EMI Execution Service Status

Luigi Zangrando

INFN-PD

On behalf of the EMI-ES team

Outline

- Some background: EMI-ES
- EMI Execution Service
  - Modules and Port-types
  - Delegation
  - Data staging model
  - Activity state model
  - Job description
- Status and plans

Background: EMI-ES

- ARC, UNICORE and gLite are European middleware providers born in different contexts
- They now coexist within the EMI project
  - keywords: interoperability, cooperation
- They provide mostly the same functionality in terms of job management but have been designed and implemented using different and incompatible approaches, often referring to proprietary solutions (e.g. different Web Service interfaces, authZ mechanisms, data-staging models, etc)

Background: EMI-ES

- authoritative specifications (namely JSDL and OGSA-BES) for job description and job management exist
- unfortunately they are not really suited for production because they lack significant capabilities such as:
  - lack of standard security profile in relation to OGSA-BES
  - missing functionality with respect to data staging for computation
  - missing attributes in job description (e.g. for data requirement, to support epilogue/prologue and resubmission, to menage runtime environments, etc)

Background: EMI-ES

- Common specification adopted by ARC, UNICORE and gLite is needed
- This guarantee an high level of interoperability since a single job description language and a common job management interface will allow users to easily access and use different implementations of job management services using the same mechanisms.

Background: EMI-ES

- EMI-Compute is addressing this issue in parallel in two front-lines:
  - within the **EMI JRA1-WG** where **EMI Execution Service** (**EMI-ES**) also known as AGU (ARC, gLite, Unicore) specification is being defined
    - task force composed of 10 key persons from ARC, gLite and Unicore has been formed
    - activities, events, documents in the twiki page
      - https://twiki.cern.ch/twiki/bin/view/EMI/EmiExecutionService
  - within the **OGF PGI-WG** where, besides the EMI partners, some other international Grid community partners are involved

EMI-ES specification

- The EMI-ES specification represents the EMI agreement around a production execution service
  - https://twiki.cern.ch/twiki/pub/EMI/EmiExecutionService/EMI-ES-Specification_draft.odt
- This document covers the following items:
  - Interface to manage jobs (Web-service interface)
  - Job description (“EMI-JSDL”)
  - Job related information
  - Resource related information
  - Data staging
  - Delegation (needed to implement data staging)
- it is a continuation of the work previously know as the GENEVA, then AGU, then PGI execution service.
  - as a starting point, the v0.42 of the “PGI Execution Service Specification” (doc15839) was used.

Basic assumptions and scope

- Targets: Computing Element (CE) service which accepts and manages single computational jobs (Activities) running on some back-end
  - Cluster managed by an LRMS
  - “Standalone” compute node
- Out of scope (at least for now)
  - Brokering / forwarding of jobs
  - job collections
  - job parametric
  - DAG

Modules and Port-types

- The EMI-ES consists of two modules:
  - **Activity Factory** creates activities and manages user credentials and resource (CE) information (GLUE2)
    - Create port-type: CreateActivity
    - ResourceInfo port-type: GetResourceInfo, QueryResourceInfo
    - Delegation port-type: InitDelegation, PutDelegation, GetDelegationInfo
  - **Activity Manager** manages activities, user credentials and GLUE2 activity related information
    - ActivityManagement port-type: PauseActivity, ResumeActivity, CancelActivity, WipeActivity, RestartActivity (optional), GetActivityStatus , GetActivityInfo, NotifyService
    - ActivityInfo port-type: ListActivities, GetActivityStatus, GetActivityInfo
    - Delegation port-type: InitDelegation, PutDelegation, GetDelegationInfo

Bulk operations

- Where possible, operations support sets of requests (“vector operations”)
- Operations are asynchronous
  - Avoid slow replies (network timeout)
- Example:
  - Creation of several activities (createActivity())
    - Service immediately replies with vector of IDs
    - Validation stage processed asynchronously
  - Cancellation of several activities (cancelActivity())

Creation port-type

- CreateActivity(): requests the creation of a vector of activities where each activity is described by a job description document.
  - Request: vector of Job Description documents encoded in XML (EMI-JSDL compliant)
    - Job desc. validation steps performed asynchronously
  - Response: vector of values in the same order as in the request
    - Global unique JobID assigned by the ES
    - Address of ActivityManager that MUST be contacted to manage the Activity
    - Current activity status
    - Optionally: stage-in, stage-out, session directories
    - Fault

ActivityManagement port-type

- PauseActivity(): requests to stop processing of the activity (stop whatever the ES was doing).
  - Request: vector of activityIds
  - Response: vector of values in the same order as in the req.
    - ETP (Estimated Time to Pause)
    - Fault for those activities which cannot be paused
- ResumeActivity(): resume activities stopped as result of PauseActivity operation
  - Request: vector of activityIds
  - Response: vector of values in the same order as in the req.
    - ETR (Estimated Time to Resume)
    - Fault for those activities which cannot be resumed

ActivityManagement port-type

- CancelActivity(): requests the cancellation of a set of activities (executed asynchronously)
  - Request: vector of activityIds
    - Filtering mechanisms out of scope
  - Response: vector of values in the same order as in the request
    - ETC (Estimated Time to Cancel)
    - Fault for those activities which cannot be canceled
- NotifyService(): operation used to notify the ES that the client completed an operation (useful for client data push or client data pull).
  - Request: vector of activityIds, CLIENT-DATAPULL-DONE | CLIENT-DATAPUSH-DONE
  - Response: vector of values
    - Acknowledgment for those activities the notification has been accepted
    - Fault otherwise

ActivityManagement port-type

- GetActivityStatus(): provides the status information (according to the state model) of a set of activities
  - Request: vector of activityIds
  - Response:
    - Status returned as a pair of values (ActivityId, Status)
    - Fault for those activities where the status cannot be returned
- GetActivityInfo(): provides the activity information (according to the GLUE2 model) of a set of activities
  - Request:
    - vector of activityIds
    - optionally a vector of GLUE2 attributes which are requested
  - Response:
    - information returned as a pair of values (ActivityId, GLUE2 attributes)
    - Fault for those activities where the activity info cannot be returned

ActivityInfo port-type

- ListActivities(): allows the user to retrieve the list of her Activities (as allowed by the access control) handled by the ES
  - Request: accepts the following NOT mandatory parameters useful for filtering (none parameters => returns the full list of ActivityIds):
    - FromDate, toDate: the activity creation window
    - StatusList: the list of possible states
    - Limit: the max number of ActivityIds within the returned list
  - Response:
    - Returns a list of ActivityIds or an appropriate fault if some input parameter is illegal
    - An additional boolean flag “truncated” indicates that the result list was truncated
- GetActivityStatus(): as described above
- GetActivityInfo(): as described above

Delegation port-type

- Allows to delegate trust to the service, avoiding GSI
  - Currently limited to X.509 (RFC3820) proxies certificates
  - SAML assertions as a future option

- InitDelegation(): client asks for a X.509 Certificate Signing Request (CSR) from the server
  - Request:
    - The required credential type (e.g. X.509)
    - The optional renewalId (for renewing a previous delegation)
  - Response:
    - CSR + delegationID
    - Fault in case of an internal service error

Delegation port-type

- PutDelegation(): client puts the signed CSR + delegationID to the server
  - Request:
    - The credential type (e.g. X.509)
    - The delegationID
    - The RFC3280 style proxy certificate signed by the client
  - Response:
    - SUCCESS string
    - Fault in case of some error
- GetDelegationInfo(): used to get information about a previously created delegation (e.g lifetime)
  - Request:
    - The delegationID
  - Response:
    - Some info containing at least the lifetime of the proxy
    - Fault in case of some error

Data staging

- The data staging entities are just files (not collection of files)
- data staging can be done before or after job execution
  - job itself is free to do data staging during its execution
- Data staging capabilities published by the ES as part of resource information (GLUE2)
- Three directories per Activity, available at various stages in the processing
  - stage-in / stage-out / session directories

Data staging directories

- stage-in directory
  - the place offered to clients to upload data
  - the place to which the ES may pull input data
  - single dir per activity and created by the ES (createActivity())
  - possibly accessible by multiple protocols
- stage-out directory:
  - the place where the output data are collected and can be retrieved by the client
  - the place to which the ES may push output data
  - single dir per activity and created by the ES (createActivity())
  - possibly accessible by multiple protocols
- session directory:
  - the directory on the WN where the job is executed
  - the provision of access to the end user to this directory is optional

Data staging models

- stage-in scenarios
  - **Server data pull**: the ES pulls the needed data from the sources specified in the Job desc doc and makes them available later in the session directory
    - These data MAY be first uploaded into the stage-in directory
    - The delegation support is required
  - **Client data push**: the client uploads the data into the stage-in directory.
    - The ES is informed about the staging by a flag into the Job desc
    - The data to be pushed MAY be declared in the job desc
    - When done with data push, the client MUST explicitly tells the service to continue processing the activity by invoking the NotifyService()

Data staging models

- stage-out scenarios
  - **Server data push**: the ES pushes the relevant data to the targets specified in the Job desc doc
    - The delegation support is required
  - **Client data pull**: the client pulls the data from the stage-out directory.
    - The ES is informed about the staging by a flag into the Job desc
    - The data to be pulled MAY be declared in the job desc
    - When done with data pull, the client MUST explicitly invoke the NotifyService()

Activity State model

- It specifies the possible states of activities and the possible transitions between states
- Primary states + state attributes
  - An activity can only be in one state but it can have multiple state attributes
  - Main states grouped in 5 “phases”
    - ACCEPTED
    - PREPROCESSING
    - PROCESSING
      - PROCESSING-ACCEPTING
      - PROCESSING-QUEUED
      - PROCESSING-RUNNING
    - POSTPROCESSING
    - TERMINAL

State attributes

- Indicate some transient condition, or convey additional information about the primary state
- Examples in “Preprocessing”
  - Client-stagein-possible
  - Client-paused
  - Server-stagein
  - Server-paused

Job description

- Model + XML rendering
- Elements
  - Job metadata (job name and such)
  - Resources (job requirements)
    - OperatingSystem / Platform / NetworkInfo / ...
  - Application
    - Executable + its attributes
    - Standard Input / output / Error
    - Environment (OS environment variables)
    - RemoteLogging (URL for a logging service)
    - ExpirationTime / WipeTime
    - Notification (e.g. status changes by e-mail)
  - Data staging (describes all files to be transferred in/out)
  - Runtime environments
    - the job “ecosystem” (default executable, env vars, paths, ...)

EUROPEAN MIDDLEWARE INITIATIVE

EMI-ES specification presented at PGI session at OGF30 (Oct 26 2010, Brussels)

by Bernd Schuller

Nov 22-24, 2010

Status and plans

- First version (EMI Milestone) due Nov. 2010
  - specification document and xsd/wsdl
- Prototype implementations including clients in 2011 by all the providers
- Finalization of such implementations in the second year of EMI project (milestone MJRA1.3, planned for PM18)
- feedback loop

Summary

- EMI ES effort will provide a joint web service interface to ARC, gLite, UNICORE compute services
- Understood and agreed (sort of)
  - Basic modules and port types
  - Processing model
  - Data staging
- Work in progress
  - Job description and its XML rendering
- Agreement needed within EMI
  - Delegation

EMI-ES team

- Massimo Sgaravatto, Eric Frizziero, Luigi Zangrando (gLite)
- Martin Skou Andersen, Aleksander Konstantinov, Balazs Konya, Oxana Smirnova (ARC)
- Shahbaz Memon, Shiraz Memon, Bernd Schuller (UNICORE)

- https://twiki.cern.ch/twiki/bin/view/EMI/WebHome

Thank you!

EMI is partially funded by the European Commission under Grant Agreement RI-261611