УДК 004.75, 004.42

EMI EXECUTION SERVICE — THE KEY TO PROVIDE INTEROPERABILITY OF GRID INFRASTRUCTURES

V.V. YURCHENKO, A.O. LITOVCHENKO

Implementation of the EMI Execution Service (ES) specification will ensure interoperability in Grid infrastructures, including the NGI-UA, since a single set of specifications and a common job management interface will be used. This paper presents the results of the study of applying of the EMI-ES specification to provide interoperability in the Ukrainian national grid. During the research next features were studied: the structure of the EMI-ES specification, developed on the basis of the analysis of computing services in industrial environments by Working Group OGF Production Grid Infrastructure (PGI), functionality in data staging, presented by elements of stage-in (file download), and stage-out (extraction of results). The modular structure of the EMI-ES, which allows launching cross submissions between different middleware and computing elements, is represented. The execution order of the ES submission command is reviewed and shown schematically. The implementation of the EMI-ES service in middleware ARC is described. Next steps in development of the EMI Execution Service (ES) are also described.

INTRODUCTION

In 2010 in order to develop and standardize middleware software four major middleware providers: gLite, ARC, UNICORE and dCache have joined efforts in a joint project. This project was named EMI (European Middleware Initiative). EMI supports broad scientific experiments and initiatives, such as the Worldwide LHC Computing Grid (for the Large Hadron Collider).

One of the main objectives of EMI is to create common approaches and implement developed standards to provide a unified approach to the implementation of compute, information and other services for various middleware. The term Unified Middleware Distribution (UMD) was introduced to denote an integrated distribution of gLite, UNICORE and ARC, dCache.

The specification of the EMI (ES) was developed by Working Group OGF Production Grid Infrastructure (PGI) according to results of the analysis of computing services in the industrial grid infrastructures. Results of research of the ES are presented in this article [1].

There are eleven major grid sites in Ukraine, five of which are under the management of gLite middleware, and others use ARC middleware. The problem of their interaction occurs even within a single infrastructure.

Implementation of the EMI-ES specification will provide interoperability in grid infrastructures, including Ukrainian national grid, as a single set of specifications and a common job management interface will be used. The ES was presented for the first time in EMI 2 in May 2012. New version was released in EMI 3 (28 February 2013) [1].

The subject of this article is to study the implementation of the EMI-ES specification for gLite, ARC. Main directions of research are to examine new modules and the principles of their interaction and to identify key functional solutions for implementation of the ES service.

THE STRUCTURE OF THE SPECIFICATION

It is stated, that the EMI-ES is a web service and has WSDL description. The ES acts as an interface to create and manage tasks (activities). Description of the tasks should be performed in the ADL (Activity Description Language) format, dialect of XML. It allows determining a necessary application, a type of the requested resource, transferring parameters of the job, identifying input and output files, settings, data staging (the control of the data placement) and other attributes of the QoS (Quality of Service).

The extraction of information about resources and tasks is performed by the ES command line interface or using XPATH, XQUERY, SQL query languages. Information about the resources and tasks is provided according to the GLUE2 specification.

The transfer of delegations token is provided by the mechanisms of GSI — X.509 proxy and SAML (Security Assertion Markup Language) (as a part of the exchange of SOAP messages) [2].

Powerful functionality in data management is represented by the following elements:

- downloading files (stage-in) with the support of the mechanisms of server data pull and client data push;
- retrieving results (stage-out) using server data push, client data pull or access to a session directory during execution phase via GridFTP or other data transfer tools.

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

- Server data pull: the ES pulls the needed data from the specified (in the activity description document) sources and makes them available later in the session directory. These data may be first uploaded into the stage-in directory. This «server data pull» scenario requires delegation support, since the server has to act on behalf of the activity owner. Server data pull takes place in the preprocessing or processing-running state. The server-stagein attribute is used to report about this server-initiated data transfer.
- Client data push: the client uploads the data into the stage-in directory. The activity description must contain a flag informing the server that the client wishes to push data (attribute ClientDataPush of the DataStaging element). When done with data push, the client explicitly tells the server to continue processing the activity via the NotifyService operation.

The data to be pushed may be declared in the activity description. In this case, client implementations must stage-in all the declared files. Data can be pushed when the stage-in directory has been created, and the activity is in a state with the client-stagein-possible attribute set.

For what concerns stage-out, that is the staging of output data from the execution service done after job execution, there are two possible scenarios [3]:

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

• Client data pull: the client pulls the data from the stage-out directory of the ES. The downloading must take place in states with the client-stageout-possible state attribute set. The data to be pulled must be declared in the activity description.

FUNTIONAL DESCRIPTION

Due to a modular structure, the EMI-ES allows launching cross submissions between different middleware and computing elements (Fig. 1).

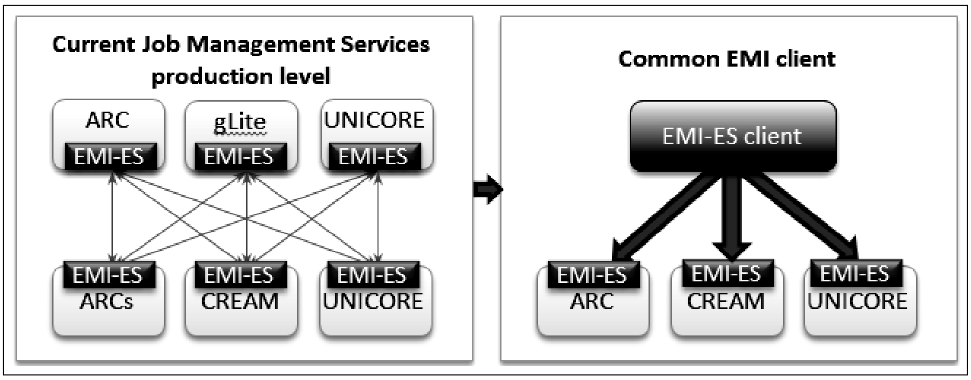

Fig. 1. Functional schema of EMI-ES

EMI-ES consists of five main modules (Fig. 2). These modules offer different functionalities embodied as independent port-types, and can be grouped and offered via independent services, usually on the same machine, eventually running separately on different machines.

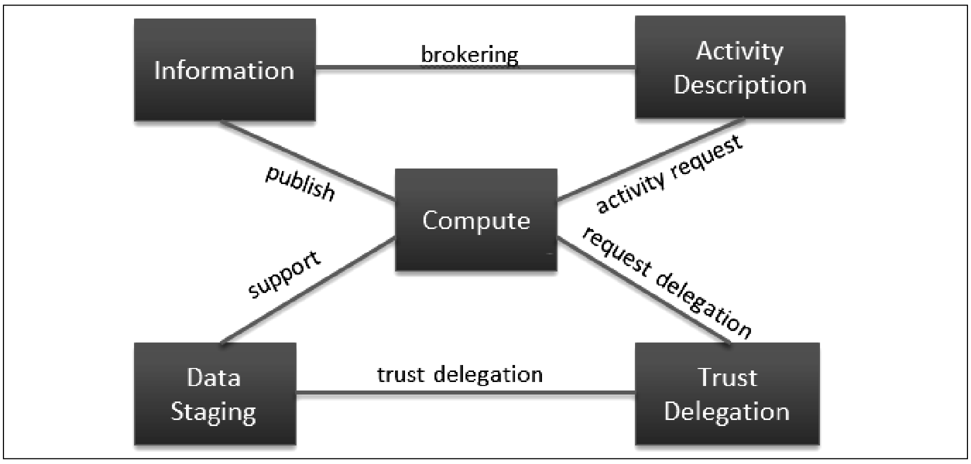

Fig. 2. The relationships between main ES modules

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

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

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

It must be stressed that ResourceInfo port-type refers only to information related to the Computing Element and it does not contain information about activities. Activities information can be retrieved using ActivityInfo port-type [3].

The execution order of ES submission command consists of next steps (Fig. 3):

1. Checking proxy by X509 mechanisms, VOMS.
2. Loading parser and analysis of the job description, which set in a common ADL format.
3. Getting information about the ES service of computing element via SOAP messages based on the GLUE2 specification.
4. Performing CreateActivity action on the computing element.
5. Sending a job description file and input files (Stage-In) to the computing element by using gridftp utilities.

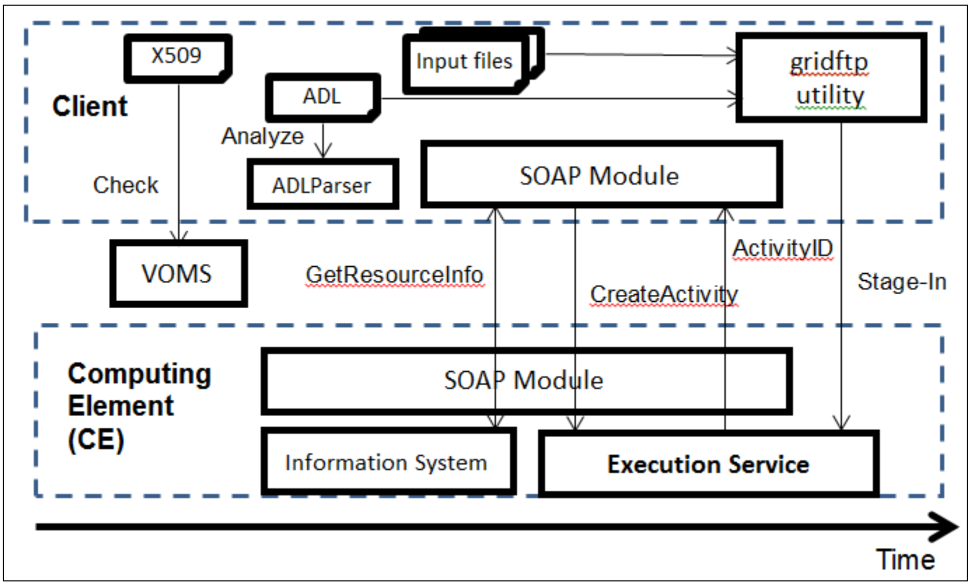

Fig. 3. The execution order of ES submission command

EMI-ES IN ARC

EMI-ES interface is implemented as a part of A-REX production service.

Service is split into modules by functionality (Fig. 4):

- interface;
- job management;

- data staging;
- credentials delegation.
- batch system communication.

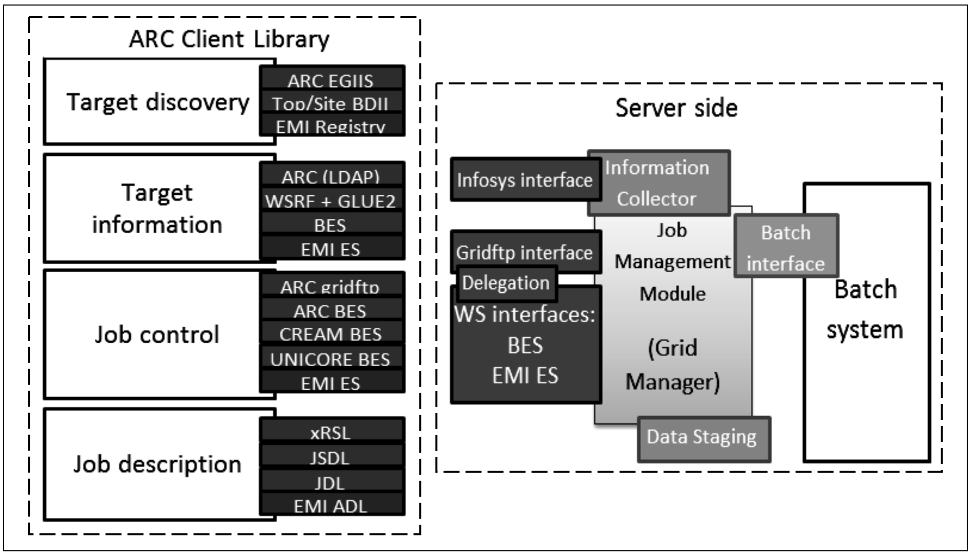

Fig. 4. Implementation of EMI-ES in ARC

Modular design allows having multiple interfaces simultaneously.

Result of implementation on the client side — C++ library with a set of dynamically loadable plugins:

- for indexing services;
- for computing services' information (based on GLUE2 model);
- for job control;
- for job description languages (based on EMI ADL) [3].

FUTURE DESIGNS

A further step in the development of ES service is to provide a common EMI client to replace three basic middleware. This client will use a single job description language and a common interface for job management, and will enable users to access various services management tasks with a single mechanism [1].

Despite the fact, that work on the implementation of the ES Specification in gLite client of EMI 3 was suspended, we have a good reason to believe, that the common client will be based on existing results.

Efforts are also focused on standardization of queries, reports, and schemas of the interaction of computing elements, which is a basis for the implementation of solutions offered by the EMI-ES.

OBTAINED RESULTS AND CONCLUSION

During the research we explored principles of construction and operation of the EMI-ES service and discovered opportunities for interoperability of Grid resources with different software architecture.

Current versions of software that supports the ES specification were installed, including a client and server-side distributions of ARC and gLite middleware.

Several changes in the client side of ARC were made. Standard ADL and JDL parsers are not able to process description files right, so ARC broker can't perform successful matchmaking against CREAM-CE target. Some attributes (QueueName, BatchSystem and InputSandbox) were hardcoded in order to make a job description compatible with CREAM. This allowed making a successful submit to CE with the CREAM interface.

Since these attributes are unique to each grid site, this solution is ad-hoc. In order to implement obtained results, ARC client should be recompiled according to the known instructions.

Our solution can already be implemented in ARC middleware, and since the ES specification is no longer amended, it allows us to hope that our results will be demanded in further development of EMI ES.

REFERENCES

1. *EMI*-Compute. — http://www.eu-emi.eu/compute.
2. *EMI* Execution Service Factsheet. — http://www.eu-emi.eu/documents/10147/31168/EMI-ES.pdf.
3. *EMI* Execution Service Specification. — https://twiki.cern.ch/twiki/pub/EMI/EmiExecutionService/ EMI-ES-Speci_cation v1.16.odt.

Received 08.07.2013

From the Editorial Board: the article corresponds completely to submitted manuscript.