Architectures for workflow integration in science gateways

Abstract

Keywords:

Architectures for workflow integration in science gateways

1. Introduction

Context. The question of integrating workflow engines in science gateways can be seen at various levels, corresponding to various definitions of workflows. One level is the SHIWA level, where it was considered that workflow engines are aware of the DCI (and 'D' is important because of data transfers, proxies, etc). Another level is to remove 'D' from the definition: a workflow engine becomes a program that submits jobs, potentially only to local clusters. It opens a whole new class of workflow engines that we used to consider as "applications". For instance, in neuroinformatics: Nipype, PSOM, but also FSL through the fslsub tool. A workflow engine is not supposed to be aware of the science gateway. A wide-array of workflow engines are available with specificies coming from the application domain, available tools, etc Their integration in science gateways becomes critical. Several of the examples presented in the paper will be taken from medical image analysis, in particular neuroimaging.

Goal. this paper reviews and compares the architectures to integrate workflow engines in science gateways.

Contributions.

- We evaluate architectures based on our experience with existing systems
- We propose a new architecture

2. Workflow engines and science gateways

2.1. Workflow engines

In the last decade, the e-Science workflow community has developed high-level workflow systems to help developers access distributed infrastructures such as grids and web services, resulting in tools such as Askalon, Hyperflow, MOTEUR, Pegasus, Swift, Taverna, Triana, VizTrails, WS-PGRADE/gUSE, etc. Such workflow engines usually describe applications in a specific language that offers operators for parallel computing, visual description and edition, links with domain-specific tool repositories, etc. Descriptions and experiments conducted with e-Science workflow engines were published in journals such as Future Generation Computer Systems and conference venues such as WORKS.

At the same time, specific toolboxes were emerging in various scientific domains to facilitate interactions between different software components. In neuroimaging for instance, tools such as Nipype, PSOM, SPM and FSL provide abstractions and functions to define processes that handle the data flow between other processes. Soon, such tools were interfaced to computing systems, in particular clusters: they were extended to create cluster tasks, handle their dependencies, and execute them on clusters. For instance, FSL can launch tasks on SGE through its `fsl_sub` tool, Nipype ... , PSOM ... , and SPM ... . Although distributed infrastructures are hardly handled, such domain-specific tools have to be called workflow engines. They represent a tremendous opportunity for science gateways to leverage existing tools and applications. Whether these should be combined with e-Science workflow engines is an open question.

A workflow engine is defined as a software that submits jobs to a cluster, grid or cloud. Workflows may be expressed in any language, including scripts. Several workflow engines may be combined in the same science gateway. Workflow: a process that submits tasks to the infrastructure. FSL tools, say Feat, can be considered as workflows when they are configured to submit tasks to clusters using fslsub. A workflow consists of activities. We are talking only about concrete workflows with a configuration (see terstyansky et al 2014 "enabling scientific workflow sharing...")

Workflow engine: executes workflows, i.e. pro-

cess their dependencies and create computing tasks. Might transfer data too. A workflow engine is not supposed to be aware of the science gateway.

2.2. Science gateways

Science Gateway: An ecosystem, usually accessible through a web portal, that provides tools to access distributed infrastructures. Tools usually help users manage data transfers, task execution and authentication on multiple computing and storage locations. Examples: CBRAIN, VIP, NSG, etc

Multi-engine, multi-language.

2.3. Infrastructures

Maybe this is not relevant

The infrastructure consists of the computing and storage resources involved in the workflow execution, and the software services used to access these resources. Infrastructures can be servers, clusters, grids or clouds. Some workflow engines may assume specific characteristics about the infrastructure, such as the presence of a shared file system between the computing nodes.

3. Architectures

Architectures are defined below from their main components and the interactions between these components.

Introduce the notion of architectural diagram with an abstract figure.

3.1. Interactions

The interactions described below are labeled consistently with the notations used on Figures 1-6.

`(a)` Workflow integration: consists in adding a new workflow to the science gateway so that users can execute it. It is conducted by science gateway administrators, developers or users, and it results in an interface, for instance a web form, where users can enter the parameters of the workflow to be executed. Integrating a workflow is not the same process as integrating a workflow engine.

`(b)` Task control: operations to manage tasks on the infrastructure, including: submission, monitoring, termination, deletion, etc.

`(c)` Data control: operations to manage data on the infrastructure, such as: upload, download, deletion, browsing, replication, caching, etc. Data movements can be triggered by the user, to upload input data or download processed data – interaction c_1, or by the workflow engine, to transfer data across the infrastructure – interaction c_2. *add c1 and c2 to the Figures*

`(d)` Workflow control: operations to execute a workflow, including: submission, monitoring, termination, etc. Workflow control can be coarse-grained (a.k.a. black box) or fine-grained (white box). In a coarse-grained model, workflow activities are masked.

`(e)` Sub-task control: operations used by tasks to submit sub-tasks on the infrastructure, including: submission, monitoring, termination, deletion, etc.

`(f)` Pool-agent: specific to the architecture described in 3.5.

`(g)` Workflow conversion: translation from one workflow language to another one.

3.2. Tight integration

See Figure 1. The workflow engine is tightly integrated with the science gateway, which means that it is deployed on the same machine and potentially shares code, libraries and other software components with the science gateway. For instance, the workflow engine might be a portlet in a Liferay portal, a controller in a Ruby on Rails application, and so on. The workflow engine and the science gateway usually share a database where application, users and other resources are stored. In this model, task control and data control are initiated from the science gateway. This is the model adopted in the Catania Science Gateway Framework [1] (see specific documentation on workflows[1]), CIPRES [2] and LONI Pipeline Environment [3]. *Add better justification*

3.3. Service invocation

See Figure 2. The workflow engine is available externally to the science gateway. The science gateway controls it through a specific interaction that might be implemented as a web-service call (e.g.

[1] `http://bit.ly/1oQrzvQ`

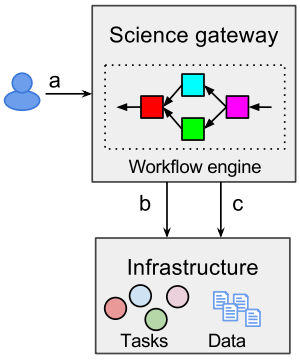

Figure 1: Tight integration

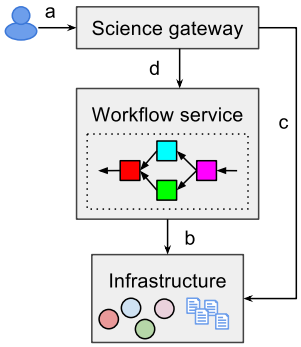

Figure 2: Service integration

RESTful or SOAP), as a command-line or as any other method that separates the workflow engine from the science gateway. The workflow engine might be invoked either as a black box that completely masks the infrastructure and workflow activities, or as a grey box that allows for some interaction with them. The workflow engine is responsible for controlling the tasks on the infrastructure, and performing the required data transfers to execute them. User data is usually managed through the science gateway, although it might as well be delivered by the workflow engine directly to the user.

This architecture is largely adopted, in systems such as Apache Airvata [4], Vine Toolkit [5], Virtual Imaging Platform [6], the WS-PGRADE/gUSE framework [7] and the numerous science gateway instances that use it [8].

3.4. Sub-tasking

See Figure 3. The science gateway is responsible to execute the tasks on the infrastructure, dealing with heterogeneous batch managers and meta-schedulers. The workflow engine is a particular

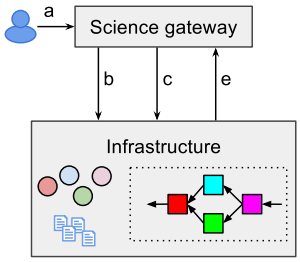

Figure 3: Sub-tasking

task that can submit sub-tasks to the science gateway. The workflow engine keeps track of the dependencies between the sub-tasks but their execution is delegated to the science gateway. The science gateway may implement mechanisms to deal with task dependencies such as basic dependency lists as available in most batch managers. Although it has no global vision of the workflow, it can keep track of the sub-tasks submitted by a given task, for instance to be able to cancel them when the task is canceled. The fact that tasks are submitted through the science gateway does not restrict the range of possible execution models: pilot jobs, for instance, can still be used.

This model is implemented in CBRAIN [9] where it is used to integrate the PSOM workflow engine and the FSL toolkit through its `fsl_sub` tool. In particular, the CBRAIN-PSOM integration is described in [10] and uses an agent computing model (a.k.a pilot jobs).

3.5. Pool model

See Figure 4. Workflows are submitted by the science gateway to a pool to which agents connect asynchronously to retrieve and execute workflows (interaction `f`). Agents may be started according to various policies, for instance to ensure load balancing. Agents may wrap different types of workflow engines. This model was implemented in the SHIWA pool [11].

3.6. Nested workflows

See Figure 5. A nested workflow is an abstract architectural pattern where an activity of a master workflow is itself a workflow that is executed by another engine. Nested workflows can be instantiated in the various architectures described previously. We focus on instantiation with the service invocation model as this is the most used architecture. The master workflow is also called meta-workflow.

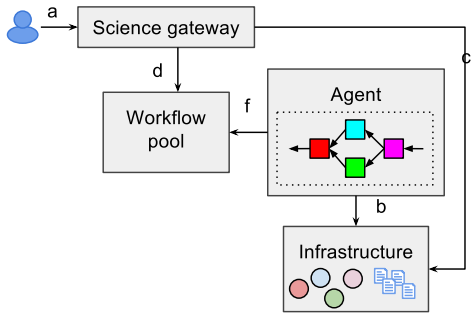

Figure 4: Pool model

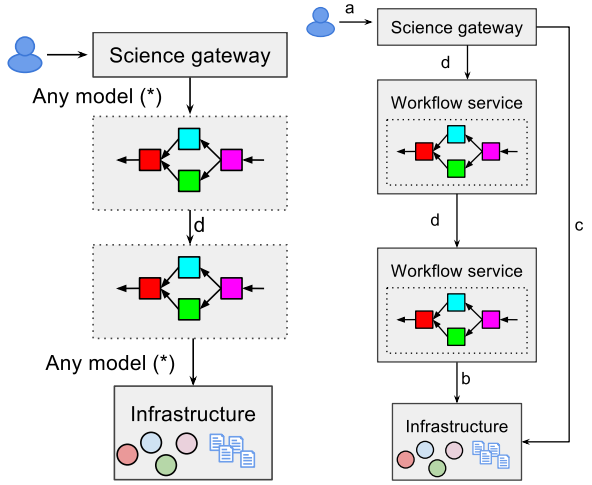

Figure 5: Nested workflows. Left: abstract model. Right: instantiation with service invocation.

The nested workflow might be described with a different language than the one used to describe the master workflow.

Nested workflows have long been available in workflow engines, for instance in the Taverna workbench [12]. This model was used by the SHIWA Simulation Platform to implement Coarse-Grained workflow interoperability [13], i.e. to integrate various workflow engines in a consistent platform. Nested workflows are also often used implicitly to wrap scripts as black-boxes while these scripts actually correspond to complete workflows.

3.7. Workflow import

See Figure 6. This is an abstract model that we instantiate with the service invocation architecture for consistency. Workflows are integrated in the science gateway through format conversion from a native format to the science gateway format. This

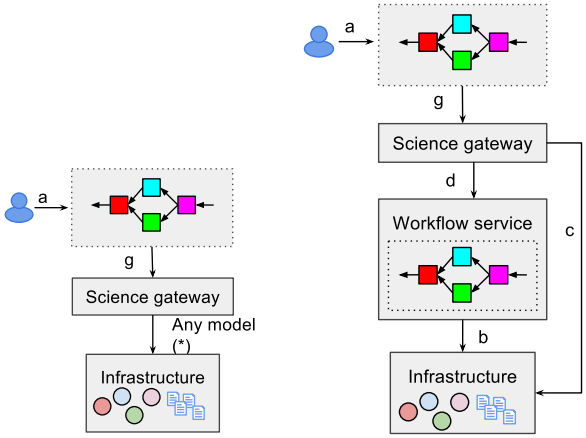

Figure 6: Workflow import. Left: abstract model. Right: instantiation with service invocation.

was implemented in the SHIWA Simulation Platform through the IWIR language that provided a common language for portability across grid workflow systems [14].

4. Evaluation

4.1. Criteria

We use four main criteria to evaluate the architectures: integration effort, robustness, modularity and scalability. Criteria break down to specific metrics for which low values indicate good performance.

Explain these metrics better, including why the related features are useful.

Integration effort measures the development effort required to build the system, assuming that none of the components could initially communicate with each other. It is measured by counting the number of interactions to develop and components to modify on the architectural diagram. It breaks down to 2 metrics that relate to the direction of interaction arrows on the architectural diagrams:

- Interactions from the science gateway.
- Iteractions from the workflow engine.

Robustness relates to the complexity of the architecture and its ability to provide relevant debugging information about the workflow executions. It breaks down to 3 metrics:

- Components: number of unique components involved in workflow executions.
- Interactions: number of unique interactions involved in workflow executions.

- Debugging: availability of debugging information about workflow activities (1: difficult to obtain; 0: easy to access).

Modularity measures the ability to replace elements in the architecture, or to integrate new elements. Elements include workflow engines, workflows and infrastructure. It breaks down in 4 metrics:

- New engine type: number of interactions or components to modify to integrate a new workflow engine type in the architecture.
- New engine version: number of interactions or components to modify to integrate a new version of a workflow engine in the architecture, assuming that another version of the same engine type is already available.
- New infrastructure: number of interactions or components to modify to integrate a new type of infrastructure in the architecture.
- New workflow: number of interactions required to integrate a new workflow in the architecture.
- Meta-workflow: ability to describe meta-workflows from existing workflows (1: not available; 0: available).

Scalability measures the ability of the architecture to support high workloads. It breaks down into 4 metrics:

- New engine instance (0: automated; 1: manual intervention needed).
- Elastic engines (0: easy to implement; 1: difficult to implement).
- Distributed engines (0: possible; 1: not possible).
- Task scheduling (0: difficult; 1: more difficult).

4.2. Evaluation

See Table 1.

We focus on the integration of workflow engines in science gateways, i.e. we assume that the science gateway is already interfaced with the infrastructure.

Service invocation. Integrating a workflow engine as a service requires 3 interactions. First, the science gateway has to be extended to call the engine interface (`d`), which might require some adaptation on the workflow engine too, for instance wrapping in a web service. In addition, the workflow engine has to be interfaced with the infrastructure, which requires the two interactions `b` and c_2.

+ Easy to implement in the SG (just call the service) - Load-balancing between services is difficult (refer to VIP's experience). - Need to wrap workflow engines in services. - Need to know where to submit each workflow, e.g. when multiple workflows are involved. - The science gateway has no knowledge of the detailed progress of the workflow, in particular job-level information (statuses, stdout, stderr). If we wanted to do that, specific feedback channels have to be developed, which requires more work (see VIP).

Sub-task. Science gateway has to make sure that the machine on which a workflow task executes is reliable as several sub-tasks will depend on the success of this task. Or workflow engine task has to be recoverable.

Performance: there could be a deadlock when there is contention. Also, the workflow task has to be reliable (for instance, proper walltime estimation), which is difficult because these tasks are long. Restarting a workflow task may mean that all the tasks in the workflow have to be restarted, depending on the capability of the workflow engine. This complexifies task scheduling (a system could ignore these issues but they may backfire).

Pool. = Evaluation + Scale-up and load balancing + Multi-language - Needs a complete framework (the pool). Most likely third-party software as it's a lot of work to implement.

5. Discussion

These architectures can be combined (give examples).

6. Conclusion

7. Acknowledgments

FLI-IAM, Labex PRIMES, Ludmer Centre

References

[1] V. Ardizzone, R. Barbera, A. Calanducci, M. Fargetta, E. Ingrà, I. Porro, G. La Rocca, S. Monforte, R. Ricceri, R. Rotondo, D. Scardaci, A. Schenone, The decide science gateway, Journal of Grid Computing 10 (4) (2012) 689–707. doi:10.1007/s10723-012-9242-3. URL http://dx.doi.org/10.1007/s10723-012-9242-3

Metric	Tight	Service	Sub-task	Pool	Nested	Import
Integration effort						
From science gateway	3	1	2	2	2	3
From workflow engine		1	2	2	2	1
Total	**3**	**3**	**3**	**4**	**4**	**4**
Robustness						
Components	2	3	2	4	4	4
Interactions	2	3	3	4	4	4
Debugging	0	0	1	0	1	0
Total	**4**	**6**	**6**	**8**	**9**	**8**
Modularity						
New engine type	3	2	1	2	2	3
New engine version	1	0	0	0	0	0
New infrastructure	2	2	2	2	2	2
New workflow	1	1	1	1	1	2
Meta-workflow	1	1	1	1	0	1
Total	**8**	**6**	**5**	**6**	**5**	**8**
Scalability						
New engine instance	1	1	0	0	1	1
Elastic engines	2	1	0	0	1	1
Distributed engines	1	1	1	1	0	1
Task scheduling	0	0	1	0	0	0
Total	**4**	**3**	**2**	**1**	**2**	**3**
Grand total						
	19	**18**	**16**	**19**	**20**	**23**

Table 1: Architecture evaluation. Lower values (brighter colors) indicate better performance.

[2] M. A. Miller, W. Pfeiffer, T. Schwartz, Creating the cipres science gateway for inference of large phylogenetic trees, in: Gateway Computing Environments Workshop (GCE), 2010, IEEE, 2010, pp. 1–8.

[3] I. D. Dinov, J. D. Van Horn, K. M. Lozev, R. Magsipoc, P. Petrosyan, Z. Liu, A. MacKenzie-Graham, P. Eggert, D. S. Parker, A. W. Toga, Efficient, distributed and interactive neuroimaging data analysis using the loni pipeline, Frontiers in neuroinformatics 3 (JUL).

[4] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler, et al., Apache airavata: a framework for distributed applications and computational workflows, in: Proceedings of the 2011 ACM workshop on Gateway computing environments, ACM, 2011, pp. 21–28.

[5] D. Szejnfeld, P. Dziubecki, P. Kopta, M. Krysinski, T. Kuczynski, K. Kurowski, B. Ludwiczak, T. Piontek, D. Tarnawczyk, M. Wolniewicz, P. Domagalski, J. Nabrzyski, K. Witkowski, Vine toolkit - towards portal based production solutions for scientific and engineering communities with grid-enabled resources support, Scalable Computing: Practice and Experience 11 (2).

[6] T. Glatard, C. Lartizien, B. Gibaud, R. Ferreira da Silva, G. Forestier, F. Cervenansky, M. Alessandrini, H. Benoit-Cattin, O. Bernard, S. Camarasu-Pop, N. Cerezo, P. Clarysse, A. Gaignard, P. Hugonnard, H. Liebgott, S. Marache, A. Marion, J. Montagnat, J. Tabary, D. Friboulet, A virtual imaging platform for multi-modality medical image simulation, IEEE Transactions on Medical Imaging 32 (1) (2013) 110–118. doi:10.1109/TMI.2012.2220154.

[7] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, I. Marton, Ws-pgrade/guse generic dci gateway framework for a large variety of user communities, Journal of Grid Computing 10 (4) (2012) 601–630. doi:10.1007/s10723-012-9240-5.
URL http://dx.doi.org/10.1007/s10723-012-9240-5

[8] P. Kacsuk, Science Gateways for Distributed Computing Infrastructures, Springer, 2014.

[9] T. . Sherif, P. . Rioux, M. . Rousseau, N. . Kassis, N. . Beck, R. . Adalat, S. . Das, T. Glatard, A. Evans, Cbrain: A web-based, distributed computing platform for collaborative neuroimaging research, Frontiers in Neuroinformatics 8 (54). doi:10.3389/fninf.2014.00054.

[10] T. Glatard, P. Quirion, R. Adalat, N. Beck, R. Bernard, B. Caron, Q. Nguyen, P. Rioux, M.-E. Rousseau, A. Evans, P. Bellec, Integration between psom and cbrain for distributed execution of neuroimaging pipelines (2016).
URL https://ww4.aievolution.com/hbm1501/index.cfm?do=abs.viewAbs&abs=1859

[11] D. . Rogers, I. . Harvey, T. T. . Huu, K. . Evans, T. Glatard, I. . Kallel, I. . Taylor, J. Montagnat, A. . Jones, A. . Harrison, Bundle and pool architecture for multi-language, robust, scalable workflow executions, Journal of Grid Computing 11 (3) (2013) 457–480. doi:http://dx.doi.org/10.1007/s10723-013-9267-2.
URL http://link.springer.com/article/10.1007%2Fs10723-013-9267-2

[12] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, et al., Taverna: a tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (17) (2004) 3045–3054.

[13] G. Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, Á. Balaskó, Z. Farkas, Enabling scientific workflow sharing through coarse-grained interoperability, Future Generation Computer Systems 37 (2014) 46–59.

[14] K. Plankensteiner, R. Prodan, M. Janetschek, T. Fahringer, J. Montagnat, D. Rogers, I. Harvey, I. Taylor, Á. Balaskó, P. Kacsuk, Fine-Grain Interoperability of Scientific Workflows in Distributed Computing Infrastructures, Journal of Grid Computing (JOGC) IF=1.603 11 (3) (2013) 429–456.
URL http://hal.archives-ouvertes.fr/docs/00/83/22/14/PDF/jogc12-iwir.pdf