Architectures for workflow integration in science gateways

Abstract

Keywords:

Architectures for workflow integration in science gateways

1. Introduction

[Say that the review is done based on our experience with VIP, CBRAIN, and to some extent SHIWA.]

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. process 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

The architectures are diagramed in Figure 2 using the graphical notations shown in Figure 1. Architectures are described from their main software *components* and *interactions*. Software components include science gateway and infrastructure in all architectures, and workflow service, workflow pool and agent are involved in some architectures. The workflow engine itself is represented by a specific symbol. Software interactions among these components are described in the next Section. *Abstract* interactions are specific types of interactions that may be implemented by various different software interactions. An architecture that has an abstract interaction is an abstract architecture. Red color is used to represent the components and interactions that need to be specifically developed to integrate workflow engines. Tasks refer to computing tasks that are created by the workflow engine and executed on the infrastructure. Data represents any type of file, or database that is involved in the workflow execution and stored on the infrastructure while the workflow executes. It does not cover

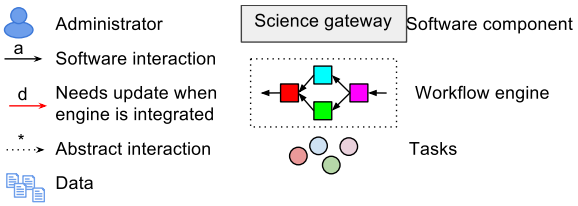


Figure 1: Graphical notations

workflow parameters such as strings, numbers, etc. The reminder of this section describes the different types of interactions involved in the architectures, and details each architecture.

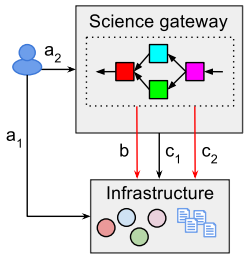
3.1. Interactions

The interactions involved in the architectures are described below and labeled consistently with the notations used on Figure 2.

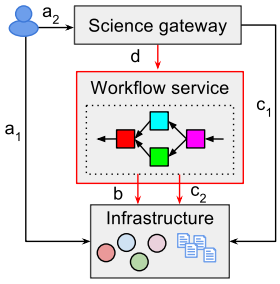
(a) Workflow integration: consists in adding a new workflow to the system so that users can execute it. It is triggered by an administrator of the science gateway and it results in an interface, for instance a web form, where users can enter the parameters of the workflow to be executed. The interaction has two aspects: (a_1) the programs used in the workflow are installed on the infrastructure, which may or may not require administrative privileges on the infrastructure; (a_2) the workflow has to be configured in the science gateway so that it becomes available to users. Note that integrating a workflow is not the same process as integrating a workflow *engine*.

(b) Task control: operations to manage tasks on the infrastructure, including: authentication, submission, monitoring, termination, deletion, etc. Controlling tasks requires to deal with the heterogeneous batch managers and meta-schedulers that might be available on the infrastructure. When the infrastructure is a grid or a cloud, it is for instance implemented using libraries such as SAGA, DRMAA, OCCI and similar initiatives *[refs needed]* .

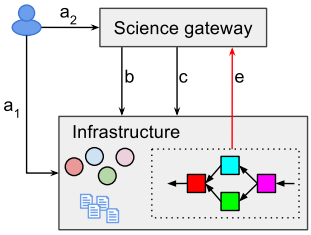
(c) Data control: operations to manage data on the infrastructure, such as: upload, download, deletion, browsing, replication, caching, etc. Data movements can be triggered by the user in the science gateway (c_1), to upload input data or download processed data. They can also be performed by the workflow engine (c_2), to transfer data across



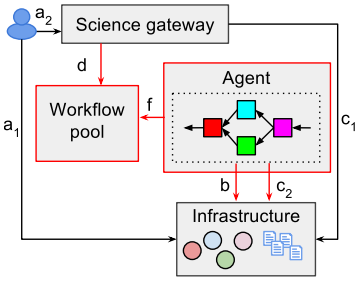
(a) Tight integration



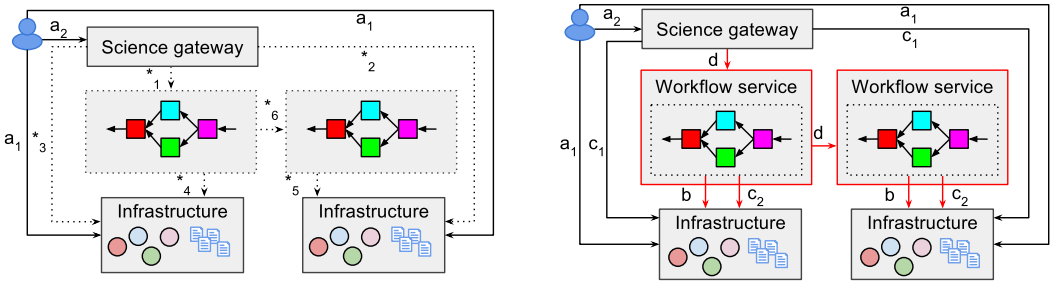
(b) Service invocation



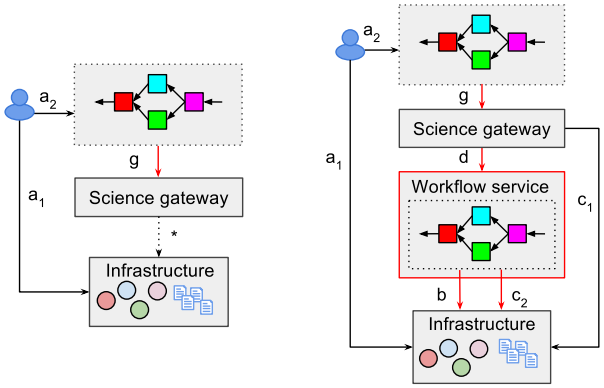
(c) Sub-tasking



(d) Pool



(e) Nested workflows. Left: abstract model. Right: instantiation with service invocation.



(f) Workflow import. Left: abstract model. Right: instantiation with service invocation.

Figure 2: Architectures

the infrastructure. The infrastructure might offer various data storage backends with heterogeneous interfaces. Tools and services such as JSAGA [1] or Data Avenue [2] can be used to homogeneize these interfaces.

(d) Workflow control: operations to execute a workflow with an engine, including: workflow 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 and the user only has a global view of the workflow execution. In a fine-grained model, user is exposed to the workflow topology, i.e. to the outputs of the individual tasks, their statuses and so on.

(e) Sub-task control: operations used by tasks to submit sub-tasks on the infrastructure, including: submission, monitoring, termination, deletion, etc. Sub-task control is similar to interaction b, except that information about the parent task which submitted a sub-task is usually available and used for additional control. For instance, the parent task may wait for all its sub-tasks to finish before finishing, and conversely all the sub-tasks may be killed if the parent task is killed.

(f) Pool-agent: specific to the architecture described in Section 3.5. This is an interaction used when agents retrieve work from a central pool. It covers agent registration and de-registration to the pool, protocols to send work from the pool to the agent, mechanisms to update work status, and so on.

(g) Workflow conversion: translation from one workflow language to another one. This interaction may not be available or implementable for every workflow language. It has been developed mostly for well-structured and relatively simple workflow language such as between GWorkflowDL and Scufl [3] and between the 4 languages that were involved in the SHIWA initiative. In SHIWA, workflow conversion is done through an intermediate representation, the Interoperable Workflow Intermediate Representation [4], which allows to convert among n workflow languages using $2n$ interactions instead of n^2.

[In the following sections, for each architecture, add a description of a real system that implements it: integrated: ?, service: VIP, sub-task: CBRAIN-PSOM, pool: SHIWA pool, nested workflow: SHIWA CGI, import: SHIWA FGI]

3.2. Tight integration

See Figure 2a. 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 or a controller in a Ruby on Rails application. The workflow engine and the science gateway usually share a database where application, users and other resources are stored. In this model, task and data control are both initiated from the science gateway. Interactions b and c_2 are initiated from the workflow engine while c_1 comes from other parts of the science gateway, for instance a data management interface. As in any other model, the installation of new workflows in the science gateway (a_2) and infrastructure (a_1) is done by an administrator, for security reasons. This is the model adopted in the Catania Science Gateway Framework [5] (see specific documentation on workflows[1]), CIPRES [6] and LONI Pipeline Environment [7].

3.3. Service invocation

See Figure 2b. The workflow engine is available externally to the science gateway, in a service. The science gateway controls the service through a specific interaction (d) that might be implemented as a web-service call (e.g. RESTful or SOAP), as a command-line or as any other method that offers a well-defined interface to the workflow engine. The workflow engine might be invoked either as a black box that completely masks the infrastructure and workflow activities, or as a white box that allows for some interaction with them. The workflow engine is responsible for controlling the tasks on the infrastructure (b) and for performing the required data transfers to execute them (c_2). User data is usually managed through the science gateway (c_1), 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 [8], Vine Toolkit [9], Virtual Imaging Platform [10], the WS-PGRADE/gUSE framework [11] and the numerous science gateway

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

instances that rely on WS-PGRADE/gUSE [12]. Figure 3 shows the architecture of the Virtual Imaging Platform, where the MOTEUR workflow engine [13] is invoked as a service at step `3` (interaction `d` on Figure 2). Step `2` on Figure 3 corresponds to interaction a_1, and step `4` maps to interactions `b` and c_2 (in VIP, workflow data transfers are performed by the tasks and embedded in their descriptions).

3.4. Sub-tasking

See Figure 2c. The workflow engine is a particular task that can submit sub-tasks to the science gateway through interaction `e`. The workflow engine keeps track of the dependencies between the sub-tasks but their execution is delegated to the science gateway that executes them on the infrastructure through interaction `b`. Although the science gateway 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 science gateway may also implement mechanisms to facilitate the handling of task dependencies, for instance basic dependency lists as available in most batch managers (see for instance attribute `depend` of option `-W` in Torque[2]).

The science gateway also transfers both user and workflow data across the infrastructure, so that interactions c_1 and c_2 are both covered by `c`. In practice, both c_1 and c_2 can be implemented using the same pieces of code.

The sub-tasking architecture is implemented in CBRAIN [14] where it is used to integrate the PSOM workflow engine [15] and the FSL toolkit. The CBRAIN-PSOM integration [16] is illustrated on Figure 4. On this Figure, the science gateway is represented by components `CBRAIN portal` and `CBRAIN execution server` and the infrastructure consists of `Storage servers` and `Computing cluster with shared file system`. Interaction `b` on Figure 2 is implemented by steps `4`, `5`, `7` and `8`. Interaction `c` is implemented by steps `3` and `10`. Interaction `e` is implemented by step `6`. The PSOM workflow engine adopts a pilot-job architecture [17] where the a master coordinates workflow execution by submitting workers and establishing direct communication channels with them. Note how this peculiar execution model is totally supported by the sub-tasking architecture.

[2] `http://docs.adaptivecomputing.com`

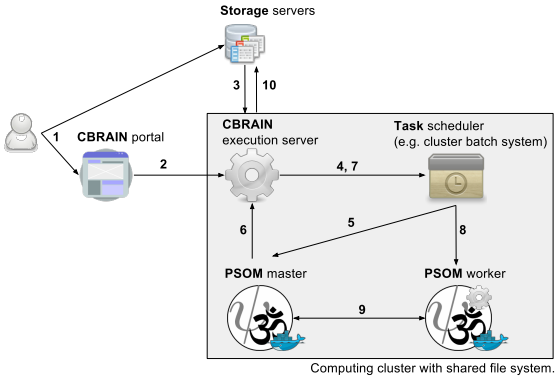


Figure 4: Architecture of the CBRAIN-PSOM integration that illustrates the sub-tasking model (Figure extracted from [16]). 1: User sends data and pipeline execution request to storage server(s) and CBRAIN portal. 2: CBRAIN portal sends execution request to execution server on cluster. 3: Execution server transfers data from storage server(s). 4-5: Execution server starts PSOM master via task scheduler. 6: PSOM master submits PSOM workers to execution server. 7-8: Execution server starts PSOM workers through task scheduler. 9: PSOM master and workers execute pipeline. 10. Execution server transfers results to storage server(s). [*Improve graphics*]

The CBRAIN-FSL integration is also remarkable as it allows to leverage FSL pipelines that are written in a very low-level workflow language (Linux executables and script) that submits tasks uniformly through a specific tool called `fsl_sub`. To integrate FSL in CBRAIN, `fsl_sub` was modified to implement interaction `e`[3] [*Add a diagram and maybe a code excerpt to illustrate that.*] .

3.5. Pool model

See Figure 2d. 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 [18].

3.6. Nested workflows

See Figure 2e. In nested workflows (Figure 2e-Left), an activity of a *parent* workflow executed by the *parent* engine is itself a *child* workflow that

[3] See `fsl_sub` modified script available at `https://github.com/aces/cbrain-plugins-neuro`

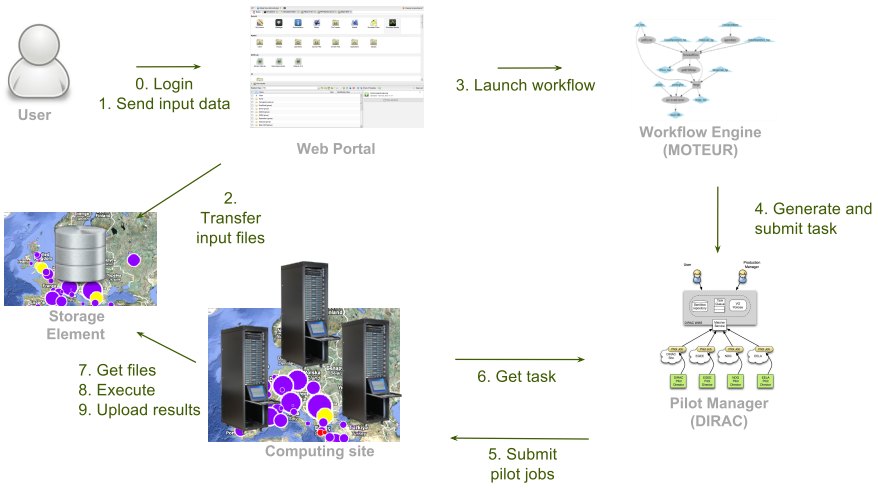


Figure 3: Architecture of the Virtual Imaging Platform that integrates the MOTEUR workflow engine through service invocation.

is executed by a *child* engine. The parent and child engines might use different workflow description languages. They might also execute workflows on different infrastructures. The parent workflow is also called meta-workflow. The science gateway communicates with the parent engine through interaction $*_1$. The science gateway also communicate with the infrastructure to transfer user data through abstract interactions $*_2$ and $*_3$. Both workflow engines communicate with the infrastructure through abstract interactions as well, $*_4$ and $*_5$. The parent engine communicates with the child engine through abstract interaction $*_6$. Administrator installs workflows in the science gateway through interaction $\mathtt{a}_2$, and installs software on infrastructures through interactions $\mathtt{a}_1$.

Nested workflows are abstract architectural patterns that can be instantiated in the various architectures described previously. We focus on instantiation with the service invocation model(see Figure 2e-Right) as this is the most used architecture. In such an instantiation, we assume that the parent and child workflow engines are distinct pieces of software that require different workflow services invoked by distinct `d` interactions. If this is not the case, then workflow services can be collapsed in a single one with a `d` interaction with itself. Workflow engines communicate with infrastructures using `b` and $\mathtt{c}_2$. Science gateway transfer user data to infrastructures using $\mathtt{c}_1$ interactions.

Nested workflows have long been available in workflow engines, for instance in the Taverna workbench [19]. They were also used by the SHIWA Science Gateway to implement so-called Coarse-Grained workflow interoperability [20], i.e. to integrate various workflow engines in a consistent platform. Figure 5 shows the architecture used in the SHIWA Science Gateway for nested workflow execution with service invocation as represented in Figure 2e. The parent workflow engine is WS-PGRADE, invoked as a service in the Science Gateway (step `1` on Figure 5, interaction `d` on Figure 2e). Ten different child engines can be used by nested workflows, invoked through the `Submission service` (step `2`, interaction `d`). Each of these engines can submit jobs and transfer data to a distributed computing infrastructure (DCI, step `3`, interactions `b` and $\mathtt{c}_2$). Data interactions (`c`) and application porting ones (`a`) are not represented on Figure 5.

3.7. Workflow import

See Figure 2f. 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 was implemented in the SHIWA Science Gateway through the IWIR language that provided a common language for portability across grid workflow systems [21].

[workflow import is an offline process.]

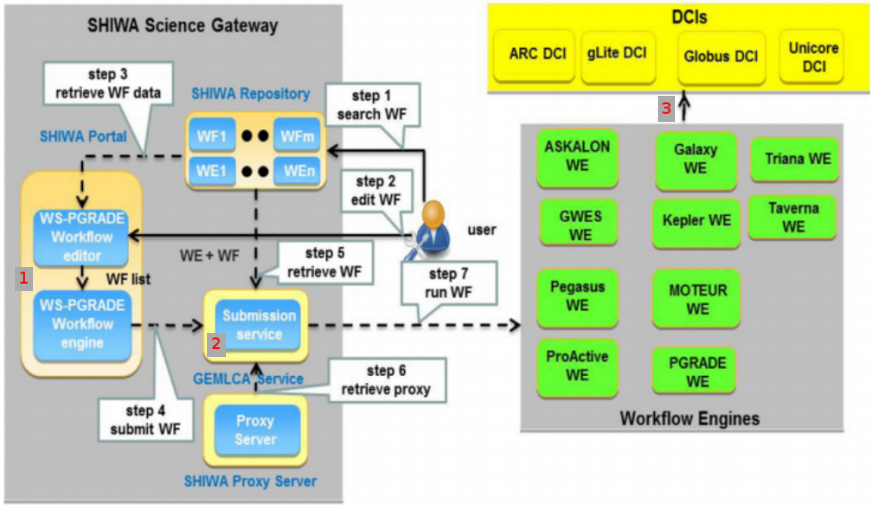


Figure 5: Nested workflow execution through SHIWA Science Gateway (Figure reproduced from [20] with permission of the first author).

4. 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.

4.1. Evaluation metrics

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.*]

[*Refer to the metric codes.*]

Integration measures the development effort required to build the architecture, assuming that the science gateway and infrastructure are already integrated. It is measured by counting on the architectural diagram the number of interactions and components to develop or modify. It breaks down to 4 metrics that quantify the developments related to:

- Science gateway ($\mathtt{I}_1$),
- Workflow engine ($\mathtt{I}_2$),
- Infrastructure ($\mathtt{I}_3$),
- Other components ($\mathtt{I}_4$).

Interactions are counted only once, in the component from which they originate.

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). Fine-grained information about workflow activity is required to properly troubleshoot workflow executions.

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. Modification of an interaction is deemed necessary when the parameters involved in this interaction are modified. Modification of a component is required when its code needs to be modified or recompiled (science gateway or workflow service), or when a new piece of software has to be installed (infrastructure only). Adding a new type of workflow engine allows to execute

the workflows that have been described in the language(s) used by the new engine.

- 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. We assume that different versions of a workflow engine share the same interface, i.e. they can be invoked using the same software. When this is not the case, the different versions have to be considered as different engine types.
- 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; 2: not possible).
- Elastic engines (0: easy to implement; 1: difficult to implement; 2: not available).
- Distributed engines (0: possible; 1: not possible).
- Task scheduling (0: difficult; 1: more difficult).

4.2. Tight integration

Integration. Tightly integrating a workflow engine in a science gateway requires to modify the science gateway ($\texttt{I}_1$=1) and to develop interactions `b`, and $\texttt{c}_2$ in the workflow engine ($\texttt{I}_2$=2). No other modification is required ($\texttt{I}_3$=$\texttt{I}_4$=0).

Robustness. The science gateway and the infrastructure are involved in a workflow execution ($\texttt{R}_1$=2), with interactions `b`, $\texttt{c}_1$ and $\texttt{c}_2$ ($\texttt{R}_2$=3). Debugging is not an issue since the science gateway can retrieve any information from the workflow engine directly ($\texttt{R}_3$=0).

Modularity. Integrating a new type of workflow engine requires to modify the science gateway as well as interactions `b` and $\texttt{c}_2$ ($\texttt{M}_1$=3). Updating a workflow engine version requires modifications in the science gateway ($\texttt{M}_2$=1). Adding a new infrastructure generates updates in interactions `b`, $\texttt{c}_1$ and $\texttt{c}_2$ ($\texttt{M}_3$=3). Inserting a new workflow is done through interaction `a` ($\texttt{M}_4$=1). Meta-workflows are not supported by default ($\texttt{M}_4$=1).

Scalability. Adding a new engine instance requires a new instance of the science gateway, which is in general not possible since load-balancing mechanisms available in web servers are usually meant to deal with short web requests rather than long workflow executions ($\texttt{S}_1$=2). Coherently, elastic engines are usually not possible either ($\texttt{S}_2$=2). Distributed engines are not available by default ($\texttt{S}_3$=1). The scheduling of tasks on the infrastructure is as complex as in any other architecture since the workflow engine might implement any kind of scheduling policy ($\texttt{S}_4$=0).

4.3. Service invocation

Integration. Integrating a workflow engine as a service requires to develop 3 interactions and 1 component. First, a workflow service has to be developed and interfaced with the infrastructure through interactions `b` and $\texttt{c}_2$ ($\texttt{I}_2$=3). In addtion, the science gateway has to be extended to call the engine interface `d` ($\texttt{I}_1$=1).

Robustness. Workflow execution requires 3 components: the science gateway, the workflow service, and the infrastructure ($\texttt{R}_1$=3). It involves the 4 interactions `b`, c_1, c_2 and `d` ($\texttt{R}_2$=4). Fine-grained debugging information is usually easy to obtain since the workflow service provides direct access to the engine ($\texttt{R}_3$=0).

Modularity. Adding a new type of workflow engine requires to implement the corresponding workflow service, to modify interaction `d`, and to implement interactions `b` and c_2 ($\texttt{M}_1$=4). New engine versions can be added by updating the workflow service without modifying any interaction ($\texttt{M}_2$=0). Adding a new type of infrastructure requires updates in interactions `b`, c_1 and c_2 ($\texttt{M}_3$=3). New workflows are added in the science gateway or in the workflow engine through interaction `a` only ($\texttt{M}_4$=1). Meta-workflow are not supported by default ($\texttt{M}_5$=1).

Scalability. The service architecture supports multiple engine instances through multiple workflow services. In VIP for instance, this feature has been available from release 1.17 (April 2016). A basic load-balancing mechanism is available that sends new workflow executions to the engine instance that has the least active executions. To avoid "black-hole" syndromes created by failing engine instances, engine instances are automatically disabled when workflows cannot be submitted to them. Adding a new engine instance, however, requires manual intervention to declare the new instance in the science gateway (S_1=1). Consequently, elastic engines are difficult to implement because they require a mechanism to update the science gateway configuration when a new engine instance is available (S_2=1). Distributing the execution of a single workflow in multiple engines is usually not possible unless the workflow engine has specific abilities (S_3=1). The scheduling of tasks on the infrastructure is as complex as in any other architecture since the workflow engine might implement any kind of scheduling policy (S_4=0).

4.4. Sub-task

Integration. Integrating a workflow engine as a sub-task only requires to implement interaction `e` (I_2=1) and to install the workflow engine on the infrastructure (I_3=1). A new task type is created in the science gateway only when a new workflow has to be integrated (see M_4), but this is not required to integrate the engine itself (I_1=0).

Robustness. Only 2 components and 3 interactions are involved in workflow execution (R_1=2, R_2=3). Obtaining fine-grained information about workflow activities is not straightforward since the science gateway has no knowledge about the workflow topology, and the workflow engine is integrated as a task (R_3=1).

Modularity. Integrating a new type of workflow engine requires to develop interaction `e` and to install the engine on the infrastructure (M_1=2). Updating an engine version requires updates only on the infrastructure (M_2=1). Adding a new infrastructure requires to update interactions `b` and `c` in the science gateway (M_3=2). New workflows are integrated by creating a new task in the science gateway through interaction `a` (M_4=1), and meta-workflows are not available (M_5=1).

Scalability. New engine instances are spawned and executed on the infrastructure as any other task upon user submission (S_1=0, S_2=0). Such scalability properties are one of the major interests of the sub-task architecture. Distributed engines are not supported by default (S_3=1). Task scheduling is slightly more complex than in the other approaches due to the special role of the task that executes the workflow engine (S_4=1). Indeed, the reliability of this task is critical since all the sub-tasks in the workflow depend on it and, depending on the recovery capabilities of the workflow engine, may need to be resubmitted if the workflow task fails. The workflow task is also longer than all its sub-tasks, which increases its chances of failure. In addition, task parameters, for instance estimated walltime, are more difficult to estimate for the workflow task than for sub-tasks which may generate issues such as selection of wrong batch queues on clusters. Finally, the interdependencies between the workflow task and its sub-tasks may create deadlocks when there is contention. Say for instance that only 1 computing resource is available for the science gateway and that the workflow task is running on it and submits sub-tasks, then the sub-tasks could only execute when the resource is available, which will never happen because the workflow task will not complete until the sub-tasks complete. This configuration can be generalized to an infrastructure with n resources where n workflows are submitted. In practice, however, the number of submitted workflows usually remains lower than the number of computing resources available on this infrastructure, which makes such deadlocks unlikely to happen.

4.5. Pool

Integration. Integrating workflow engines through the pool model requires interaction `d` in the science gateway (I_1=1), interactions `b` and c_2 in the workflow engine (I_2=2), the development of the workflow pool, agent and interaction `f` between them (I_4=3).

Robustness. The science gateway, workflow pool, agent and infrastructure are involved in a workflow execution (R_1=4), and these components are connected through 4 interactions (R_2=4). Accessing debugging information is not likely to be an issue since the workflow pool could implement specific functions for that (R_3=0).

Modularity. Adding a new engine type requires to wrap the engine in the agent and to update interactions `b` and c_2 (M_1=3). Updating the version of

an engine is transparent (M_2=0), and integrating a new infrastructure requires updates in interactions b, c_1 and c_2 (M_3=3). Integrating a new workflow is done through interaction a only (M_4=1) and meta-workflows are not available by default (M_5=1).

Scalability. New engine instances only require new agents, which is easily automated (S_1=0) and by design very suitable for elastic computing (S_2=0). For instance, auto-scaling rules can be implemented to start new agents when the workload in the science gateway exceeds a certain threshold [*ref needed*] . Distributed engines are not available by default (S_3=1) and task scheduling is as complex as in any other architecture (S_4=0).

4.6. Nested workflows with service invocation

Integration. Setting up a nested workflow architecture with service invocation requires interaction d in the science gateway (I_1=1), and the development of 2 workflow services with 2 instances of interactions b and c_2 (I_2=5).

Robustness. Workflow execution involves the science gateway, 2 workflow services and the infrastructure (R_1=4), which includes 7 interactions (R_2=7). Debugging is difficult because the science gateway cannot directly access fine-grained information in the sub-workflow (R_3=1).

Modularity. Adding a new type of *parent* engine requires to implement the corresponding service, to implement interactions b and c_2 in the parent engine, and to implement interaction d in the science gateway and in the parent service (M_1=5). Adding a new type of *child* engine only requires to implement the corresponding service, to develop interactions b and c_2 in the child engine, and to implement interaction d in the parent service (M_1=4). We use M_1=4.5 in Table 1 to reflect both conditions. Adding a new version in the parent or child engine only requires modifying this engine (M_2=0). Adding a new infrastructure requires to re-implement interactions b and c_2 twice, and interaction c_1 once (M_3=5). Adding a new workflow is done through interaction a (M_4=1). Meta-workflows are possible, which is one of the main interest of this architecture (M_5=0).

Scalability. As in the service architecture, adding a new workflow instance requires manual configuration in the science gateway (instance of a parent engine), or in the parent engine (instance of a child engine) (S_1=1). Similarly, elastic engines are difficult to achieve (S_2=1). Distributed engines are possible, through meta-workflows (S_3=0). Task scheduling is more complex than in other architectures though, due to the fact that workflow execution is split in different engines (S_4=1).

4.7. Workflow import with service invocation

Integration. Compared to the service architecture, workflow import requires an additional interaction g in the science gateway (I_1=2, I_2=3).

Robustness. Since workflow conversion is not involved in the execution (it is an offline process), metrics are as in the service architecture (R_1=3, R_2=4, R_3=0).

Modularity. Since adding a new type of workflow engine aims at supporting more workflows, we consider that it only requires to re-implement interaction g in this architecture. Note, however, that implementing interaction g can require very substantial work depending on the complexity of the language used by the new engine (M_1=1). Based on the same logic, adding a new engine version only requires modifying interaction g (M_2=1). As in the service architecture, interfacing with a new infrastructure requires modifications in the workflow service and in interactions b and c_2 (M_3=3). Adding a new workflow is done through interactions a and g (M_4=2). Meta-workflows are available after import, by connecting workflows in the language used in the science gateway (M_5=0).

Scalability. Since workflow conversion is not involved in the execution (it is an offline process), metrics are as in the service architecture (S_1=1, S_2=1, S_3=1, S_4=0).

5. Discussion

Comparison between architectures, per criterion (robustness, etc).

This ignores pre-existing work (e.g. a workflow engine is already available as a service). Migration across architectures too.

These architectures can be combined (give examples).

Metric	Tight	Service	Sub-task	Pool	Nested	Import
Integration						
Science gateway – I_1	1	1	0	1	1	2
Workflow engine – I_2	2	3	1	2	5	3
Infrastructure – I_3	0	0	1	0	0	0
Other – I_4	0	0	0	3	0	0
Total	**3**	**4**	**2**	**6**	**6**	**5**
Robustness						
Components – R_1	2	3	2	4	4	3
Interactions – R_2	3	4	3	4	7	4
Debugging – R_3	0	0	1	0	1	0
Total	**5**	**7**	**6**	**8**	**12**	**7**
Modularity						
New engine type – M_1	3	4	2	3	4.5	1
New engine version – M_2	1	0	1	0	0	1
New infrastructure – M_3	3	3	2	3	5	3
New workflow – M_4	1	1	1	1	1	2
Meta-workflow – M_5	1	1	1	1	0	0
Total	**9**	**9**	**7**	**8**	**10.5**	**7**
Scalability						
New engine instance – S_1	2	1	0	0	1	1
Elastic engines – S_2	2	1	0	0	1	1
Distributed engines – S_3	1	1	1	1	0	1
Task scheduling – S_4	0	0	1	0	1	0
Total	**5**	**3**	**2**	**1**	**3**	**3**
Grand total						
	22	**23**	**17**	**23**	**31.5**	**22**

Table 1: Architecture evaluation. Lower values (brighter colors) indicate better performance. [*update colors and find a more relevant way to compute a grand total without favoring criteria that have more metrics than the others (e.g. normalize each criterion between 0 and 1). Same comment for each sub-total. M1 might be the same as integration. Update nested workflow as infrastructures were separated.*]

The difficulty of implementing interaction g is not properly reflected. For instance, fsl uses bash, which would be very difficult to convert.

6. Conclusion

7. Acknowledgments

[FLI-IAM, Labex PRIMES, Ludmer Centre, whatever grant is funding POQ for integrating CBRAIN with PSOM.]

We warmly thank Rafael Ferreira da Silva for implementing the VIP platform and creating Figure 3.

References

[Review use of capitals in references]

[1] S. Reynaud, Uniform access to heterogeneous grid infrastructures with jsaga, in: Production grids in Asia, Springer, 2010, pp. 185–196.

[2] Á. Hajnal, Z. Farkas, P. Kacsuk, Data avenue: remote storage resource management in ws-pgrade/guse, in: Science Gateways (IWSG), 2014 6th International Workshop on, IEEE, 2014, pp. 1–5.

[3] S. Olabarriaga, T. Glatard, A. Hoheisel, A. Nederveen, D. Krefting, Crossing HealthGrid Borders: Early Results in Medical Imaging, in: HealthGrid'09, Berlin, 2009.

[4] K. Plankensteiner, J. Montagnat, R. Prodan, IWIR: A Language Enabling Portability Across Grid Workflow Systems, in: Workshop on Workflows in Support of Large-Scale Science(WORKS'11), , Seattle, USA, 2011.
URL http://hal.archives-ouvertes.fr/hal-00677832/document

[5] 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

[6] 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.

[7] 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).

[8] 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.

[9] 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).

[10] 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.

[11] 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

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

[13] T. Glatard, J. Montagnat, D. Lingrand, X. Pennec, Flexible and efficient workflow deployment of data-intensive applications on grids with moteur, Journal of High Performance Computing and Applications 22 (3) (2008) 347–360.
URL http://rainbow.polytech.unice.fr/publis/glatard-montagnat-etal:2008.pdf

[14] 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.

[15] P. Bellec, S. Lavoie-Courchesne, P. Dickinson, J. Lerch, A. Zijdenbos, A. C. Evans, The pipeline system for octave and matlab (psom): a lightweight scripting framework and execution engine for scientific workflows, Frontiers in neuroinformatics 6 (2012) 7.

[16] 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

[17] M. Turilli, M. Santcroos, S. Jha, A comprehensive perspective on the pilot-job abstraction, arXiv preprint arXiv:1508.04180.

[18] 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

[19] 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.

[20] G. Terstyanszky, T. Kukla, T. Kiss, P. Kacsuk, Á. Bal-

askó, Z. Farkas, Enabling scientific workflow sharing through coarse-grained interoperability, Future Generation Computer Systems 37 (2014) 46–59.

[21] 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