Computer Science • 14 (2) 2013 http://dx.doi.org/10.7494/csci.2013.14.2.307

Akos Balasko
Zoltan Farkas
Peter Kacsuk

BUILDING SCIENCE GATEWAYS BY UTILIZING THE GENERIC WS-PGRADE/GUSE WORKFLOW SYSTEM

Abstract *Enabling scientists to use remote distributed infrastructures, parametrize and execute common science-domain applications transparently is actual and a highly relevant field of distributed computing. For this purpose a general solution is the concept of Science Gateways. WS-PGRADE/gUSE system offers a transparent and web-based interface to access distributed resources (grids, clusters or clouds), extended by a powerful generic purpose workflow editor and enactment system, which can be used to compose scientific applications into data-flow based workflow structures. It's a generic web-based portal solution to organize scientific applications in a workflow structure and execute them on remote computational resources. As the portal defines nodes as black-box applications uploaded by the users, it does not provide any application specific interface by default. In this paper we show what kind of tools, APIs and interfaces are available in WS-PGRADE/gUSE to customize it to have an application specific gateway.*

Keywords distributed computing, workflow systems, Science Gateway

1. Introduction

Science gateways are popular solutions to enable scientists to run commonly used scientific applications with its own parametrization. Generally, science gateways are connected to large computational and data resources to deal with long term or computation intensive applications. The gateways provide a web interface, tailored to the needs of the targeted community, and hide all complexity of the underlying subsystems. They focus on a specific scientific area by collecting its commonly used applications. According to the user's requirements, these applications can be executed independently or they can be organized in a workflow structure. The main advantages of the gateway solutions are the accessibility, the transparency, the customizable interface, and the fact that they can minimize the user knowledge necessary to run applications on distributed systems.

This paper is organized in the following way. Community requirements and a taxonomy according to them are detailed in the Introduction, then we compare the available solutions in the Related work. Section 3 is a short introduction about the WS-PGRADE/gUSE system, in Section 4 we focus its customization tools, then in Section 5 we show the results achieved and conclude the paper. Acknowledgements end the paper.

1.1. Community requirements

Members of a scientific community can be organized in two groups. Normal scientists would like to focus on their own research area, they have needs, but they don't have the time or capacity to learn techniques and processes that are completely different than their interest. They would like to use scientific applications without bothering of their local computer and caring about licences, all of complexity must be hidden from them. Team of technical Supporters have the ability to identify the needs of Scientists, and to fit needs with possible solutions. They are able to provide computational resources, scientific applications that can be executed on them, and to share composition of applications as workflows. In the followings to be able to compare the available science gateway technologies, a basic taxonomy is introduced containing the most important properties of a science gateway from the user's point of view.

1.2. Taxonomy

According to the community requirements, three different aspects are identified: sharing, because the users must know what is the benefit of using the gateway; complexity of execution, to show that gateway provides scientific applications for simple execution, or complex composition of them; and interface, which can be created building on the given gateway technology.

Sharing. Science Gateways provide access for community members to common hardware resources and/or applications. Usually, these software are pre-installed on the

infrastructure but they can come from another trusted machine (e.g. from a Repository or the Science Gateways itself). In most of the cases gateways provide access to a shared distributed, HPC, or cloud infrastructures to let the scientists run their measurements. The main advantage of sharing applications is to enable users to use the set of applications, which has an institutional license, difficult interface, or large computational requirements which makes it impossible to use them against large data on a local machine.

Complexity. An application which is going to be shared can be a simple stand-alone application, or a composition of these application organized in a workflow structure. Since managing the execution of simple applications by the gateway is much easier than enacting a complex applications' workflow structure, therefore the gateways can be separated according to the complexity of the execution management to Simple Job and Workflow.

Interface. According to their interest and taste, communities have different demands for a science gateway. In many cases communities are using a convenient and familiar interface, and – besides they would like to feel the benefits of the new shared resources and/or applications – they would not change it to learn a new one. To serve this case, gateways should provide a solution to establish their functionalities with an existing interface via a common and standardized way. In the classification we denote this case as Existing. Some communities do not need to have a completely different and specialized interface for each application, therefore a generic interface can be generated automatically or semi-automatically. This case is called Automatic. The last class defined contains gateways, which provide opportunity to customize or replace their default user interface by a completely self-developed one, but opens an API to exploit the system's functionalities. These are called Fine-tuned gateways.

2. Related work

Science gateways can be developed in two ways. We can develop the whole framework from scratch or we can use one of the available web-based portal solutions, which provide tools or APIs to create a customized interface. An example of the first case is CIG Seismology Web Portal [1] that enables seismologists to request seismograms for any given earthquake, and simulates them on the Teragrid infrastructure. Much more gateways were developed based on existing generic portal frameworks, because in this way the duration of the development can be decreased, and such gateways based on a highly supported framework are more sustainable.

As it is introduced in [2], EnginFrame is a generic cloud Portal that provides access to Applications, Data, and the HPC compute farm (grid, cluster or cloud) through a standard web browser, developed by NICE. It can be customized for specific community needs, e.g. as it is described in [3], the framework was used to create a community-used web-based interface on the top of iRods Data Grid. It is integrated with different workflow engines [6] such as Taverna [4] or Kepler [5] to enable the users

to organize their applications in complex structures. Since it does not contain a native workflow enactment system, in our taxonomy Enginframe is capable to submit jobs only.

Vine Toolkit [7] provides a general set of extensible APIs written in JAVA to ease creating web applications hiding real applications executed on distributed infrastructures for creating science gateways as is described in [8].

Another approach to build an interface that provides the basic functionality required for executing distributed applications is SAGA [9], which offers a standardized programming interface without any GUI support needed to create an application-domain science gateway. Since its just a specification, it is implemented in many programming languages such as Python, C++ or JAVA.

The Lunarc Application Portal [10] provides easy access to grid resources for commonly available applications like MATLAB. The portal provides easy to use forms for each supported application where single and multiple jobs can be created and submitted to the grid. Functions for controlling and monitoring jobs are also available.

GridSpace [11] offers a generalized interface for scientists to develop their scientific applications and to execute them on a distributed system. Beside the framework has a possibility of sharing applications (experiments in their terminology), and allows the users to define an interface specifically for adding, typing, selecting or uploading inputs for a new execution.

The DECIDE [12] science gateway is customized from the SAGA API. The science gateway of the DECIDE project helps the early diagnosis and research on brain diseases. MosGrid gateway [13], customized from the WS-PGRADE [14] portal framework, aims to support users in all fields of simulation calculations. Users can access data repositories,where information on molecular properties are stored then simulation jobs can be submitted into the Grid. Moreover, the users are supported at the analysis of their calculation results via a specific web-based interface.

Table 1 shows the evaluation of the frameworks and tools in our taxonomy. As it can be seen, all of them provide an API to create application specific interface, the only difference is the abstraction level of these APIs. As Vine Toolkit and SAGA support JSDL job submission, they can be used for existing interfaces, however, they don't contain possibilities for native workflow enactment (Kepler is connected to them). None of them has the facility to generate interface automatically, developers must create, compile and deploy interfaces by hand for each.

In this paper we are focusing on the generic purpose workflow management system WS-PGRADE portal. The SCI-BUS FP7 project aims to serve scientific communities around Europe by demand-specific science gateways based of WS-PGRADE. On this score, and as this generic purpose portal framework contains various functionalities, it is ideal to extend it with a customization tool so that each gateway could fit its own community's requirements. In this paper we show what possibilities, interfaces are available in WS-PGRADE to be able to convert it into a science gateway.

Table 1
Evaluation of frameworks against the taxonomy.

Sharing		**Execution management**		**Interface**			**Framework**
Hardware	Software	Simple Job	Workflow	Existing	Automatic	Fine-tuned	
☑	☐	☑	☐	☑	☐	☐	EnginFrame
☑	☑	☑	☐	☑	☐	☑	Vine Toolkit
☑	☑	☑	☐	☑	☐	☑	SAGA
☑	☑	☑	☑	☐	☐	☑	LUNARC Application portal
☑	☑	☑	☐☑	☑	☐	GridSpace	

3. WS-PGRADE/gUSE – general introduction

WS-PGRADE portal based on the gUSE (grid User Support Environment) service set is the second generation P-GRADE portal that introduces many advanced features both at the workflow and architecture level compared to the first generation P-GRADE portal [14]. The major lesson we learnt from the usage of the P-GRADE portal was that there are different kinds of user communities and they require different kind of user interfaces to reach various types of DCIs. Based on this lesson the whole concept of the P-GRADE portal was redesigned and a multi-tier service architecture was being developed that can support various user types and various DCIs. The Architectural tier enables access to many different kinds of DCIs through the DCI Bridge job submission service as shown in Figure 1.

The DCI Bridge is a web service based application that provides standard access to various distributed computing infrastructures (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). 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.

The Middle tier contains those high level gUSE services that enable the management, storing and execution of workflows.

In the regard of the focus of this paper we must mention that gUSE contains a component called Repository to allow users to share workflows within the portal internally. During this sharing process, users can set read-only permissions for any property of a workflow configuration, which means that others, who import the work-

flow, cannot modify them. In the terminology of gUSE this kind of workflows are called templates.

Finally, the Presentation tier provides the graphical WS-PGRADE user interface of the generic DCI gateway framework. This layer can be easily customized and extended according to the needs of the science gateway to be derived from WS-PGRADE/gUSE.

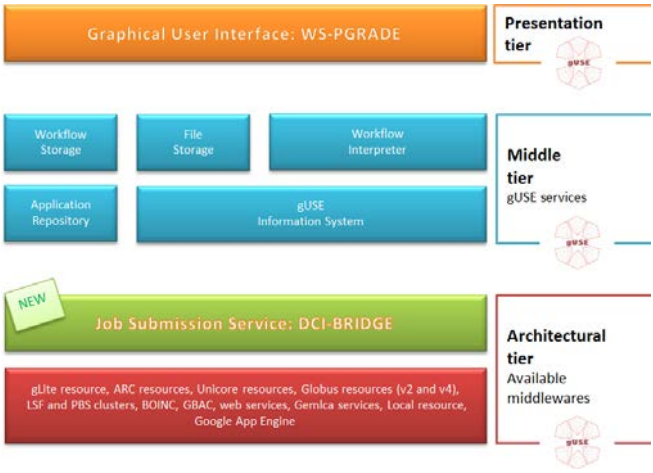

Figure 1. Multi-tier architecture of WS-PGRADE/gUSE.

WS-PGRADE/gUSE system extends the DAG-based workflow concept with advanced parameter study features through special workflow entities, (generator and collector jobs, parametric files), condition-dependent workflow execution and workflow embedding support, moreover it extends the concrete workflow concept of P-GRADE with new concepts and objects like graph (abstract workflow), workflow instance, template, application and project.

Among the WS-PGRADE/gUSE design concepts another important requirement was to enable the simultaneous handling of a very large number of jobs even in the range of millions without compromising the response time at the user interface. In order to achieve this level of concurrency of job handling the workflow management back-end of WS-PGRADE portal is implemented on the Service Oriented Architecture (SOA) concept and is supported by gUSE and with the DCI Bridge service as shown in Figure 1. gUSE is a set of services with well-defined interface protocols to realize the workflow management back-end of the WS-PGRADE portal. Since the workflow concept of WS-PGRADE is much more sophisticated than in the P-GRADE the Condor DAGMan workflow engine is replaced with a newly developed workflow engine, called as Zen, which can manage very large number of job executions. Further information about the complete system, its structure and its functioning is published in [15], while publication [16] introduces the DCI-Bridge component in detail.

Opportunities available for the system to be able to customize it to a Science Gateway are concluded and evaluated in Table 2. An integrated possibility for customization and dynamic interface generalization is called End User Interface; completely new interfaces which exploit functionalities (e.g. workflow submission) of gUSE can be developed with using Application Specific Module (ASM). Remote API is for submitting and enacting workflows from client-side. And as DCI-Bridge implements a standard BES interface, it can be invoked externally for executing a job.

Table 2
Evaluation of WS-Pgrade against the taxonomy.

Sharing		**Execution management**		**Interface**			**Framework**
Hardware	Software	Simple Job	Workflow	Existing	Automatic	Fine-tuned	
☑	☐	☑	☐	☑	☐	☐	DCI-Bridge
☑	☐	☐	☑	☑	☐	☐	Remote API
☑	☑	☐	☑	☐	☑	☐	End-User Interface
☑	☑	☐	☑	☐	☐	☑	ASM

4. Customization interfaces of WS-PGRADE/gUSE

Science gateways based on WS-PGRADE/gUSE can serve different demands, according to the community requirements about the computation power needed, the complexity of the applications to be submitted and the specificity of the interface to fit the community needs and to meet its terminology. If the community needs just a generic, but simple interface, WS-PGRADE/gUSE can be tailored automatically for each application by setting it to the End-User Mode (detailed in 4.1). If the community would like to have a completely new web-based interface, they can use an Application Specific Module to develop a portlet connected to the core system to access and use the applications defined there previously. This concept hides all default interfaces of WS-PGRADE/gUSE from the users. Subsection 4.2 details this possibility.

The next scenario is if the community has a conventional interface, but they would like to use the workflow management and execution system of WS-PGRADE/gUSE. In this case they can use Remote API detailed in 4.3. The last one, where the community would like to extend their computational possibilities with other middlewares to submit particular jobs to be executed. This scenario is covered by the standard interface of the DCI Bridge detailed in 4.4.

4.1. Generate interface for a workflow

In most cases applications do not require specific interfaces (e.g. a visualizer, or external tools for adding inputs), therefore the most common functionalities can be identified in a general case. Besides, because of the structure of the workflow all input fields are known exactly by the system, a generic interface can be created automatically to be able to modify these fields and to manage the workflow execution. This interface is called as the *End User* Interface of WS-PGRADE, which enables modifying some basic properties of already existing workflows by hiding most of the workflows' details.

In this solution users are not allowed to create new workflows, but they have access to the local repository to import already developed workflows. After importing a shared workflow, the generic End User Interface will generate an interface dynamically based on the imported workflow's different properties with read and write permission.

Figure 2 illustrates an End User interface generated for a workflow shown on its left hand side. Its first job has 5 input ports, 4 of them are set to be modifiable, therefore the interface generates four input fields for file upload (receptor.pdb, docking.pdb, docking.gpf, ligand.pdb). Besides this, and the last job requires a number as command line argument for which two extra input fields are generated (number of work units, maximum number of best results).

Figure 2. End User interface generated for AutoDock application.

As is shown in Figure 3, the auto-generated interface is an integrated feature in WS-GRADE, it can be enabled by configuration of the portal itself, and it does not require any programming skills by the developers or by the portal administrators to be able to have an application specific Science Gateway.

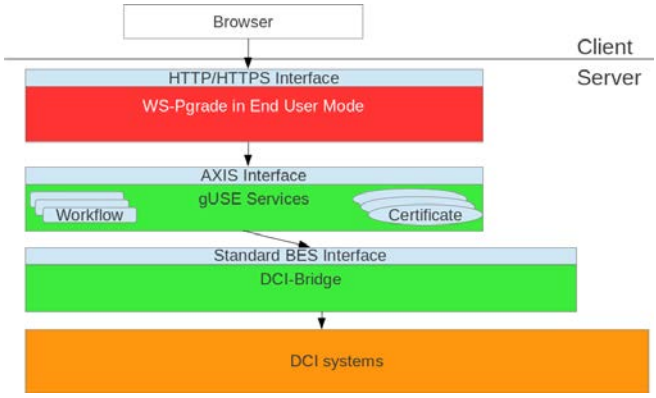

Figure 3. Structure of EndUser mode in the WS-PGrade.

4.2. Customize interface for a workflow

4.2.1. Concept

Users of WS-PGRADE define their applications as workflows. They can share their applications among each other by exporting them to the repository. Following this way, other users can import such applications and execute or modify them in their user space. Concept of Application Specific Module (ASM) that solves the problem of customization is based on this scenario. In this case two different user roles can be defined: the Application Developer, who created and shared the application, and the End Users, who import and execute it. Or analogously, Application Developers are administrators, or scientists with developer skills; and the End Users are those, who just use their product.

Using this solution development of science gateways technically means the development of web applications that produce a transparent interface, handle the interaction coming from the users, and, according to them, call the internal components. This calling mechanism is simplified by ASM as it hides complex algorithms and web-service calls and provides these functionalities as simple Java methods covering the whole life-cycle of the workflow in aspect of End Users. ASM contains a method to get the list of Application Developers, the applications shared by a particular Developer, and to import a particular workflow.

To guarantee that the workflow will do the same that the Developer wanted originally, End Users have restricted possibilities to manipulate the workflow. Especially they cannot modify the workflow structure by adding or removing a node, or they cannot replace the program placed in a node, but they can upload and attach a new input file, set or modify command line arguments, etc.

Finally, End Users can manage the workflows; they can submit them to a distributed resource, check the execution, download their outputs or delete them.

gUSE can be extended with an interface that hides the complexity of the inner abstraction levels, and inner callings of different core services. Without this component, one or more difficult web-service callings should be constructed each time when a customized portlet should get or pass information from/to the portal. In order to avoid this complexity Application Specific Module(ASM) API covers all of these internal information accesses by a simple call of well-parametrized JAVA methods.

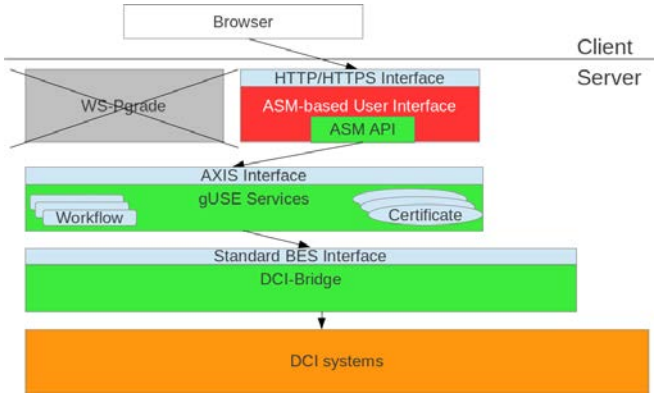

Figure 4. Concept of ASM.

4.2.2. API

The functionalities provided by ASM can be separated into three different subsets: (i) methods covering application management issues, (ii) methods that can be used for input/output manipulation and (iii) methods to handle user activities during execution such as aborting or rescuing applications. According to this classification, methods are detailed in Table 3.

Within the set of methods for application management, as it is shown in the first part of the table, there are several methods available for getting information from workflows stored in the local repository: getting a list of application developers, getting list of applications according to a specified developer id, importing an application to local user space and getting a list of applications that have already been imported.

The set of Input/Output Manipulation covers various methods, shown in the middle part on the table, to handle different input cases such as uploading a file to a specified port, setting a file that is currently exists on the portal server, or setting command-line parameters for a job. Some methods of this set contain possibilities to fetch the outputs of the calculations.

The set of Execution methods contains methods not only for workflow submission, but for many other activities like methods for getting workflow execution status in simple or in detailed format, for aborting or for rescuing a workflow. These methods are detailed in third part of the table.

Table 3. Common methods of ASM API.

Functions	Parameters	Return values	Description
getWorkflowDevelopers	type: enumeration, possible values are "Application", "Workflow", "Graph"	Array of Strings	Gets the list of application developers that have exported at least one application to the gUSE Application repository in the given type
getASMWorkflows	String userId	List of ASMWorkflow objects	Returns a list of applications has been already imported by the user identified by userId
ImportWorkflow	String userId, String newWorkflowName, String developerId, String workflowType, String workflowName	Void	Imports an application (identified by workflowName) exported by developerId, from the gUSE Application repository to space of userId. The application will be named as newWorkflowName.
DeleteWorkflow	String userId, String workflowId	Void	Removes the workflow called workflowId from space of userId.
getFiletoPortalServer	userId, workflowId, jobId, portId	Void	Downloads the file associated to the portId of jobId contained by workflowId, to the portal server.
uploadFiletoPortalServer	FileItem file, userId, fileName	Void	Uploads file to a temporary space of userId and names it fileName.
placeUploadedFile	String userId, File fileOnPortalServer, String workflowId, String jobId, String portId	Void	Associate the file that has been already uploaded to portal server, to the portId of jobId contained by workflowId.
getRemoteInputPath	String userId, String workflowId, String jobId, String portId	String	Returns the current remote input path associated to a specified job and port.
setRemoteInputPath	String userId, String workflowId, String jobId, String portId, String newRemotePath	Void	Sets remote input path to newRemotePath for portId of jobId contained by workflowId.
getRemoteOutputPath	String userId, String workflowId, String jobId, String portId	String	Returns the actual remote input path associated to a specified job/port.
setRemoteOutputPath	String userId, String workflowId, String jobId, String portId, String newRemotePath	Void	Sets the remote input path according to the string added as parameter.
getCommandLineArg	String userId, String workflowId, String jobId	String	Returns the actual command line argument set for jobId of workflowId.
setCommandLineArg	String userId, String workflowId, String jobId, String commandline	Void	Sets command line argument for jobId of workflowId.
getResource	String userId, String workflowId, String jobId	ASMResourceBean	Returns the current resource to where the job is being submitted.
setResource	String userId, String workflowId, String jobId, String type, String grid, String resource, String queue	Void	Sets a resource specified by grid, resource and queue.
getNodeNumber	String userId, String workflowId, String jobId	String	Returns the value of the required number of nodes if the given job is an MPI job.
setNodenumber	String userId, String workflowId, String jobId, int nodenumber	Void	Sets the number of the required nodes to nodenumber for jobId.
getDetails	String userId, String workflowId	WorkflowInstanceBean	Returns detailed statuses of each job of workflowId that is being submitted.
submit	String userId, String workflowId	Void	Submits application named workflowId.
rescue	String userId, String workflowId	ASMService	Rescues an application named workflowId
abort	String userId, String workflowId	ASMService	Aborts the application named workflowId

As portlets in general are deployed in portlet containers that supervise the most common user activities and manage user sessions, ASM does not have to provide any security feature with the exception of issues related to the underlying infrastructures and complex systems. To provide these, ASM uses inner-level solutions of gUSE requiring proxy certificates. These certificates will be created and used with the help of dedicated Certificate portlets by the end-users who – similarly to the members of other user groups – have account to the portal.

4.3. Using own interface to execute a workflow

4.3.1. Concept

gUSE system can be extended to let the community send a complete workflow via HTTP protocol circumventing the original WS-PGRADE interface. Technically this extension is a new web-service component called Remote API, which should be deployed to the portal server, and be registered among the general gUSE components. Detailed structure is shown on Figure 5.

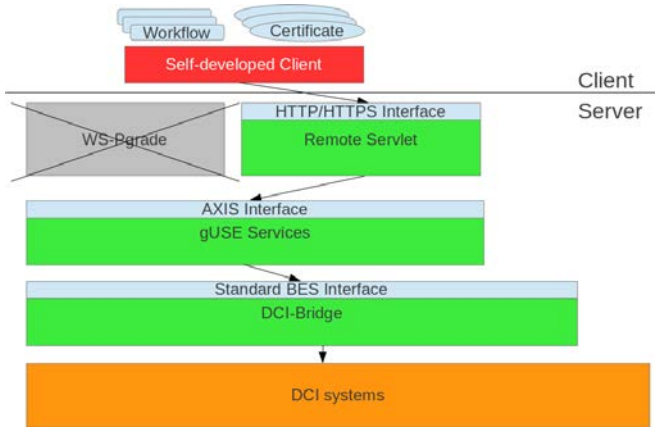

Figure 5. Structure of remote access of gUSE.

In this solution, members of a community do not have to register to the WS-PGRADE portal, namely they don't have to have a valid user account on the portal. It means that they can not develop their own workflow, they have to use workflows that were developed previously in a general way by valid portal users, and all the workflows must be available on the client machine.

If the workflow nodes are going to be executed on a DCI, which requires authentication (username and password in the case of clusters, or X509 user proxies for grids), it must be present on the client machine as well, next to the workflow (structure, binaries and its inputs).

As the users are unauthenticated in the portal's view, the API call creates a new temporary user before every execution, and the workflow will be submitted on behalf

of this temporary user. After downloading the outputs of the workflow, this temporary user and all of its joint files and database rows are erased. It shows the main disadvantage of this solution, since the users' execution are not stored on the portal, the users cannot retrieve outputs which have already been downloaded once.

As it would be really insecure to open a servlet for connection from anywhere, the clients need to know a password set previously with the server to establish the connection.

4.3.2. Provided functionalities

Remote API is implemented as a simple servlet, in the followings we describe its available functions and their parametrization.

We assume that a complete workflow including its binaries and inputs is placed on the client machine from where the execution is going to be started. In the original folder structure each job has its own folder named as the job itself containing the executable named as "executable.bin" (since all executable called like this, resolving naming conflicts is required) and inputs in the following structures: inputs/portNumber/number, where "inputs" is a static text, portNumber is the number of the given port set in the workflow structure, and the number started from zero will store the exact file, or (that's why it is numbered) a set of files. One more thing to do is to move the workflow's inputs and executables into one folder and compress them into a zip archive. Then, we need to create a file and define the concrete mappings between the files and the entities (port association or jobs' executables) in the workflow structure. It is a simple text file, which contains key/value pairs separated by newlines, in the following format:

- If the file is an input file: `inputfilename=workflowName/jobName/portNumber`.
- If the file is an executable: `exename=workflowName/jobName`.

Finally, we have to have files for authentication if the target computational resources require this, and compress them into a particular file (e.q. certs.zip).

Table 4 shows the possible functions and their parameters required from the users. Technically each call of Remote API means to call the servlet itself, but with different parametrization. Parameter "m" defines the method itself, the next one is the password agreed between the server and the client, "id" denotes the identifier of the workflow submitted, "wfdesc" means a file notation to the description of the workflow structure, "inputzip" does it with the compressed inputs, "portmapping" with the mapping file and "certs" with the compressed file of the authentication files.

To show using Remote API in practice, we use a Linux-based application called cURL, which implements HTTP protocol and capable to call servlets from console. For instance a workflow submission using cURL, if gUSE extended by Remote API is available on `http://my.own.guse.hu/remote`; we agree in the password string "password"; the workflow structure is stored in workflow.xml; inputs are compressed in inputs.zip; mappings are described in portmapping.txt; files of authentication are collected in certs.zip; looks like this:

Table 4. Functions provided by the Remote API.

Parameters	Functions						
	Submission	Status Information	Detailed Status Information	Suspend	Rescue	Abort	Download
m	submit	info	detailsinfo	suspend	rescue	stop	download
pass	☑	☑	☑	☑	☑	☑	☑
id		☑	☑	☑	☑	☑	☑
wfdesc	☑						
inputzip	☑						
portmapping	☑						
certs	?				?		
Return value	workflow identifier	Workflow Status: submitted; running; finished; error; suspended; not valid data	Workflow Status; ∀ job ∈ Nodes: job:init=A; running=B; finished=C; error=D	TRUE or FALSE	TRUE or FALSE	TRUE or FALSE	A zip file that contains the output and log files

```
curl -k -F m=submit -F pass=password -F wfdesc=@workflow.xml -F
inputzip=@inputs.zip -F portmapping=@portmapping.txt -F certs=@certs.zip
http://my.own.guse.hu/remote
```

4.4. Using own interface to submit jobs

Let us assume that our community does not need any workflow to have, therefore they do not require a complete workflow management system. However, they have a couple of single scientific applications used as frequently as they need more computational power for serving the grown load. To extend existing resource set for a single job submission, a component of the WS-PGRADE/gUSE system called DCI Bridge can be used, which provides a standard interface to submit jobs into different middlewares transparently.

4.4.1. Concept

DCI Bridge accepts standardized JSDL job description documents. These documents are based on a well-defined XML scheme containing 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.

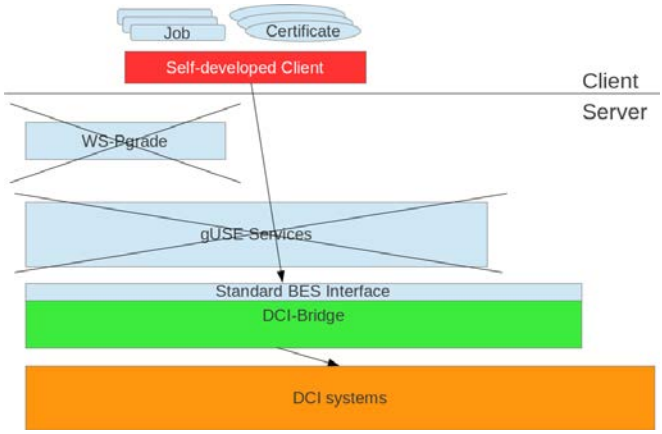

Figure 6. Structure of native access of DCI-Bridge.

Besides 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 required 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 one. 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 various DCI-dependent credential handling options within the different DCI plugins instead of relying on the capabilities of the servlet container running the DCI Bridge service.

4.4.2. Functionalities

To communicate with DCI-Bridge in a native way, we need to implement an interface to invoke DCI-Bridge service via BES interface, an other to provide certificates, and one for retrieving statuses of the jobs submitted. Sample interface implementations are detailed in the Developer Cookbook of the WS-PGRADE available on the portal's Sourceforge page [17].

5. Results and conclusions

As results we can show some available science gateway operating for different communities based on solutions developed for WS-PGRADE. Swiss Grid Proteomics gateway was developed in cooperation between ETH Zurich, SystemsX and MTA SZTAKI and its recent version is based on WS-PGRADE and the ASM concept. HP-SEE Portal [18] also uses functions of ASM and with its help it provides a science gateway for bio-informatics with an application called DiseaseGene which makes an in-silico mapping between the genes coming from the different model animals and searches for unexplored potential target genes. In an ongoing project SCI-BUS, VisIVO application uses Remote API to submit complete workflows to be executed on gLite-based resources, to create customized views of 3D renderings from astrophysical data tables from mobile devices [19, 20, 21]. Industrial partner 4DSoft uses native DCI-Bridge interface for infrastructure testing purposes. End User interface is used in AutoDock [22] portal to allow bio-scientists to run molecular docking simulations powered by the EDGeS@home volunteer desktop grid.

To demonstrate the adoptability and the flexibility of WS-PGRADE/gUSE system extended by the customizing possibilities shown in this paper, we can show some of the science gateway installations focusing on the different middlewares they are connected to. Since the core portal solution supports many middlewares, large variety of resources can be connected to a science gateway that is based on WS-PGRADE/gUSE. For instance Swiss Grid Proteomics gateway uses PBS clusters to execute applications, HP-SEE Portal serves its users by providing access to resources via ARC middleware for execution. MosGrid Science gateway [23] based on WS-PGRADE/gUSE is connected to a UNICORE-based distrubuted infrastructure. All of these connections are covered by the core portal solution, therefore the developers

of the gateway can focus on the interface itself, they do not take care of accessing the underlying infrastructure.

In this paper we presented interfaces, APIs, internal and external tools developed for WS-PGRADE to open it for customization to create a science gateway instance. We set up a basic taxonomy for comparing different science gateway tool sets and APIs, then we showed how can gUSE/WS-PGRADE serve all possible requirements by providing easy-to-use and generic interfaces to use.

Acknowledgements

The research leading to these results has partially been supported by the European Commission's Seventh Framework Programme (FP7/2007-2013) *under grant agreement no 283481* (SCI-BUS).

References

[1] `https://geodynamics.org/portals/seismo/` (accessed at 10.11.2012)

[2] Torterolo L, Porro I et al: Building Science Gateways with EnginFrame: a Life Science example. *Proc. of International Workshop on Portals for Life Sciences*, Edinburgh, United Kingdom, September 14–15, 2009.

[3] Venuti N., Torterolo L. et al: A Service-Oriented Interface to the iRODS Data Grid. *Proc. of the International Worshop on Science Gateways*, Catania, Italy, September 20–21, 2010.

[4] Hull D., Wolstencroft K. et al: Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, p. 34, 729–732, 2006.

[5] Ludäscher B., Altintas I. et al: Scientific workflow management and the Kepler system: Research Articles. *Concurrency and Computation: Practice and Experience*, 18(10): 1039–1065, 2006.

[6] Bartocci E, Cacciagrano D et al.: A Grid infrastructure for managing workflows in bioinformatics applications. *Proc. of the NETTAB2006*, Santa Margherita, pp. 38–44. 2006.

[7] Dziubecki P., Grabowski P. et al: Easy Development and Integration of Science Gateways with Vine Toolkit. *Journal of Grid Computing*, pp. 1–15, 26 October 2012.

[8] Dziubecki P. et al: Nano-Science Gateway development with Vine Toolkit and Adobe Flex. *Proc. of the International Worshop on Science Gateways*, Catania, Italy, 2010.

[9] Goodale T., Jha S. et al: SAGA: A Simple API for Grid Applications. *Computational Methods in Science and Technology*, 12(1): 7–20, 2008.

[10] Lindemann J., Sandberg G.: *An extendable GRID application portal.* Advances in Grid Computing-EGC 2005, pp. 322–325, 2004.

[11] Gubała T., Bubak M.: *GridSpace – Semantic Programming Environment for the Grid*. Parallel Processing and Applied Mathematics. vol. 3911, p. 172–179, Springer Berlin Heidelberg, 2006.

[12] Ardizzone V., Barbera R. et al: *The DECIDE Science Gateway*. Journal of Grid Computing, pp. 1–19, 2011.

[13] S. Gesing, I. Márton, et al.: *Workflow Interoperability in a Grid Portal for Molecular Simulations*. Proc. of the International Worshop on Science Gateways, Catania, Italy, September 20–21, 2010

[14] Kacsuk P.: *P-GRADE portal family for Grid infrastructures*. Concurrency and Computation: Practice and Experience, vol. 23, issue 3, pp. 235–245, 2012.

[15] Kacsuk P., Farkas Z. et al.: *WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities*. Journal of Grid Computing, vol. 9, no. 4, pp. 479–499, 2012.

[16] Kozlovszky M., Karoczkai K. et al: *Enabling generic distributed computing infrastructure compatibility for workflow management systems*. Computer Science, vol. 13, pp. 61–79, 2012.

[17] gUSE portal package and source code: `http://sourceforge.net/projects/guse/` (accessed at 19.08.2012)

[18] HP-SEE Portal: `http://ls-hpsee.nik.uni-obuda.hu:8080/liferay-portal-6.0.5` (accessed at 19.08.2012)

[19] Becciani U., Costa A. et al: *VisIVO: A Library and Integrated Tools for Large Astrophysical Dataset Exploration*. Astronomical Data Analysis Software and Systems vol. 461, pp. 505, 2012.

[20] Massimino P., Bandieramonte M. et al: *Learning Astrophysics through Mobile Gaming*. Astronomical data analysis software and systems XXII, Astronomical Society of the pacific, 2012

[21] Costa A., Becciani U. et al: *VisIVOWeb: A WWW Environment for Large-Scale Astrophysical Visualization*. Publications of the Astronomical Society of the Pacific, vol. 123, issue 902, pp. 503–513, 2011.

[22] Autodock Portal: `https://autodock-portal.sztaki.hu/` (accessed at 10.11.2012)

[23] Gesing, S., Herres-Pawlis, S. et.al.: *A Science Gateway Getting Ready for Serving the International Molecular Simulation Community*. Proc. of the EGI Community Forum 2012 / EMI Second Technical Conference, Munich, Germany, 26–30 March, 2012.

Affiliations

Akos Balasko
Laboratory of Distributed and Parallel Systems, MTA SZTAKI, Budapest, Hungary,
`balasko@sztaki.mta.hu`

Zoltan Farkas
Laboratory of Distributed and Parallel Systems, MTA SZTAKI, Budapest, Hungary,
zfarkas@sztaki.mta.hu

Peter Kacsuk
Centre for Parallel Computing, University of Westminster, London, United Kingdom,
kacsuk@sztaki.mta.hu

Received: 20.11.2012
Revised: 9.01.2013
Accepted: 11.02.2013