe slot jobs it is recommended to use IndividualCPUTime instead of this element.
Comment: new structure obsoletes the TotalCPUTime of the GFD136.

7.3.3.12 IndividualWallTime

This is an optional scalable time element where the Value sub-element specifies the wall clock time of single slot job or every element of a multi slot job. There is no default value of this element. Multiplicity is zero or one.
Comment: it obsoletes the WallTimeLimit of the Posix of the GFD 136.

7.3.3.13 TotalWallTime

This is an optional scalable time element where the Value sub-element specifies the wall clock time required for multi slot job from the start of the first process until the completion of the last process. There is no default value of this element. Multiplicity is zero or one. For singe slot jobs it is recommended to use IndividualWallTime instead of this element.
Comment: new.

7.3.3.14 Homogeneous

This optional boolean element defines wether the job is required to be run on computing element with a single type of Execution Environment as defined in the GLUE2 Homogenous attribute. Multiplicity zero or one. The default value is false.
Comment: new.

7.3.3.15 Benchmark

This complex optional element enables the request of clusters with specific benchmark results advertised according to the GLUE2 Benchmark element. The element is composed of *Type* and *Value* pair. Multiplicity is on or more. If multiple elements are given they are connected by OR logical operation.
Comment: new.

7.3.3.15.1 Type

Mandatory element specify the type of the benchmark used to obtain the Value. Multiplicity is one. It is an open enumeration with values of the GLUE2 Benchmark_t type:

- bogomips
- cfp2006
- cint2006
- linpack
- specfp2000
- specint2000

7.3.3.15.2 Value

Mandatory integer range element specify the requested result range of the benchmark defined by Type. Multiplicity is one.

7.3.3.16 CEType

This optional SoftwareRequirement element defines the requested computing service implementation type. There is no default value of this element the Family sub element of the Software sub-element of the SoftwareRequirement is not used in the CEType. Multiplicity is one or more.
Comment: new.

7.3.3.17 SlotRequirement

This optional complex element specify the requested count of slots and its optional distribution for multi-slot grid jobs. The required NumberOfSlots subelement MAY used for machmaking but the optional ProcessPerHosts, ThreadsPerProcesses are only meant to be passed to the underlying LRMS for scheduling. The ProcessPerHost, ThreadsPerProcesses elements can refine distribution of NumberOfSlots and the optional SPMDVariation characterizes the parallel application.
Comment: new and obsolates TotalCPUCount of the GFD 136 and reuses elements from the SPMD profile GFD 115.

7.3.3.17.1 NumberOfSlots

This mandatory integer range element specify the total number of slots where the separate instances of the executable of the parallel application (multi slot job) requested to run.
Slot is a portion of exectubale time offerd by a logical CPU. Logical CPU correspondes to a CPU as visible by the Operating System running on either real or virtual machine. The slots can be thread, process etc. Multiplicity is one.

7.3.3.17.2 ProcessPerHost

This optional integer element specifies the number of instances of the executable that the consuming system MUST start per host. There is no default value of this element. Multiplicity is zero or one.

7.3.3.17.3 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.

7.3.3.17.4 SPMDVariation

This optional element defines the type of multi-slot application. There is no default value of this element. It is a open enumeration with values of from the GFD 115.

- *MPI:* Any MPI environment
- *GridMPI:*The GridMPI environment2
- *IntelMPI*:The Intel MPI environment3
- *LAM-MPI*:The LAM/MPI environment4
- *MPICH1*:The MPICH version 1 environment5
- *MPICH2*: The MPICH version 2 environment6
- *MPICH-GM*: The MPICH-GM environment7
- *MPICH-MX*: The MPICH-MX environment8
- *MVAPICH*: The MVAPICH (MPI-1) environment9
- *MVAPICH2*: The MVAPICH2 (MPI-2) environment10
- *OpenMPI*: The Open MPI environment11

- *POE*: The POE (IBM MPI) environment12
- *PVM*: A Parallel Virtual Machine environment13

7.3.3.18 CandidateTarget

This optional complex element represents the computing service prefered to be used. The element is composed of *EndPointURL* and *QueueName* elements. Multiplicity is zero or more. There is no default value of this element.
Comment: the new structure obsolates the CandidateHosts, HostName of the GFD 136.

7.3.3.18.1 EndPointURL

This mandatory element contains the computing element endpoint's URL. Multiplicity is one.

7.3.3.18.2 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.

7.3.3.19 RunTimeEnvironment

This optional SoftwareRequirement element defines the runtime environment required by the job. Multiplicity is zero or one. There is no default value of this element.
Comment: the complex element obsoletes the Application, ApplicationName, ApplicationVersion of the GFD 136.

7.3.4 DataStaging

Data staging is a optional complex element which describes all the files and directories should be transferred to the computing element (stage in) and the files and directories that should be transfered from the computing element (stage out). The data movement can be performed by both the client and execution service. The Data staging element is composed of File and Directory sub-elements which order is not significant. Multiplicity is zero or one. There is no default value of this element.
Comment: the new structure largely redefines the DataStaging of the GFD 136.
TBD: Data-staging for non expl specified data-stagings elements

7.3.4.1 File

This is an optional complex element which defines stage in or stage out request of a file. Multiplicity is zero or more. The ordering of the Target and Source sub-elements not significant. The file data staging element MUST contain at least one Source or Target.

7.3.4.1.1 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 or cache.

7.3.4.1.2 Source

This optional complex element specifies the source location of the stage in data transfer of a file. Multiplicity zero or more. In case of multiple sources it up to the computing service implementation how utilize them. There is no default value of this element.

7.3.4.1.2.1 URL

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

7.3.4.1.2.2 Threads

This optional integer element defines the number of threads which can be optionally used by execution service during the staging process. The default value is 1. Multiplicity is zero or one.

7.3.4.1.2.3 DataIndexingService

This optional URL element contains the location of the data indexing service. Multiplicity is zero or more. There is no default value of this element. It is up to a user to make sure the computing service is able communicate to the given data indexing service.

7.3.4.1.3 Target

This optional complex element specifies the target location of the stage out data transfer of a file. Multiplicity zero or more. In case of multiple targets the execution service MUST upload the file to all the mandatory targets or at least one of the targets in case there was no mandatory element defined. There is no default value of this element.

7.3.4.1.3.1 URL

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

7.3.4.1.3.2 Threads

This optional integer element defines the number of threads which can be optionally used by execution service during the stagout process. The default value is 1. Multiplicity is zero or one.

7.3.4.1.3.3 Mandatory

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

7.3.4.1.3.4 NeededReplica

This optional integer element can be used to define the required count of the replicas on a given target storage system. The default value is 1. Multiplicity is zero or one.

7.3.4.1.3.5 DataIndexingService

This optional URL element contains the location of the data indexing service. Multiplicity is zero or more. There is no default value of this element. It is up to a user to make sure the computing service is able communicate to the given data indexing service.

7.3.4.1.4 KeepData

This optional boolean element specify wether the file required to be keep in the session directory after successful stage out. Multiplicity is zero or one. The default value is false.

7.3.4.1.5 IsExecutable

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

7.3.4.1.6 DownloadToCache

This optional boolean value specifies if the file is to be staged in to the cache instead of the session directory. Multiplicity is zero or one. The default value is false.

7.3.4.2 Directory

This is an optional complex element which defines stage in or stage out request of an entire directory. Multiplicity is zero or more. The ordering of the Target and Source sub-elements not significant. The directory data staging element MUST contain at least one Source or Target.
Separate element for directory required because the handling at the computing element side of the directory may differs from the files. The involved data transfer operations are always assumed to be recursive.
Currently the Directory element has the same set of sub-elements as the File element.

7.3.5 JobMetaData

This is an optional complex element which describe the context in which the job description was generated and provide some further instruction how the specified requirements can be interpretered.
Comment: new.

7.3.5.1 Author

This optional string element may contain the identifier of the creator of the job description. Multiplicity zero or one. There is no default value of this element.

7.3.5.2 DocumentExpiration

This optional date element contains a date and time when the job request became irrelevant. Multiplicity is zero or one.There is no default value of this element.

7.3.5.3 Rank

This optional string element states how to rank computing elements that have already met the other requirements. Multiplicity is zero or one. There is no default value of this element.

7.3.5.4 FuzzyRank

This optional boolean element enables fuzzyness in the ranking computation. Multiplicity is zero of one. The default value is false.

8 Security Considerations

Security considerations are important in the context of an execution service. But the intention of this section is rather to provide an overview and thus refers to other PGI work in context of security that provide much more detail.

Security Kick-Start Information
We assume that typically an information service is queried before contacting the AGU execution service in order to retrieve the correct security setup. The security experts refer to this as the 'chicken-and-egg problem of initially contacting a resource' since also information systems may run with different security setups. However, suitable GLUE2 instance with service capabilities that provide (security) information about the execution system might be as follows:
<Capability>security.authentication.pgi.gsi</Capability>
<Capability>security.authentication.pgi.openssl</Capability>

Authentication
We assume that each service should provide authentication in terms of using transport level security (TLS) and thus checks whether the certificate used to establish a connection is valid, not revoked, and trusted.
We refer to the PGI authentication specification that currently consists of three approaches how authentication can be achieved in production Grids today. The different adoptions can be exposed by each service via the GLUE2 instance of it provided by a suitable information service.

Authorization
We assume that each service should provide authorization capabilities and in the context of PGI we even raised the requirement for attribute-based authorization.
However, in this specification, we only work with security credentials that do not touch the encoding of these attributes (i.e. SAML assertions, Attribute-Certificates) and rather refer to the two approaches mentioned in the PGI authorization specification.

Delegation for Data-Staging
Although delegation is rather a security topic that should be handled outside of this specification, there is one particular use case that raise the demand to establish a suitable technique for delegation without the need for implementing the Grid Security Infrastructure (GSI). This use case refers to the different delegated credentials often used in data-stagings encoded within AGU JSDL documents. Examples are GridFTP transfers and SRM access as part of a computational job.
While typical delegation of service invocations is described elsewhere, this specification covers the the availability of a delegation technique (i.e. delegation port-Type) explicitly usable for data-staging, but that could be used for other purposes as well.
As a sideremark, this delegated credentials should also provide attributes in order to enforce the attribute-based authorization of data transfers or data access (cp. above).

9 References

[1] Strawman of the Geneva grid job Execution Service (GES): http://forge.gridforum.org/sf/go/doc15590

10 Appendix

This section will include
- the AGU-JSDL schema
- an example AGU-JSDL document
  - A work-in progress AGU-JSDL example can be found at:
- the WSDL of the AGU-ES
- an example resource description document with activity elements according to the XML rendering of the GLUE2 model