Computer Science • 13 (3) 2012 http://dx.doi.org/10.7494/csci.2012.13.3.61

M. Kozlovszky
K. Karoczkai
I. Marton
A. Balasko
A. Marosi
P. Kacsuk

ENABLING GENERIC DISTRIBUTED COMPUTING INFRASTRUCTURE COMPATIBILITY FOR WORKFLOW MANAGEMENT SYSTEMS

Abstract *Solving workflow management system's Distributed Computing Infrastructure (DCI) incompatibility and their workflow interoperability issues are very challenging and complex tasks. Workflow management systems (and therefore their workflows, workflow developers and also their end-users) are bounded tightly to some limited number of supported DCIs, and efforts required to allow additional DCI support. In this paper we are specifying a concept how to enable generic DCI compatibility for grid workflow management systems (such as ASKALON, MOTEUR, gUSE/WS-PGRADE, etc.) on job and indirectly on workflow level. To enable DCI compatibility among the different workflow management systems we have developed the DCI Bridge software solution. In this paper we will describe its internal architecture, provide usage scenarios to show how the developed service resolve the DCI interoperability issues between various middleware types. The generic DCI Bridge service enables the execution of jobs onto the existing major DCI platforms (such as Service Grids (Globus Toolkit 2 and 4, gLite, ARC, UNICORE), Desktop Grids, Web services, or even cloud based DCIs).*

Keywords workflow management system, infrastructure interoperability, Distributed Computing Infrastructure, DCI, DCI Bridge

1. Introduction

For the sake of clarity we would like to provide here firstly our truly simplified workflow and workflow management system definitions: A workflow is composed by connecting multiple tasks according to their dependencies. Workflows are frequently used by research communities. Workflow management systems that control and supervise the execution of workflows are used for a wide range of scientific applications. We can categorized these solutions from many aspects, some are client based (e.g.: Taverna workbench [7], UNICORE Rich Client [3]), others are centralized (P-GRADE, WS-PGRADE/gUSE). Existing workflow management systems are also different in middleware support, workflow engines and workflow description languages. They interpret, execute and manage workflows differently since they have been defined by different scientific or software developer communities. In most cases workflow management systems (and therefore their workflows) are bounded tightly to some small number of specific Distributed Computing Infrastructure (DCIs), and efforts required to allow additional DCI support. As a result, solving workflow management system's DCI incompatibility, or their interoperability [11] issues are very challenging and complex tasks. In this paper we are specifying a generic concept how to enable generic DCI compatibility, which is proved to be feasible for many major grid workflow management systems (such as ASKALON [4], MOTEUR [6], gUSE/WS-PGRADE [8]) on job (and indirectly on workflow) level. To enable DCI compatibility among the different workflow management systems we have developed the DCI Bridge, which become one of the main components of the so called fine-grained interoperability approach (FGI) developed by the SHIWA (SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs) project [2]. In this paper we will target the generic DCI Bridge service component and describe its internal architecture, provide usage scenarios and show how the DCI Bridge can resolve the DCI interoperability issues between various middleware types (e.g. between gLite, ARC and UNICORE).

2. Motivation and state of the art

2.1. The scientific problem

Heterogeneous infrastructure is used to solve computational or data intensive problems. This heterogeneity of such systems is coming from different sources such as different hardware infrastructure size or different access methods. Middleware solutions are able to unify heterogeneous infrastructure in an effective way, however the existing different middleware solutions are not compatible with each other. These different middleware solutions are building up non-compatible, island like infrastructures and locking in (indirectly) all the workflow management systems, the workflow developers, and the end-users. By Farkas Z. et al. "Generic Grid-Grid Bridge" (3G Bridge) [5] was introduced to solve P-GRADE portal's compatibility issues between the different Grid infrastructures; between Service Grids (SG) and Desktop Grids (DG). The P-GRADE Portal solution supports parameter sweep job submission to Globus and

gLite based Service Grids only. However, for compute-intensive parameter sweep jobs the usage of Desktop Grids are more ideal than Service Grids since they are less expensive. In order to forward parameter sweep jobs to Desktop Grids the 3G Bridge service was developed in the frame of the EDGeS project. Later on this solution was further developed to support some Cloud infrastructures as well [9]. In parallel we have tried another way to resolve generic distributed computing infrastructure (DCI) interoperability issues. This developed solution is called DCI Bridge. At first sight 3G Bridge and DCI Bridge seems to provide similar functionalities, however the major difference between to two solutions are the following:

- they are using different internal architecture (DCI Bridge is Java based, and 3G Bridge is not),
- their middleware support is different: 3G Bridge is focusing on Grid infrastructures, DCI Bridge is capable to utilize Grids additionally also ARC and UNICORE, clusters and clouds,
- their load balancing capabilities are different: 3G Bridge is monolithic, DCI Bridge is distributed, multilayered, and service oriented. For more detailed comparison of the solutions one can refer to [9] [5].

2.2. FGI vs. CGI

The SHIWA project [2] has defined two different approaches to provide workflow interoperability among the different workflow management systems and workflow languages. The first solution called Coarse-Grained Interoperability /CGI/. It allows arbitrary workflow systems to invoke another workflow system as a distributed service and treats the foreign workflow as an embedded black-box type job. The second solution called Fine-Grained Interoperability (FGI) defines a community-driven workflow representation (IWIR-Interoperable Workflow Intermediate Representation [12]) that allows a workflow created in one system, to be converted or translated to the representation understand by other workflow engines. At enactment time, a workflow engine can translate the common representation into its own native format for the purposes of execution. FGI supports the distributed modification or editing of third party internal workflow components as well. Other important key elements of the fine-grain interoperability approach are the following services: DCI Bridge, Proxy Manager, Repository, Translator service (shown in Fig. 1.). Actually the fine-grained interoperability solution – in parallel with DCI interoperability – focuses more on the transformation of workflow representations in order to achieve workflows migration from one system to another. In this paper we will skip the workflow transformation part and focus only on DCI interoperability.

2.3. gUSE and WS-PGRADE

The Grid User Support Environment (gUSE) is basically a virtualization environment providing large set of high-level DCI services (including workflow manager, storage, broker, etc.) by which interoperation among classical service and desktop

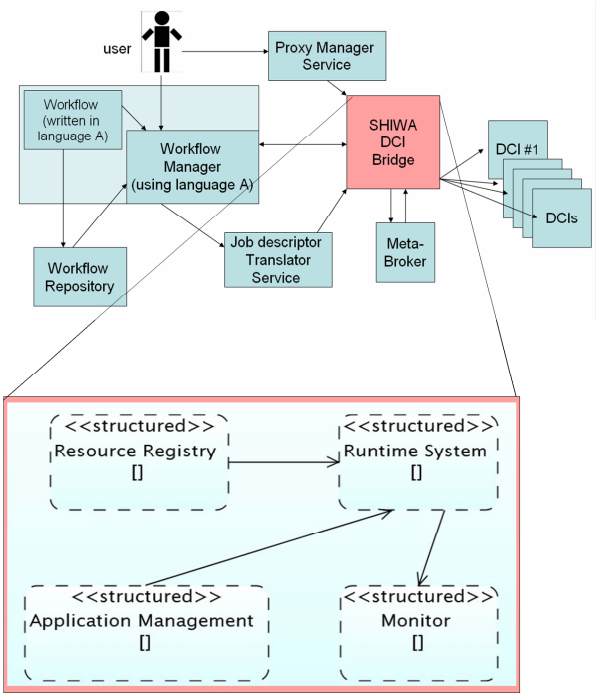

Figure 1. Components of the Fine Grain Interoperability (FGI).

grids, clouds and clusters, unique web services and user communities can be achieved in a scalable way. gUSE has a graphical user interface, which is called WS-PGRADE. From the v3.3 release of gUSE the WS-PGRADE portal is part of the enterprise open source Liferay [1] portal solution. gUSE is implemented as a set of Web services that bind together in flexible ways on demand to deliver user services and provide access to various Distributed Computing Infrastructure (DCI). WS-PGRADE hides the communication protocols and sequences behind JSR286 compliant (Liferay compatible) portlets and uses the client APIs of gUSE services to turn user requests into sequences of gUSE specific Web service calls. End users can access WS-PGRADE via Web browsers. WS-PGRADE/gUSE is used worldwide by many scientific communities, and numerous eScience gateways based on gUSE. In our performance test environment we have used the WS-PGRADE/gUSE workflow management system to create workflows and submit the jobs to the DCI Bridge.

3. The DCI Bridge – the solution

Originally gUSE had an internal core service to handle the different DCIs, at beginning only gLite and GT2 was supported. The DCI Bridge was derived from this core service component to support SHIWA's FGI solution, however later on it turned

out that it is useful for all other OGSA Basic Execution Service 1.0 (BES) enabled workflow management systems to solve their DCI interoperability issues. The DCI Bridge is a web service based application, which provides standard access to various distributed computing infrastructure (DCIs) such as: grids, desktop grids, clusters, clouds and service based computational resources (it connects through its DCI plug-ins to the external DCI resources). The main advantage of using the DCI Bridge as web application component of workflow management systems is, that it enables workflow management systems to access various DCIs using the same well defined communication interface (shown in Fig. 2.). When a user submits a workflow, its job components can be submitted transparently into the various DCI systems using the OGSA Basic Execution Service 1.0 (BES) interface. As a result, the access protocol and all the technical details of the various DCI systems are totally hidden behind the BES interface. The standardized job description language of BES is JSDL.

Additionally, DCI Bridge grants access to a MetaBroker service called GMBS [10]. This service acts as a broker among different types of DCI: upon user request selects an adequate DCI (and depending on the DCI, an execution resource as well) for executing the user's job. Just like the DCI Bridge, GMBS accepts JSDL job descriptions, and makes use of the DCI Bridge service to actually run the job on the selected DCI.

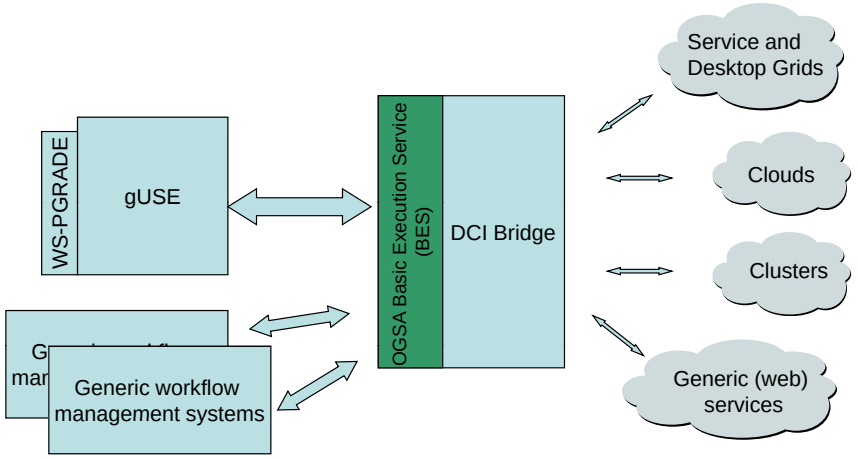

Figure 2. Schematic overview of the DCI Bridge and its external communication channels.

3.1. Internal architecture and main components of the DCI Bridge

DCI Bridge is based on four main components: the Resource Registry, the Application Management, the Runtime System, and the Monitor component. All components of the DCI Bridge can run within a generic web container (such as Tomcat or Glassfish).

- **Resource registry** The Resource Registry subsystem provides an online configuration interface to configure the accessible DCI. Also it provides information

about the configured resources to other external software components. Main components of this subsystem are:

– Online configuration interface
– ResourceConfiguration service

Wide range of different middleware types are supported by the DCI Bridge (shown in Fig. 3.). The number of supported DCI is growing constantly. So far the following DCIs are supported: service grids (gLite, GT2, GT4, ARC, UNICORE), clusters (PBS, LSF), web services, BOINC, Google App Engine, GEMLCA, local resources.

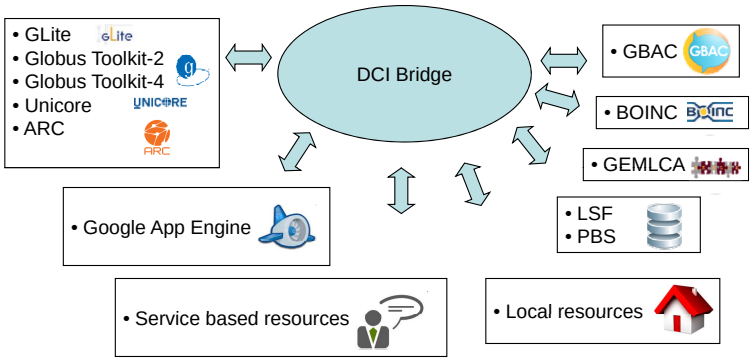

Figure 3. Supported DCIs by the DCI Bridge.

The authentication mechanism of the common portal container is used by all online graphical user interfaces, thus also the configuration interface is accessible by the same user database. To visualize the available resources, the Resource-Configuration service provides both HTTP and HTTPS communication channels for the users. This service gives details on the configuration of different grid middleware supported by the DCI Bridge service. This information is propagated by the DCI Bridge web application also back to the workflow management system's user interface (WS-PGRADE), and displayed in the Resource portlet. Fig. 4. shows a screenshot of the Resource portlet displaying information about a DCI Bridge service connected to a number of DCIs.

- **Application management** The Application Management subsystem is the implementation of the BES-Management Port-type from the 5th volume of the OGSA Basic Execution Service 1.0 specification which makes possible to supervise the software based access of the BES Factory service. Main components of this subsystem are:

  – BESManagement + online web interface

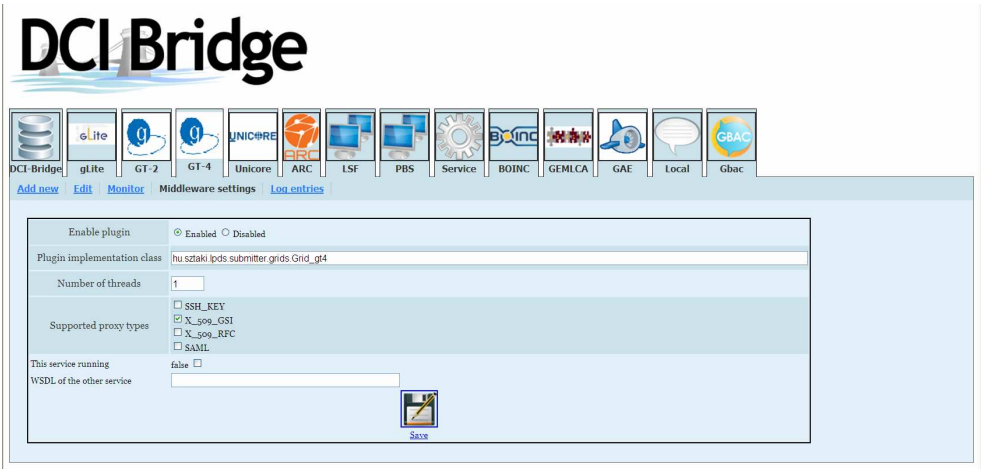

Figure 4. GUI of the Resource Configuration service.

- **Monitor** The Monitor subsystem handles and visualizes the logs and messages of the DCI Bridge, the plug-ins and the running jobs. Main components of this subsystem are:
  - Application monitor + online interface
  - Plug-in monitor + online interface
  - Job monitor + online interface
- **Runtime system** The Runtime System does the actual job running. The subsystem can be called with a service made by OGF which implements the BES WSDL and it makes the operations defined by the OGSA Basic Execution Service 1.0 specification on different grid/cloud/service based middleware. The separate running systems can be handled with plug-ins and their numbers can be increased without any restriction. Main components of this subsystem are:
  - BESFactoryService service
  - Job registry
  - Proxy manager
  - Executor layer handler
  - Input queue
  - Meta Broker client
  - DCI Plug-in manager /Middleware plug-in and queue handler/
  - Output queue
  - Status manager

  The overview of the Runtime System is shown in Fig. 5.

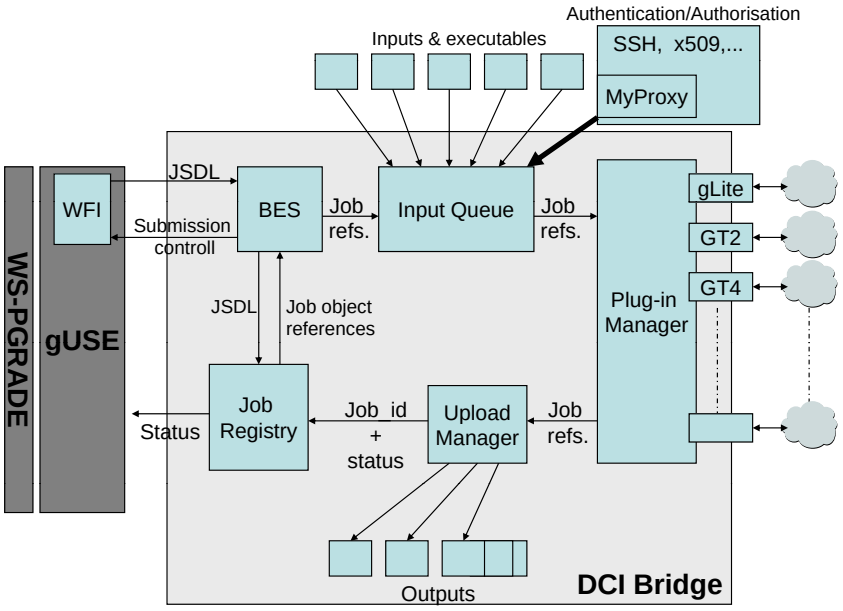

Figure 5. Internal architecture of the DCI Bridge.

The Runtime System accepts standardized JSDL job description documents which are based on well-defined XML scheme, and contains information about the job inputs, binaries, runtime settings and output locations. The core JSDL itself is not powerful enough to fit all needs, but fortunately it has a number of extensions. For example, DCI Bridge makes use of the JSDL-POSIX extension. Beside the JSDL-POSIX extension, DCI Bridge makes use of two legacy extensions: one for defining execution resources, and one for proxy service and callback service access. The execution resource extension is needed both for the core DCI Bridge in order to define specific execution resource needs and for the Metabroker service. The proxy service extension is needed for jobs targeted to DCIs, which rely on X.509 proxy certificates for job submission. The callback service extension is needed if status change callback functionality is needed: the DCI Bridge will initiate a call to the service specified in the extension upon every job status change.

User credential handing of DCI Bridge is based on content-based approach instead of a channel-based ones. This means that user credentials (proxies or SAML assertions) are not handled by the communication channel, but are rather specified in the JSDL extension mentioned earlier. This approach allows DCI Bridge to implement varied DCI-dependent credential handling options within the different DCI plug-ins instead of relying on the capabilities of the servlet container running the DCI Bridge service. Of course it is still possible to run DCI Bridge as a secured

service (for example as a service accessible through authenticated HTTP, HTTPS or even HTTPG), but credentials used for establishing connection to the service are not passed to the destination plug-in, it solely makes use of the credentials described in the JSDL extension.

Even that different DCIs are using different middleware and access solutions and they are not compatible with each other they are providing similar services for the users. To overcome the incompatibility issue, the DCI Plug-in Manager (a plug-in based framework with a common interface for all the different DCIs) was defined and implemented shown in Fig. 6.

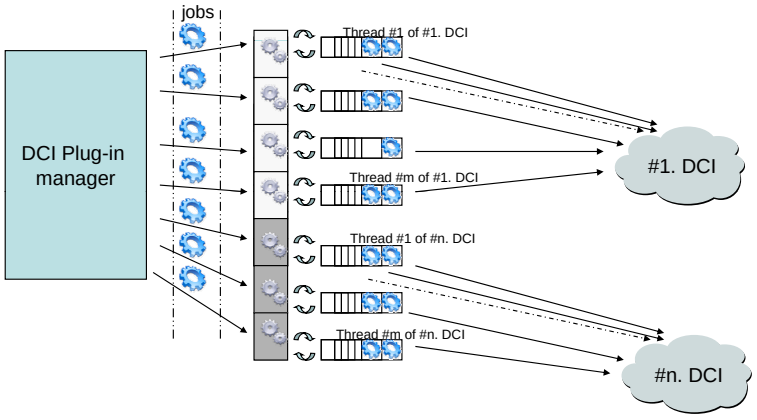

Figure 6. The multi-threaded DCI Plug-in manager's internal architecture.

Every DCI plug-in is running in a separate thread, however to increase the utilization performance of the DCI multiple plug-in threads can be launched and assigned to the same DCI entity, these threads are feeding the same DCI at the same time in a concurrent manner. DCI plug-in is a JAVA object internally, which needs to contain minimum the following functionalities:

- service-like operation, can be controlled with start/stop as a service (it is a stand-alone thread),
- optionally can contain a reference to another (external) object/resource in a form of WSDL,
- it has a plug-in-queue,
- it has a proxy type (e.g.: X509 GSI, X509 RFC, SAML, etc.) with reference to its implementation java class.

3.2. Job submission through the DCI Bridge

To show the DCI Bridge in operation, we are assuming, that the end user can submit a workflow through the WS-PGRADE GUI. Internally in the gUSE the workflow

nodes are parsed and submitted as individual jobs one-by-one to the DCI Bridge. Let's assume the user configured the workflow node successfully to run on local resource.

- **External Job submission**
  - The WFI (inside gUSE) initiate the job submission.
  - In the WFI's submit pool (RamSubmitPoolImpl) the job is waiting, and when processed, according to its job configuration a job description XML file (JSDL based) is generated.
  - Job submission (from WFI to BES Factory service).
  - The WFI calls the BES Factory service of the DCI Bridge with the generated XML (create Activity).

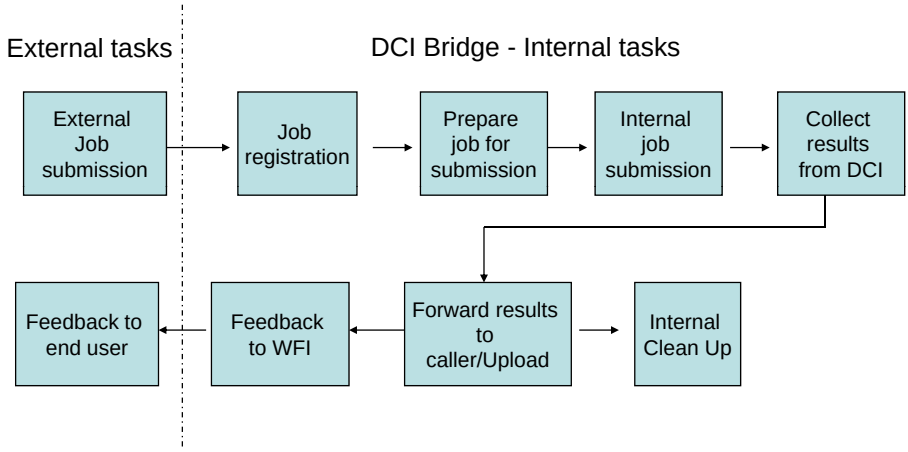

Figure 7. DCI Bridge's internal operation phases.

- **Job registration**
  - The BES Factory service receives the job and sends it into the Job Registry as object for further storage.
  - The Job Registry creates references to the job object and provides back the reference and a job ID to the BES Factory service.
- **Prepare job for submission**
  - BES Factory service includes the job reference into the Input Queue.
  - BES Factory service sends back the job ID and some status information to the WFI.
  - The job references in the Input Queue are waiting for their processing. When the job processing started an internal job directory is created. The executable(s), and all the inputs of the job are downloaded from the gUSE storage into the newly created directory. All the job directories are stored lo-

cally under the same temporally directory path. The used name convention is simple: the full (unique) job ID is the directory name.

– In parallel to the download process:

∗ Metabroker service can be utilized (if any DCI decision is required). This is an optional step in the procedure.

∗ Certificate assignment (using proxy certificates), and all other authentication/authorization related tasks are taking place during this step.

- **Internal job submission**
  - If the job is ready to run, the job ID is forwarded to the DCI Plug-in Manager.
  - The DCI Plug-in Manager is using its own queue to store the pending job IDs. Each DCI can be utilized by multiple DCI plug-in threads. Each thread is trying to process the assigned queues and submit jobs into the appropriate DCI. In our example the job is using local resources. In local submission the submit returns and status query starts automatically. The job can finish with successful/failed status.
- **Collect results from DCI**
  - When the job finished, outputs are downloaded from grid (in local submission, the output is just copied between different directories)
- **Forward Result to caller/Upload**
  - The job reference is transferred into the queue of the Upload Manager. The job references in the Output Queue are waiting for their processing. When the job processing started the job outputs are uploaded to the storage.
- **Feedback to WFI**
  - The job status (e.g.: finished/failed) is forwarded back through the Job Registry to the WFI.
- **Internal Clean Up**
  - The remains of the job processing (directory, generated files, outputs) is cleaned up, the temporally job directory and the job references in the Job Registry are deleted.

4. Performance measurements

We have evaluated the performance of the most important DCI Bridge component: the BES Factory. Our test environment consists of a test application written in JAVA and the DCI Bridge itself. The test application is able to send thousands of jobs in high frequency to the BES Factory service to measure its job handling capacity. During our submission performance test we have sent 9x1000 configured single job with predefined amount of inputs into the DCI Bridge. According to our assumptions the amount of input influences the performance parameter, because every input should

be retrieved by the Input Queue before submitting the job into the targeted DCI. We have launched our tests with zero, 10, 20, 30, 40, 50, 100, 500 and 1000 inputs. All the other job and input parameters have been similarly configured. After each job submissions the BES Factory tries to return back immediately and parallel behind the scenes it forwards the job to the appropriate DCI plug-in. The starting phase of the DCI Bridge services (due to the network topology and network services like DNS) requires longer processing time, so we have manually eliminated this transient period from our performance results. As it can be seen in Fig. 8. (where the x axis is not linear), BESFactory service scales smoothly, thus the increased number of inputs cause only about linear processing time increase.

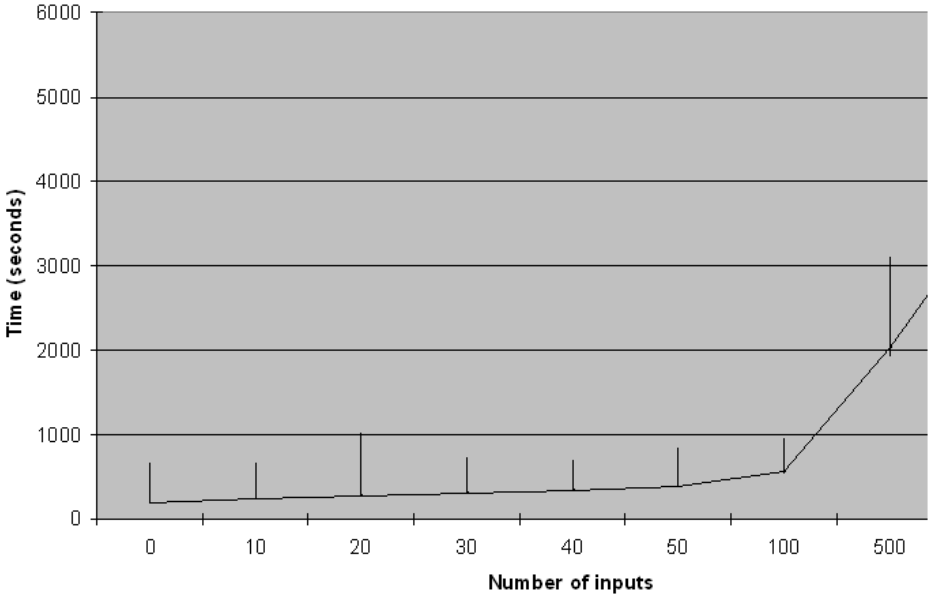

Figure 8. BESFactory service performance test results.

5. DCI Bridge usage scenarios

5.1. User's default motivations

Within Europe large number of DCI middlewares are existing in parallel (just to name a few examples in service Grids: gLite, ARC, UNICORE, GT2, GT4, GT5). Interoperability is playing crucial role in Big Data challenges, and non-interoperable DCIs (with their proprietary job submission, authentication, etc.) are preventing the seamless mobility of the users/researchers and the reusability of their scientific workflows both directly and indirectly. Indirectly preventing, because the workflow management systems are usually supporting only limited number of DCIs (1 or 2)

and workflow management systems are not interoperable with each other at all. The SHIWA project has been identified different use cases where DCI interoperability acts as important factor (shown in Table. 1).

Beside the native usage scenario when the workflow system (such as WS-PGRADE/gUSE) is connected directly to a DCI Bridge, which is connected to all the targeted DCIs, we can identify other usage scenarios as well.

Table 1

User motivations to resolve DCI interoperability issues.

Use case	Description	Added value
Workflow porting	Importing/exporting an existing workflow from/to a foreign environment	Reuse of workflows in another virtual research community.
Multi-DCi execution	Generation of Metaworkflows using different DCIs	Reuse of workflows. Scale up to larger infrastructure (Data challenge).
Multi-enactor/multi-DCI execution	Generation of Metaworkflows using different DCIs and/or workflow enactors	Reuse of workflows. Scale up to larger infrastructure (Data challenge). Combine different workflows as embedded jobs into a superworkflow.

5.2. DCI Bridge in a multi-grid multi-node installation scenario

DCI Bridge is capable to be deployed on multiple user interface nodes in order to enable access to different grid systems. Single workflow management system (WS-PGRADE/gUSE) installation can make use of a number of DCI Bridge services in the same time parallel and transparently (shown in Fig. 9.). In such case a "master" DCI Bridge is connected to the WS-PGRADE/gUSE installation. The master DCI Bridge is using its own BES Factory client to connect to the "slave" DCI Bridges. The different DCI Bridge deployments are running on different DCI user interface machines connected to the different DCIs. In such a setup, users of a WS-PGRADE installation may make use of very different DCIs through a unified user interface. This multi-node installation scenario can be useful if we would like to relieve the load of the DCI Bridge server, or the DCI UIs are incompatible with each other. In such case the "master" DCI Bridge is the single distribution point for all the submitted jobs. It assigns automatically a unique job ID to each job (this ID remains intact and unique till the job exists), and forwards further to the "slave" DCI Bridges. The

job status information and all the job outputs are not handled by the "master" DCI Bridge, this information directly transferred back to the original submitter (generic workflow manager system, WS-PGRADE/gUSE, etc.).

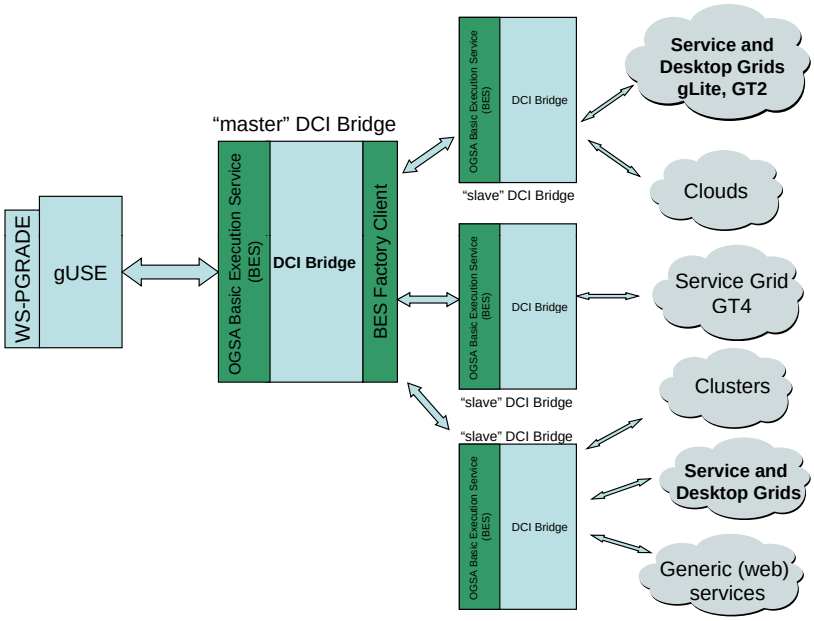

Figure 9. Multi-node installation of the DCI Bridge service.

5.3. DCI Bridge in a load-balancing scenario

In this scenario the core gUSE services are aware of one DCI Bridge installation. Although this DCI Bridge service is not connected to any DCI, it may forward the jobs it receives to other DCI Bridge deployments as they are using the same submission interface and job description language. This way the central DCI Bridge service may distribute the incoming jobs among the other services it is aware of. After the jobs are distributed, they have the possibility to report job statuses back to the central gUSE services (to be specific, the WFI) using the callback JSDL extension we have described earlier. DCI Bridge load balancing can be realized mainly in two different ways (external and internal Load Balancing Server /LBS/). Some of the server solutions (such as GlassFish) has inbuilt LBS service, which can provide external load balancing for the DCI Bridge service. Because this solution is not able to support fully the job submission (the job abort cannot initiated) we are implementing an internal LBS inside the DCI Bridge.

5.4. DCI Bridge as a cloud-deployed service

In this scenario one central DCI Bridge installation is forwarding its requests to some load-balancing service. The task of this load-balancing service is to forward incoming requests to a DCI Bridge installation deployed within the cloud. Using this setup, a big number of cloud resources can be exposed. The big advantage of this approach is, that constant VM instance startups can be eliminated, thus existing DCI Bridge deployments within the cloud using the local submission plug-in are theoretically available for use immediately.

6. Conclusions and future work

In this paper we have specified the DCI Bridge, which is an implementation of a generic concept how to enable DCI interoperability. The created solution is proved to be feasible for many major grid workflow management systems (such as ASKALON, MOTEUR, gUSE/WS-PGRADE and others) on job level and indirectly also on workflow level. We have successfully created a common service platform compatible with almost all major Grid (service and desktop grids) and some of the cloud and cluster infrastructures. Our solution not only resolves DCI interoperability, but with its API it simplifies the development of multi-DCI capable workflow management systems. The usage of the DCI Bridge can significantly foster the establishment of DCI/middleware independent eScience gateways. The created DCI Bridge API based on BES, which simplifies and standardize the generic communication between workflow management systems and DCIs. Additionally we have extended the communication API with some more functionality to support all the extra feature sets of the underlying DCIs. These functionalities are optional and merely used only for extra services. The integration work was difficult and took a long time for us. The development lasted almost 5 years, and firstly this solution was inner part of the gUSE system. As during these years we have developed connectors for almost all existing major Grid middlewares (version dependent implementations) we have started to do refactoring both at code and at functionality levels. During the last two years our solution matured into a standalone component, and we have finally released it as open source service at Sourceforge (before that it was an internal module of gUSE's SOA based system). This standalone service component enables other workflow management systems to benefit DCI interoperability the same standardized way as gUSE is capable to do. The plug-in like internal architecture of the DCI Bridge offers simple extension capabilities to the developers, if they need to utilize other, non-implemented DCIs. The DCI Bridge is not the ultimate solution for all the DCI related issues, and nor it tries to solve all the problems. The DCI Bridge is now providing support for authentication, job submission/management and status monitoring functionalities. A workflow provenance, storage service features are still out of the range of the DCI Bridge, such service features shall be embedded as internal components into the workflow management system (in our case we are doing this also with gUSE). However the job status information is propagated back from the DCI Bridge to the connected workflow management sys-

tem and it can be easily used for workflow provenance. DCI Bridge provides -for external usage- a generic/simplified job status list, and it is using internally its own status list for all the job handling/monitoring mechanisms. Basically all middleware specific job statuses are mapped to DCI Bridge's internal job status list. The DCI Bridge plug-in contains all mapping information. The original job statuses -received from the DCI- are propagated back to the workflow management system also as log information to enable workflow management system to react or do granular job status monitoring. This is only useful if the workflow management system is able to understand the middleware specific job status information (usually this is not a case). Storage references are handled transparently by the DCI Bridge. The workflow management system should be able to access and manage the storage infrastructure by default. DCI Bridge handles storage related information only as references and does not provide any translation between existing storage solutions yet. In this paper we have described DCI Bridge's internal architecture, provided information how its components are working together, and showed some additional usage scenarios as well. According to our tests the DCI Bridge implementation is able to resolve successfully DCI compatibility issues. The implemented DCI Bridge solution is used successfully in the SHIWA project as an internal service at the back-end part of the FGI solution to resolve the DCI interoperability issues between various middleware types (gLite, ARC and UNICORE). The DCI Bridge is used as one of the core component in the SHIWA Simulation Platform (`http://ssp.shiwa-workflow.eu/`) and internally in many other gUSE based eScience gateways. The modular, plug-in like architecture enables the DCI Bridge service to be extended easily, and during our development work we have included support for many DCI platforms (such as cluster and cloud infrastructure and web services). As future work we are planning to extend the capabilities of the DCI Bridge with additional middleware support; thus we are trying to include support of additional cloud and service grid type DCIs and provide support for full load balancing of the DCI Bridge service.

Acknowledgements

This work was supported by EU project SHIWA (SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs), which is an Integrated Infrastructure Initiative (I3) project (contract number 261585). The SHIWA project aims to leverage existing workflow solutions and enable cross-workflow and inter-workflow federative exploitation of DCI Resources by applying both a coarse- and fine-grained strategy. Full information is available at `http://www.shiwa-workflow.eu`.

References

[1] Liferay. `http://www.liferay.com/`, accessed 09.10.2011.

[2] Shiwa project. `http://www.shiwa-workflow.eu/`, accessed 09.10.2011.

[3] Demuth B., Schuller B., Holl S., Daivand J., Giesle A., Huber V., Sild S.: The unicore rich client: Facilitating the automated execution of scientific workflows. In *2010 IEEE Sixth International Conference on e-Science*, pp. 238–245, 2010.

[4] Duan R., Fahringer T., Prodan R., Qin J., Villazón A., Wieczorek M.: Real world workflow applications in the askalon grid environment. In *Proc. of EGC 2005*, pp. 454–463, 2005.

[5] Farkas Z., Kacsuk P., Balaton Z., Gombás G.: Interoperability of boinc and egee. *Future Generation Computer Systems*, 26(8):1092–1103, 2010.

[6] Glatard T., Montagnat J., Lingrand D., Pennec X.: Flexible and efficient workflow deployment of data-intensive applications on grids with moteur. *International Journal of High Performance Computing Applications*, pp. 347–360, 2008.

[7] Hull D., Wolstencroft K., Stevens R., Goble C., Pocock M., Li P., Oinn T.: Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34:729–732, 2006.

[8] Kacsuk P.: P-grade portal family for grid infrastructures. *Concurrency and Computation: Practice and Experience*, doi: 10.1002/cpe.1654:235–245, 2011.

[9] Kacsuk P., Marosi A., Kozlovszky M., Ács S., Farkas Z.: Parameter sweep job submission to clouds. In Cafaro M., Aloisio G., editors, *Computer Communications and Networks: Grids, Clouds and Virtualization*, pp. 123–141. Springer, 2010.

[10] Kertész A., Kacsuk P.: Gmbs: a new middleware service for making grids interoperable. *Future Generation Computer Systems*, 26:542–553, 2010.

[11] Krefting D., Glatard T., Korkhov V., Montagnat J., Olabarriaga S.: Enabling grid interoperability at workflow level. In *Proc. of the Grid Workflow Workshop, Köln, Germany*, 2011.

[12] Plankensteiner K., Prodan R., Fahringer T., Montagnat J., Glatard T., Hermann G., Harrison A.: Iwir specification ver. 0.3. SHIWA project deliverable, 2010.

Affiliations

M. Kozlovszky
MTA SZTAKI/Laboratory of Parallel and Distributed Computing, Budapest, Hungary,
m.kozlovszky@sztaki.hu

K. Karoczkai
MTA SZTAKI/Laboratory of Parallel and Distributed Computing, Budapest, Hungary,
[karoczka@sztaki.hu

I. Marton
MTA SZTAKI/Laboratory of Parallel and Distributed Computing, Budapest, Hungary,
imarton@sztaki.hu

A. Balasko
MTA SZTAKI/Laboratory of Parallel and Distributed Computing, Budapest, Hungary,
balasko@sztaki.hu

A. Marosi
MTA SZTAKI/Laboratory of Parallel and Distributed Computing, Budapest, Hungary,
atisu@sztaki.hu

P. Kacsuk
MTA SZTAKI/Laboratory of Parallel and Distributed Computing, Budapest, Hungary,
[kacsuk@sztaki.hu

Received: 24.12.2011
Revised: 23.03.2012
Accepted: 9.07.2012