A general and scalable solution for heterogeneous workflow invocation and nesting

Tamas Kukla[1]
Tamas Kiss[1]
Gabor Terstyanszky[1]
Peter Kacsuk[2]

[1] School of Informatics, University of Westminster
[2] Laboratory of the Parallel and Distributed Systems, Computer and Automation Research Institute, Hungarian Academy of Sciences

A General and Scalable Solution for Heterogeneous Workflow Invocation and Nesting

Tamas Kukla, Tamas Kiss, Gabor Terstyanszky
Centre for Parallel Computing, School of Informatics
University of Westminster, 115 New Cavendish Street
London W1W 6UAW, United Kingdom
Email: T.Kukla@student.westminster.ac.uk
T.Kiss@westminster.ac.uk
G.Z.Terstyanszky@westminster.ac.uk

Peter Kacsuk
Laboratory of the Parallel and Distributed Systems
Computer and Automation Research Institute
Hungarian Academy of Sciences
1111 Kende u. 13, Budapest, Hungary
Email: kacsuk@sztaki.hu

***Abstract*—Several widely utilized, Grid workflow management systems emerged in the last decade. These systems were developed by different scientific communities for various purposes. Enhancing these systems with the capability of invoking and nesting the workflows of other systems within their native workflows makes these communities to be able to carry out cross-organizational experiments and share non-native workflows. The novel solution described in this paper allows the integration of different workflow engines and makes them accessible for workflow systems in order to achieve this goal. The solution is based on an application repository and submitter, which exposes different workflow engines and executes them using the computational resources of the Grid. In contrast with other approaches, our solution is scalable in terms of both number of workflows and amount of data, easily extendable in the sense that the integration of a new workflow engine does not require code re-engineering, and general, since it can be adopted by numerous workflow systems.**

I. Introduction

Workflow allows e-Scientists to express their experimental processes in a structured way and provides a glue to integrate their remote applications. Several Grid workflow management systems, such as Triana [1], P-GRADE [2], Taverna [3], Kepler [4], CppWfMS [5], YAWL [6], or the K-Wf Grid [7], were developed for different scientific purposes. Making those systems interoperable in order to help e-Scientists to carry out inter-organizational experiments and reuse workflows that were developed in a workflow management system they are not familiar with raises a few fairly complex issues. In order to achieve cross-organizational collaboration between these scientific communities, workflows should be able to interoperate, communicate with and/or invoke each other during execution.

The WfMC (Workflow Management Coalition) defines *workflow interoperability* in general in [8] as: "The ability for two or more Workflow Engines to communicate and work together to coordinate work." In this definition the workflow engine is a piece of software that provides the workflow run-time environment. Sharing workflows is also a natural desire of e-Scientists, since it speeds up the design phase, improves the quality by allowing the usage of already validated workflows. Many projects and research groups are working in this field. The Workflow Management Research Group [9] of the OGF (Open Grid Forum) is focusing on workflow sharing and interoperability [10]. The Workflow Management Coalition [11] tries to decrease the risks of using business process management and workflow products via interoperability standards. The CppWfMS project is aiming to achieve workflow language interoperability.

Since workflow management systems were developed for various purposes, they differ in several aspects. Most systems are coupled with one workflow engine. Taverna uses Freefluo, Triana uses Triana Engine, K-Wf Grid uses GWES [7], older versions of P-GRADE used Condor DAGMan [12], while its recent version uses its own engine called Xen. Many workflow systems use dissimilar workflow description languages. While Triana is able to interpret BPEL (Business Process Execution Language), its own defined language and additional workflow formats (since its workflow interpreter is extendable), most systems are restricted to one language. Taverna workflows are represented in Scufl, older versions of P-GRADE used Condor DAG, now it uses its own defined format, Kepler uses MoML, YAWL system uses YAWL language, while K-WfGrid uses GWorkflowDL [13]. Because of the diversity of workflow languages, e-Scientists are unable to reuse workflows within their accustomed workflow management system if the workflow in question, was created in another system.

Workflow description languages can be based on various workflow formalisms. Some workflow languages, such as the Condor DAG, use simple directed acyclic graph (DAG) workflow structure that does not allow the usage of loop, recursion or nested workflow. However Scufl, that is also a DAG based language, is extended with control constraints supporting the usage of if/else, case and loop structures within Taverna workflows. The new version of the P-GRADE portal also uses a DAG based language, which is extended with recursion and workflow nesting. YAWL language and GWorkflowDL are based on Petri Nets, while BPEL, BPML (Business Process Modeling Language) and XLANG (XML based Language defined by Microsoft) are Pi-Calculus based. Both Petri Nets and Pi-Calculus have a wider range of expression capabilities, for instance they allow the concept of non-determinism.

Because of these differences, it is not a trivial issue to

express a workflow of one type in the description language of another. For instance, a Petri Net based workflow cannot always be converted into a DAG based language, since DAG cannot express iteration or non-deterministic choice.

Furthermore, most workflow management systems are restricted to use one (Grid) middleware. Hence, it might be a problem to reuse nodes of a workflow that was created in another workflow management system, because the node might not run on other middleware. Fault tolerance and monitoring policies are various as well, but this may not cause serious obstacles to achieve interoperability. However, dissimilar data representations and data resources used by different workflow management systems might cause data incompatibility, which also has to be resolved.

Since most workflow management systems cannot share workflows, workflow components and/or workflow data with each other, it is a slightly complex issue to attain the interoperability of their workflows at any level.

This paper describes a solution, which allows the integration of various workflow engines and makes them accessible for Grid based workflow systems in order to attain interoperability between them. In contrast with other approaches, our solution is scalable in terms of both number of workflows and amount of data, extendable in the sense that the integration of a new workflow engine to the system does not require code re-engineering, only user level understanding of the engine in question, and general, since it can be adopted to any Grid workflow system.

The rest of this paper is structured as follows. Section II describes possible solutions that may bring workflow interoperability into effect. Section III specifies the requirements of a generic and scalable workflow engine integration approach. Section IV introduces a general solution and its reference implementation, which fulfils these requirements. Section V presents a case study workflow, that, as a high-level, heterogeneous workflow, encapsulates workflows of three different workflow systems. Finally, Section VI concludes our discussion and highlights our future plans.

II. Approaches to workflow interoperability

Several projects are aiming to attain interoperability between workflows of different systems. This section describes the solutions that can bring workflow interoperability into effect.

Workflow description language *standardization* would make workflow management systems to be able to exchange their workflows. This would realize interoperability via workflow sharing by defining a standard workflow description language and convincing existing workflow management system developers and users to use this standard. This is rather unlikely to happen in the near future.

If such a standard format will be defined and accepted, workflow management systems will either adopt this format or define import/export processes for *workflow translation* [14]. Assuming that various workflows are available from different workflow repositories, this solution would allow e-Scientists to reuse workflows created in different workflow management systems and execute them using the system which they are already familiar with. XPDL (XML Process Definition Language) was defined by the WfMC for this purpose, however it did not gain universal acceptance so far.

YAWL [15] (Yet another workflow language) is based on an extensive analysis of (more than 30) existing workflow systems using a set of workflow patterns described in [16]. Because of its expressive power and formal semantics, YAWL might be a candidate to be used as an intermediate language for workflow translations. See, for instance, BPEL to YAWL translation described in [17].

The CppWfMS workflow system, that was developed by CNAF department of the National Institute of Nuclear Physics in Italy, is able to abstract from the workflow description language, as it is described in [18]. This system defines interfaces for workflow description translation to achieve workflow language interoperability. The CppWfMS system contains a JDL (Job Description Language, that is able to describe simple jobs as well as DAG based workflows) to GWorkflowDL converter and also a Scufl to GWorkflowDL converter, that transforms simple Scufl workflows using XSLT (Extensible Stylesheet Language Transformations). However, because of the different expression capabilities of the workflow languages, it is not always possible to translate one language to another.

Since standardization is not likely to happen in the near future and workflow translation is not always possible, an alternative approach to attain workflow interoperability could be realized by *workflow engine integration*. This makes a workflow management system to be able to execute non-native workflows by integrating different workflow engines into one system. Hence, these engines can be invoked when a non-native workflow is to be executed. The system will not be able to interpret these workflows; therefore, users will not be able to modify them. The basic idea behind this approach is that by executing a workflow in its native environment (by its own workflow engine), significantly reduces the complexity of the problem, although it does not provide workflow manipulation capabilities only workflow execution.

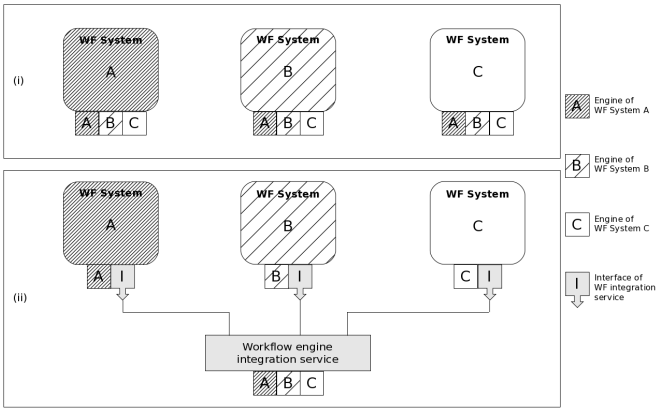

Fig. 1. (i) Tightly and (ii) loosely coupled workflow engine integration

Non-native workflow engines can be integrated to a workflow system in either (i) tightly or (ii) loosely coupled fashion. See Figure 1. *Tightly coupled integration* means that the non-native engines will run at the same site as the workflow system, while *loosely coupled approach* allows access to these engines via services and the engines are executed remotely to the workflow system. In the case of loosely coupled integration Web/Grid services could ensure the communication for workflow execution. Two different approaches to this interoperability level are described in [19], that deals with interoperability between different workflow engines and provides a solution for high level heterogeneous workflow design within the SIMDAT project. The first approach, that is also described in [20], is to wrap each workflow as a Web/Grid service and create a client which is able to invoke the workflow service. The client can be used to invoke the workflow from any workflow management system. However publishing workflows as web services and creating clients for those services makes the whole process complicated, so an automated mechanism is needed to ease workflow publishing for the users. The second approach to this interoperability level is to wrap the functionalities of the different workflow engines as Web/Grid services and create clients for workflow execution. The client passes the workflow written in the appropriate workflow language and the input data to the workflow engine that will execute it and give back the results to the client. This client can be used in any workflow management system for workflow execution.

Many further projects are aiming to solve workflow interoperability by integrating workflow engines. The CppWfMS defines interfaces not only for workflow description translation, as it was mentioned above, but for workflow engine integration as well. The architecture of the YAWL system also provides a solution for workflow engine integration via the YAWL interoperability broker, which provides an interface for workflow engine execution. Finally, the VLE-WFBus [21] system, developed by the Dutch Virtual Laboratory for e-Science project, provides a meta workflow system, that encapsulates a few popular workflow engines and allows the composition of high-level heterogeneous workflows via a Vergil based GUI provided by the Ptolemy project [22].

In contrast with the above described workflow engine integration solutions we are aiming to realize a workflow engine integration approach that: (a) provides a generic solution, which can be adopted to any Grid workflow system, (b) is scalable in the sense of both number of workflows and amount of data, and (c) the integration of a new workflow engine to the system should not require code re-engineering, only user level understanding of the engine in question. Next section describes the requirements of such an approach.

III. Requirements of workflow engine integration

Our goal is to provide a solution for workflow sharing and interoperability between Grid workflow systems by integrating different workflow engines. The integration enables workflow systems to execute non-native workflows. When a workflow system needs to execute a non-native workflow, the system does not even have to be able to interpret that workflow description or know how the workflow will be executed. For the native system, these workflows are black boxes with some certain input and/or output data.

A workflow consists of *nodes* (which are usually jobs, service calls or human interactions), *data*, and *data connections*. Nodes process input data and produce output data that can be connected to the input of the next node. The inputs and outputs of a node are called *ports*.

Workflow engine integration is able to realize two types of workflow interoperability: (i) non-native workflow nesting (nested sub-process model of interoperability) and (ii) non-native workflow invocation (chained model of interoperability). See Figure 2. Non-native workflow nesting means synchronous workflow execution, where the nested Workflow is represented as a node of the native workflow. The execution of such a workflow happens when the representing node is to be executed, since at this point all the input parameters of the workflow are available. When the execution of the non-native workflow finished, the workflow outputs will be ready to be transferred to the sites where the following nodes in the workflow graph will further process them. Hence, the execution of embedded workflows should be scheduled in the same way as they were ordinary nodes.

Non-native workflow invocation means asynchronous workflow execution, where the non-native workflow is invoked by a node of the native workflow. The invoker node should transfer the data that is required for the execution and start the enactment. Once the execution of the invoked workflow begun, there is no further interest in it.

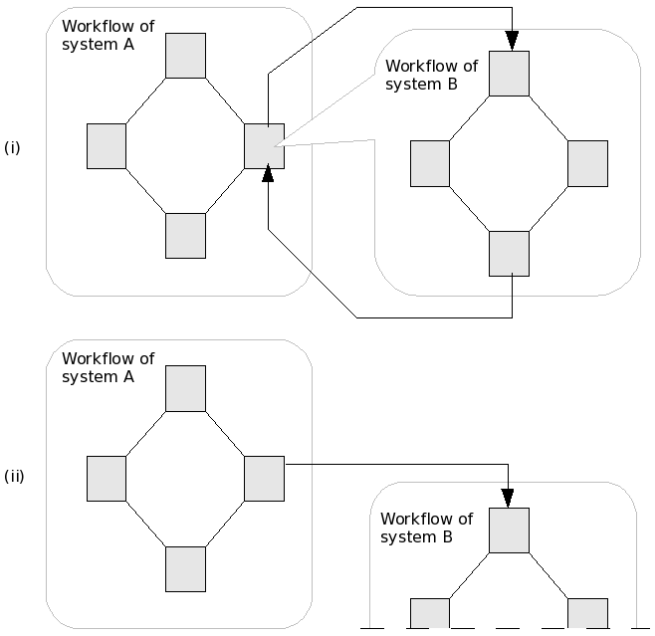

Fig. 2. (i) Non-native workflow nesting (synchronous execution) and (ii) non-native workflow invocation (asynchronous execution)

As it is illustrated on Figure 1, depending on the location of

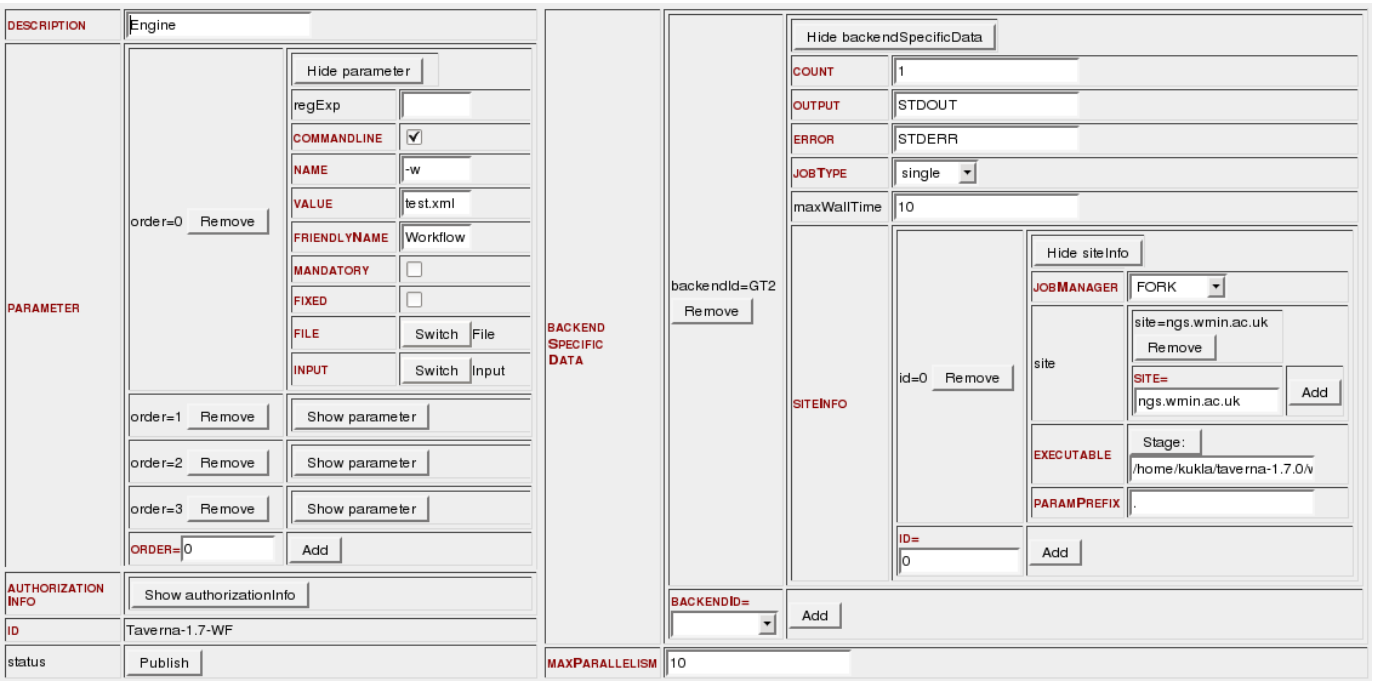

Fig. 3. Exposing Taverna workflow engine using GEMLCA Administration Portlet

the non-native engines the integration can be either tightly or loosely coupled. In the case of tightly coupled integration, non-native workflow engines are integrated directly to the workflow systems. In general, in the case of $n \in \mathbb{N}$ systems, $n(n-1)$ workflow engine integrations have to be implemented. Moreover, if a new version of a workflow engine emerges, $n-1$ integration will have to be upgraded.

Loosely coupled integration means that the workflow systems are extended with an interface via which they can access further workflow engines that will execute the non-native workflows. The non-native workflow engines are not integrated to the workflow systems directly, but via a mediator (for instance a service) that manages the non-native workflow execution. This technique provides an adoptable and easily extendable solution. In general, in the case of $n \in \mathbb{N}$ systems, $2n$ integrations have to be implemented (n workflow engine and n mediator client integrations). If a new version of a workflow engine emerges, only one integration will have to be upgraded. Therefore, it is not recommended to integrate non-native workflow engines to workflow management systems directly, in a tightly coupled fashion. It is suggested to realise loosely coupled integration.

In the case loosely coupled integration is realized via a service, the workflow engines can be executed either on the site where this workflow engine integration service is or on a third party computer. In order to support scalability, non-native workflow engines are best executed on the computational resources of the Grid. This approach brings two advantages: (1) it ensures scalability in the sense of number of running workflow instances, since the enactment of each non-native workflow can be scheduled to an individual, un-utilized computational node. (2) Both non-native child-workflows and ordinary nodes of the parent workflow are executed on the computational resources provided by the Grid. In the case of data intensive workflows, data transfer is critical. Executing workflow engines on the Grid enables e-Scientists to improve the performance of data transfer between ordinary nodes and non-native child-workflows, since non-native workflows are best to be scheduled 'close' to the sites where the native nodes are executed (from the perspective of data transfer). In order to achieve this, a mechanism is needed that helps e-scientist to control where the non-native engine will be executed.

IV. Realizing workflow engine integration

This section describes a generic solution and its reference implementation, which fulfils the above requirements. The solution integrates different workflow engines to a Grid application repository and submitter service, called GEMLCA [23]. Via GEMLCA, Grid based workflow systems can access non-native workflow engines if they have to execute such workflows. The reference implementation integrates Taverna, Triana, and Kepler workflow engines to the P-GRADE portal workflow management system through GEMLCA. The solution can be adopted by any workflow system by integrating the GEMLCA web service client to the given system. That particular workflow system will be able to execute any kind of workflow of which engine resides in a GEMLCA repository.

Within the P-GRADE portal, a workflow node (job) represents an application that has command line interface and can be submitted to the Grid. This application is defined and uploaded to the portal by the user at workflow editing time. Since, most workflow engines have command line interface,

```
<LCEnvironment maximumParallelism="10" id="Taverna-1.7-WF" status="private" xmlns="http://uk.ac.wmin.cpc.gemlca/sc
hema/legacyCodeConfig">
  <description>Engine</description>
  <parameter order="0" commandline="true" mandatory="false" fixed="false" file="true" input="true" name="-w">
    <value>test.xml</value>
    <friendlyName>Workflow</friendlyName>
  </parameter>
  <parameter order="1" commandline="true" mandatory="false" fixed="false" file="true" input="true" name="-p">
    <value>mystring.map</value>
    <friendlyName>parameter mapping</friendlyName>
  </parameter>
  <parameter order="2" commandline="true" mandatory="false" fixed="false" file="true" input="true" name="-i">
    <value>taverna_input.zip</value>
    <friendlyName>WF input</friendlyName>
  </parameter>
  <parameter order="3" commandline="true" mandatory="false" fixed="false" file="true" input="false" name="-o">
    <value>result.zip</value>
    <friendlyName>WF output</friendlyName>
  </parameter>
  <backendSpecificData backendId="GT2" error="STDERR" output="STDOUT" maxWallTime="10" jobType="single" count="1">
    <siteInfo id="0" jobManager="FORK">
      <site>ngs.wmin.ac.uk</site>
      <executable stage="false">/home/kukla/taverna-1.7.0/wfsubmit.sh</executable>
      <paramPrefix>.</paramPrefix>
    </siteInfo>
  </backendSpecificData>
  <authorizationInfo>
    <owner>/C=UK/O=eScience/OU=Westminster/L=ComputerScience/CN=tamas kukla</owner>
    <email>tamas.kukla@gmail.com</email>
  </authorizationInfo>
</LCEnvironment>
```

Fig. 4. Legacy Code Interface Description of a Taverna engine

it would be possible to execute such a non-native workflow engine within a P-GRADE portal workflow as a job if the software dependencies of the engine were resolved at the computational node where the workflow engine job would be submitted. However, it is not a sufficient solution to expect from the users to upload a workflow engine as their P-GRADE workflow jobs each time they need to execute non-native workflows and to keep in mind where the software dependencies are resolved. Therefore, a mediator is needed that manages the execution of non-native workflows on the Grid.

GEMLCA, that is unique in a sense that it is an application repository extended with a job submitter, allows the deployment of legacy code applications on the Grid. An application can be exposed via a GEMLCA service and can be executed using a GEMLCA client. The legacy application is stored either in the repository of a GEMLCA service or on a third party computational node where GEMLCA can access it. To publish a legacy application via GEMLCA, only a basic user-level understanding of the legacy application is needed, code re-engineering is not required. As soon as the application is deployed, GEMLCA is able to submit it using either GT2, GT4 [24] or gLite [25] Grid middleware. GEMLCA also provides a list of computational sites where the legacy application in question can be executed (these sites are defined by the administrator that publishes the legacy application) and allows e-Scientists to select a suitable site. Command-line workflow engines, just like other legacy applications, can be published via GEMLCA, without code re-engineering and can be automatically executed by GEMLCA on the Grid at a computational node where the required software background is available. If the workflow engine requires credentials to utilize further Grid resources for workflow execution, these are automatically provided by GEMLCA through proxy delegation.

Three engines (engine of Taverna, Triana, and Kepler) have been installed onto our cluster at the University of Westminster to a shared disk that any cluster node can access. The engines were en-wrapped by scripts so as to provide a general command line interface for them. This interface is the following:

```
wfsubmit.sh -w wf_descriptor [-p wf_input_params] [-i
wf_input_files] [-o wf_output_files]
```

Where `wf_descriptor` is the workflow descriptor file, `wf_input_params` is a text file containing the input parameters of the workflow, `wf_input_files` is an archive file containing the input files that will be processed by the workflow, finally, `wf_output_files` is also an archive file that contains the files that were created as a result of the workflow execution. During workflow execution, the wrapper scripts decompress the workflow input files, execute the workflow by parameterizing and invoking the workflow engine and finally compress the workflow outputs into one archive file.

In order to make the workflow engines accessible, they have

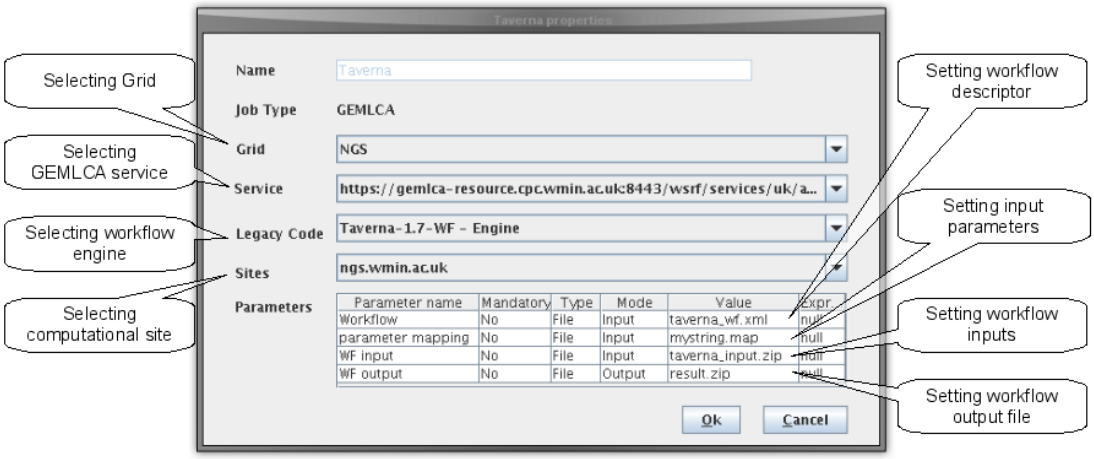

Fig. 5. Parametrization of non-native workflow execution

to be published via a GEMLCA service. This can be done using the GEMLCA Administration Portlet, that provides a web based graphical user interface and can be used either as a stand alone portlet or it can be integrated to any JSR-168 [26] based portal. Figure 3 shows how the Taverna engine can be exposed using this Portlet. First, the administrator defines the Grid and the GEMLCA service through which the engine will be published. (These are not illustrated on the figure.) Second, the input and output parameters of the workflow engine have to be defined. Then, the back-end specific data, that describes how and where the application can be executed, has to be specified. Next, the administrator has to upload the application (in this case the Taverna workflow engine) or specify the location where the application resides. Since, the engine is already installed on our cluster it does not have to be uploaded, only the location of the wrapper script has to be given. It should be noted that multiple locations of an engine

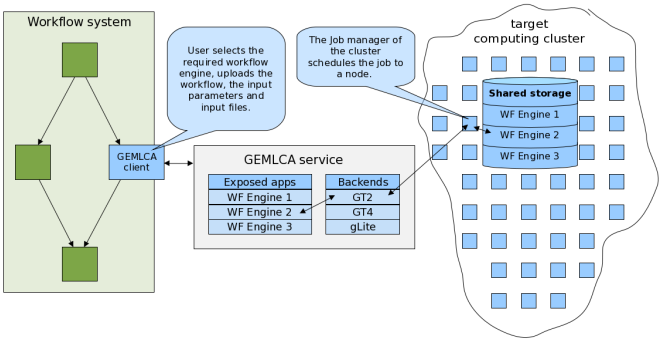

Fig. 6. Non-native workflow execution via GEMLCA

can be defined and multiple instances of the same engine can be uploaded and mapped to different computational sites. Finally, all this information is stored in an XML based LCID (Legacy Code Interface Description) file, that is generated by the portlet. This LCID file can be seen on Figure 4. According to this description, the Taverna-1.7-WF engine application has four command line file parameters: three input files and one output file. These parameters suit the interface specified by the wrapper script that will invoke the Taverna engine. The back-end that manages the execution uses the GT2 Grid middleware. The engine will be executed on the cluster at *ngs.wmin.ac.uk* and does not have to be transferred to this location, since it is already installed to a shared directory, that any cluster node and NGS user can access.

The engines can be installed to any cluster that is based on GT2, GT4 or gLite grid middleware or it even can be placed to a GEMLCA repository. In which case, GEMLCA transfers the application to the desired computational node before execution. Any further workflow engine that has command line interface can be deployed to GEMLCA. The deployment requires only user level understanding of the engine and does not require code re-engineering. Writing wrapper scripts is not necessary either, we developed them only to provide a generic command line interface for the engines that we integrated.

Figure 6 illustrates how a published workflow engine can be executed using GEMLCA as a non native workflow node within a workflow system that adopted the solution. The node connects to the GEMLCA service via which the non-native workflow engine can be executed. GEMLCA knows where the instances of the engine reside and on which computational resources those instances can be executed and/or submitted to. GEMLCA provides a list of these sites and the e-Scientist has to choose on which site the engine has to be executed. GEMLCA receives the workflow description and the workflow input data, parameterizes and executes the engine on the selected computational site. When the workflow execution is finished the results are gathered and transferred to the site where the next computational nodes in the original workflow will process them.

GEMLCA is integrated to the P-GRADE portal in a fashion that applications in GEMLCA are represented as P-GRADE workflow nodes. This approach realizes non-native workflow nesting, since, P-GRADE is able to execute any application exposed via GEMLCA as P-GRADE nodes, including

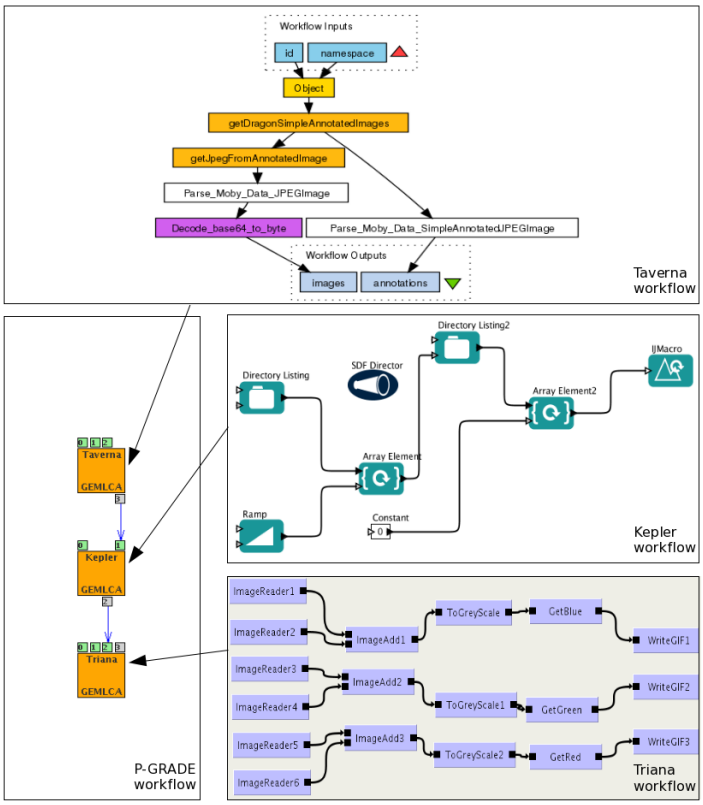

Fig. 7. Heterogeneous P-GRADE workflow embedding Triana, Taverna and Kepler workflows

workflow engines with command line interfaces. In order to realize asynchronous workflow invocation, GEMLCA has to be integrated in a special way that allows the asynchronous execution of GEMLCA jobs.

The P-GRADE portal provides graphical user interface for specifying GEMLCA nodes at workflow editing time. Figure 5 shows how a Taverna engine should be parametrized within the portal. First, the user selects the Grid and the GEMLCA service where the engine is deployed. Second, the user selects the legacy application, which in our case is the Taverna engine, and the computational site where the engine will be executed. A parameter table, that is specific to the selected engine shows up. Next, the user specifies the input and output arguments of the application. In the case of the Taverna engine, the e-Scientist has to set the Taverna workflow descriptor file, a file that contains the workflow input parameters, and an archive file that contains all the workflow input files. Finally, the name of the workflow outputs archive file has to be defined as well. The parametrization of the Triana engine is identical to the Taverna engine, while the Kepler engine that is installed on our cluster does not provide interface for specifying workflow input parameters. Therefore, only three parameters have to be specified in its parameter table: a Kepler workflow descriptor file, an archive file that contains the input files of the workflow, and the name of the archive file that will contain all the output files.

The file parameters will be represented as ports of the GEMLCA workflow node in the workflow graph. These ports can be connected to ports of other workflow nodes enabling run-time generation of the workflow input data and the workflow descriptor files and further processing the outputs of the embedded workflow by the following nodes.

V. Case Study

This section presents a P-GRADE workflow, which embeds a Taverna, a Kepler and a Triana workflow. It should be noted that this is not a real life example; it serves only demonstration purposes by presenting how different workflows of different workflow systems can interoperate and form a heterogeneous high level workflow. The P-GRADE workflow, that can be seen on the left hand side of Figure 7, consists of three nodes. Each node represents an embedded workflow of another workflow system. It should be noted that the inputs and outputs of the embedded workflows are files; there is no data transformation between them. If data transformation is needed, transformator jobs have to be defined by the user.

The first node, that represents a Taverna workflow, has three input ports representing three files: (0) the Taverna workflow descriptor, (1) the workflow input parameters and (2) the workflow input files zipped into one file. This embedded Taverna workflow fetches several images, creates a few directories and places the images into those directories as image files. Then, the whole directory structure is zipped into an archive file by the wrapper script and appears on the output port (3) of the first node in the P-GRADE workflow.

This archive file is fed to the next P-GRADE node, which represents a Kepler workflow. This node has two input ports: (0) the Kepler workflow descriptor file and (1) the workflow input files zipped into an archive file containing the images. This workflow goes through the directory structure of the archive input file and manipulates each image that it finds. The manipulation includes edge highlighting, picture resizing and image type conversion. Finally the new images are zipped to another archive file by the wrapper script and will appear on the output port (2) of the second P-GRADE workflow node.

The Triana node, similarly to the Taverna workflow, has three input ports: (0) the Triana workflow descriptor file, (1) the workflow input parameters and (2) the workflow inputs zipped into one archive, which is generated by the Keper node. This workflow couples the pictures that are in the zip file, merges each couple and converts the merged pictures to grayscale images. Then, one colour component, that can be blue, green or red, are taken of the grayscale pictures and saved as new image files. Finally, these files are compressed to the last archive file by the wrapper script, and will serve as the output of the Triana as well as the P-GRADE workflow and will appear on the output port (3) of the node.

VI. Conclusions and Future work

This paper introduced a general solution to workflow interoperability and sharing at the level of workflow integration. The solution exposes various workflow engines via a GEMLCA service, that is capable of executing the engines on the Grid. Hence, it keeps the data at computational sites and offers a solution that is scalable in terms of number of workflows and amount of data. Workflow engine deployment to this system does not require any code re-engineering, user level understanding is sufficient. The reference implementation exposes three workflow engines (Taverna, Triana and Kepler) via a GEMLCA service.

Since GEMLCA is integrated to the P-GRADE workflow system, P-GRADE became capable of executing non-native Taverna, Triana and Kepler workflows inside a P-GRADE workflow. The solution can be adopted by any other workflow system by integrating the GEMLCA web service client to the given workflow system. Any system that adopts this solution becomes a workflow system that supports high-level heterogeneous workflow design and execution.

The approach described in this paper supports two models of interoperability: asynchronous workflow execution (invocation) and synchronous workflow execution (nesting). Although, the reference implementation supports only workflow nesting, the same approach can be used to implement asynchronous workflow invocation.

Our main concern is to investigate and realize further workflow interoperability models and to extend the solution with a workflow repository. The integration technique can be applied not only for workflow engine integration. A publication that mainly concerns with data middleware integration to Grid workflow systems will describe a solution based on this approach in the near future.

References

[1] S. Majithia *et al.*, "Triana: a graphical Web service composition and execution toolkit," *Web Services, 2004. Proceedings. IEEE International Conference on*, pp. 514–521, 2004.

[2] P. Kacsuk and G. Sipos, "Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal j," *Journal of Grid Computing*, vol. 3, pp. 221–238, 2006.

[3] T. Oinn *et al.*, "Taverna: a tool for the composition and enactment of bioinformatics workflows," pp. 3045–3054, 2004.

[4] B. Ludäscher *et al.*, "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience*, 2005.

[5] CppWfMS, "CppWfMS on-line documentation," 2008. [Online]. Available: http://wfms.forge.cnaf.infn.it/documentation/index.html

[6] W. van der Aalst, L. Aldred, M. Dumas, and A. ter Hofstede, "Design and implementation of the YAWL system," *Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04)*, 2004.

[7] A. Hoheisel, "Grid Workflow Execution Service-Dynamic and interactive execution and visualization of distributed workflows," *Proceedings of the Cracow Grid Workshop*, 2006.

[8] WfMC, "Workflow Management Coalition Terminology and Glossary," *Workflow Management Coalition*, 1999.

[9] WFM-RG, "Workflow Management Research Group Project Website," 2008. [Online]. Available: http://forge.gridforum.org/sf/projects/wfm-rg

[10] M. Shields, "Gridnet2 activities for wfm research group," Oct. 2007. [Online]. Available: *http://wiki.cs.cf.ac.uk/twiki/bin/viewfile/Sandbox/OpenGridForum21?rev=1;filename=GridNet_Ian.ppt*

[11] WfMC, "Workflow Management Coalition Website," 2008. [Online]. Available: http://www.wfmc.org

[12] J. Frey, "Condor DAGMan: Handling Inter-Job Dependencies," Tech. rep., University of Wisconsin, Dept. of Computer Science, http://www. cs. wisc. edu/condor/dagman, Tech. Rep., 2002. [Online]. Available: http://www.bo.infn.it/calcolo/condor/dagman/

[13] S. Pellegrini, F. Giacomini, A. Ghiselli, and A. Hoheisel, "Using GWorkflowDL for Middleware-Independent Modeling and Enactment of Workflows," *Proceedings of the CoreGRID Integration Workshop*, 2008.

[14] WfMC, "Workflow Management Coalition Workflow Standard - Interoperability Abstract Specification," *Workflow Management Coalition*, 1999.

[15] W. van der Aalst and A. ter Hofstede, "YAWL: yet another workflow language," *Information Systems*, vol. 30, no. 4, pp. 245–275, 2005.

[16] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[17] A. Brogi and R. Popescu, "From BPEL Processes to YAWL Workflows," 2006.

[18] S. Pellegrini, F. Giacomini, and A. Ghiselli, "A PRACTICAL APPROACH FOR A WORKFLOW MANAGEMENT SYSTEM," *Proceedings of the CoreGRID Workshop*, 2007.

[19] M. Ghanem, N. Azam, and M. Boniface, "Workflow Interoperability in Grid-based Systems," *Cracow Grid Workshop*, 2006.

[20] P. Kacsuk and T. Kiss, "Towards a scientific workflow-oriented computational World Wide Grid," *CoreGRID Technical Report*, 2007.

[21] Z. Zhao, S. Booms, A. Belloum, C. de Laat, and B. Hertzberger, "VLE-WFBus: A Scientific Workflow Bus for Multi e-Science Domains," *e-Science and Grid Computing, 2006. e-Science'06. Second IEEE International Conference on*, pp. 11–11, 2006.

[22] C. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the Ptolemy Project," *University of California, Berkeley, Engineering-Electrical Engineering and Computer Sciences*, 2003.

[23] T. Delaitre *et al.*, "GEMLCA: Running Legacy Code Applications as Grid Services," *Journal of Grid Computing*, vol. 3, no. 1, pp. 75–90, 2005.

[24] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Journal of Computer Science and Technology*, vol. 21, no. 4, pp. 513–520, 2006.

[25] E. Laure *et al.*, "Programming the Grid with gLite," *Computational Methods in Science and Technology*, vol. 12, no. 1, pp. 33–45, 2006.

[26] P. Specification, "The Java Community Process, JSR 168," 2003.