Software architectures to integrate workflow engines in science gateways

Tristan Glatard[a,b], Marc-Étienne Rousseau[a], Sorina Camarasu-Pop[b], Reza Adalat[a], Natacha Beck[a], Samir Das[a], Rafael Ferreira da Silva[d], Najmeh Khalili-Mahani[a], Vladimir Korkhov[f], Pierre-Olivier Quirion[c], Pierre Rioux[a], Sílvia D. Olabarriaga[e], Pierre Bellec[c], Alan C. Evans[a]

[a]*McGill Centre for Integrative Neuroscience, Montreal Neurological Institute, McGill University, Canada.*
[b]*University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France.*
[c]*Centre de Recherche de l'Institut de Gériatrie de Montréal CRIUGM, Montréal, QC, Canada.*
[d]*University of Southern California, Information Sciences Institute, Marina del Rey, CA, USA.*
[e]*Academic Medical Center of the University of Amsterdam, Department of Clinical Epidemiology, Biostatistics and Bioinformatics, Amsterdam, NL.*
[f]*St. Petersburg State University, Russia.*

Abstract

Workflow engines are critical for the efficient and transparent exploitation of distributed infrastructures in the ecosystem of tools and services offered by science gateways. We investigate six software architectures commonly used to integrate workflow engines into science gateways. In *tight integration*, the workflow engine shares software components with the science gateway. In *service invocation*, the engine is isolated and invoked through a specific software interface. In *task encapsulation*, the engine is wrapped as a computing task executed on the infrastructure. In the *pool model*, the engine is bundled in an agent that connects to a central pool to fetch and execute workflows. In *nested workflows*, the engine is integrated as a child process of another engine. In *workflow conversion*, the engine is integrated through workflow language conversion. We describe and evaluate these architectures with metrics for assessment of integration complexity, robustness, extensibility, scalability and support for meta-workflows and fine-grained debugging. Tight integration and task encapsulation are the easiest to integrate and the most robust. Extensibility is equivalent in most architectures. The pool model is the most scalable one and meta-workflows are only available in task encapsulation, nested workflows and workflow conversion. These results provide insights for science gateway architects and developers.

Keywords: Workflow engines, science gateways, software architectures.

1. Introduction

Workflow engines are critical for the efficient and transparent exploitation of distributed infrastructures in the ecosystem of tools and services offered by science gateways. Several software architectures can be adopted to integrate workflow engines in science gateways, with important consequences on the development effort required and resulting system.

This paper describes, provides examples and compares such architectures, based on system-independent representations of their main components and interactions. It is informed by our experience in the development and sustained operation of the CBRAIN [40] and VIP [16] science gateways for medical image analysis during the past 7 years, as well as by lessons learned from several science gateway and workflow projects such as SHIWA[1] and ER-flow[2]. In this section we provide background information and definitions of workflow engines, science gateways and infrastructures. In Section 2, we describe six architectures that cover the main patterns that we encountered in the various systems that we studied during the last few years. The architectures are described within a consistent framework that underlines the functional interactions between their main software components. In Section 3, the different architectures are evaluated with metrics measuring integration complexity, robustness, extensibility, scalability and other specific

[1]`http://www.shiwa-workflow.eu`
[2]`http://www.erflow.eu`

features. These metrics are specifically designed to measure the ability of the architectures to address workflow-related issues commonly encountered in science gateways. We finally compare the architectures and highlight recommendations for science gateway architects and developers.

1.1. Workflow engines

In the last decade, the e-Science community has developed workflow systems to help application developers access distributed infrastructures such as clusters, grids, clouds and web services. These efforts resulted in tools among which Askalon [15], Hyperflow [5], MOTEUR [17], Pegasus [11, 12], Swift [49], Taverna [31], Triana [42], VisTrails [8], WS-PGRADE [25] and WS-VLAM [46]. Such workflow engines usually describe applications in a high-level language with specific data and control flow constructs, parallelization operators, visual edition tools, links with domain-specific application repositories, provenance recording and other features. An overview of workflow system capabilities is available in [10].

At the same time, toolboxes have been emerging in various scientific domains to facilitate the assembly of software components in consistent "pipelines". In neuroimaging, our primary domain of interest, tools such as Nipype (Neuroimaging in Python, Pipelines and Interfaces [21]), PSOM (Pipeline System for Octave and Matlab [6]), PMP (Poor Man's Pipeline [2]), RPPL [50], SPM (Statistical Parametric Mapping [4]) and FSL (FMRIB Software Library [23]) provide abstractions and functions to handle the data and computing flow between processes implemented in a variety of programming languages. Such tools were interfaced to computing infrastructures, in particular clusters, to execute tasks at a high throughput. Some of these tools also support advanced features such as provenance tracking, or redundancy detection across analyses to avoid re-computation. A wide array of workflows have been implemented using these pipeline systems and are now shared across neuroimaging groups world-wide, which represents a tremendous opportunity for science gateways to leverage. Domain-specific engines nicely complement e-Science systems that are more oriented towards the exploitation of distributed computing infrastructures, in particular grids and clouds.

In this paper, a *workflow engine* (also abbreviated *engine*) is a piece of software that submits interdependent computing *tasks* to an *infrastructure* (local server, cluster, grid or cloud) based on a workflow description (a.k.a *workflow*), using input data that may consist of files, database entries or simple parameter values. Although simplistic, this definition covers both e-Science workflow systems and domain-specific pipeline systems. Some workflow engines, usually from the e-Science community, may transfer data across the infrastructure, and others, usually domain-specific ones, may leave this role to external processes. Workflows may be expressed in any language, including high-level XML or JSON dialects such as Scufl or Hyperflow, and low-level scripting languages such as Bash.

1.2. Science gateways

Science gateways are used to share resources within a community and to provide increased performance and capacity through facilitated access to storage and computing power. They are often accessible through a web interface that helps users manage access rights, data transfers, task execution, and authentication on multiple computing and storage locations. Workflow engines are part of this ecosystem as core components to implement and execute applications.

Various science gateways have been developed, including frameworks such as Apache Airavata [29], the Catania Science Gateway Framework [3] and WS-PGRADE/gUSE [25]. Numerous science gateways were built using such frameworks [24, 3] or as standalone systems [40, 16]. Most of these systems include one or several workflow engines.

Integration between workflow engines and science gateways varies across systems. Some science gateways are tailored to a particular engine, while others are more general and host applications executed by different types of engines.

Extensibility is an important property of the integration. New workflows are added frequently, different types and versions of workflow engines may be integrated over time, and different kinds of infrastructure can be targeted.

Scalability is also a crucial concern for such multi-user, high-throughput systems. For this purpose, science gateways may balance the load among different instances of the same engine, start new engines elastically using auto-scaling techniques such as the ones reviewed in [27], and use advanced task scheduling policies on the infrastructure to improve performance, fault-tolerance and fairness among users.

Robustness is highly desirable as well since it is key to a good user experience. Simple architectures facilitate the implementation effort towards robust interactions which, in turn, have a positive impact on characteristics such as gateway predictability, transparency, reliability, traceability and reproducibility.

Other specific features may also be available, for instance data vizualisation and quality control, workflow edition, debugging instruments, or social tools among users.

1.3. Infrastructure

Infrastructure consists of the computing and storage resources involved in workflow execution, as well as the software services used to access these resources. Infrastructure can be composed of computing or file servers, databases, clusters, grids or clouds. Some workflow engines and science gateways may require specific characteristics, such as the presence of a shared file system between the computing nodes, the availability of a global task meta-scheduler, the presence of a file catalog, etc. In the analysis presented below, such specific requirements are not discussed. Instead, we consider the infrastructure as an abstract system that can execute tasks and store data regardless of the enabling mechanisms.

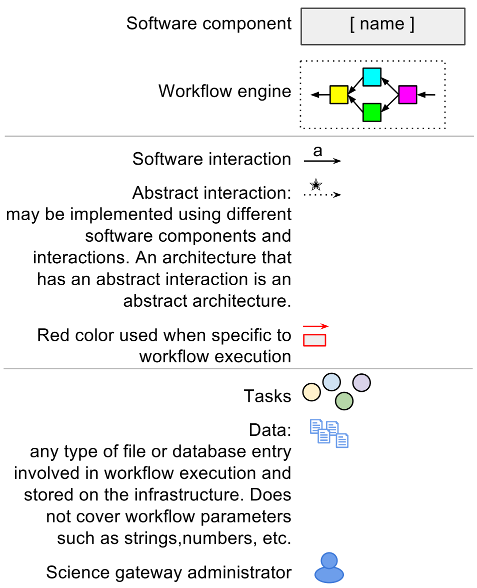


Figure 1: Graphical notations

2. Architectures

Architectures to integrate workflow engines in science gateways are presented in Figure 2 using the graphical notations defined in Figure 1. Table 1 summarizes the classification of a few systems by architecture.

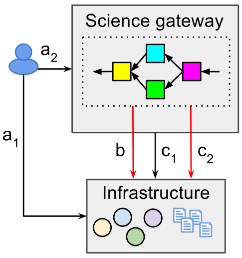
2.1. Interactions

The interactions involved in the architectures are described below and labeled (`a`,`b`,`c`, ...) as in Figure 2.

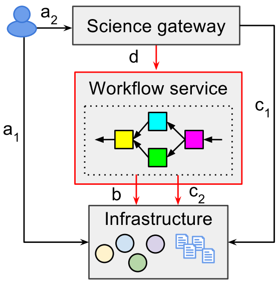
`(a)` Workflow integration: consists in adding a new workflow to the system so that users can execute it. It is triggered by an administrator of the science gateway and it results in an interface, for instance a web form, where users can enter the parameters of the workflow to be executed. The interaction has two steps: ($\mathtt{a}_1$) the programs used in the workflow are installed on the infrastructure or prepared for on-the-fly installation, which may require specific privileges; ($\mathtt{a}_2$) the workflow is 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 may for instance be achieved using libraries that implement standards such as SAGA (Simple API for Grid Applications [20]), DRMAA (Distributed Resource Management Application API [44]), or OCCI (Open Cloud Computing Interface [14]).

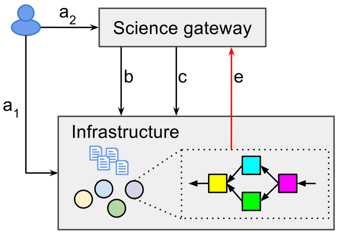
`(c)` Data control: operations to manage data on the infrastructure, such as: upload, download, deletion, browsing, replication, caching, etc. Data movements can be triggered by the user in the science gateway (c_1), to upload input data or download processed data. They can also be performed by the workflow engine (c_2), to transfer workflow data (inputs, outputs, or intermediate results) across the infrastructure so that tasks can use it. The infrastructure might offer various data storage backends with heterogeneous interfaces. Tools and services such as JSAGA [35] or Data Avenue [22] can be used to homogeneize these interfaces.



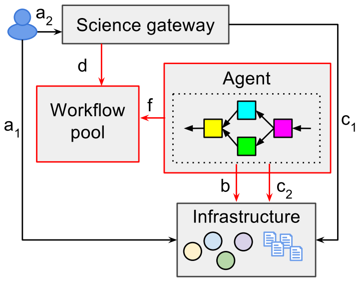
(I) Tight integration



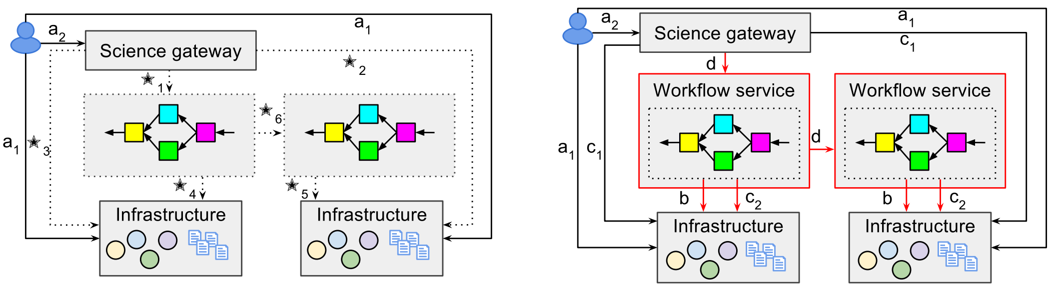
(II) Service invocation



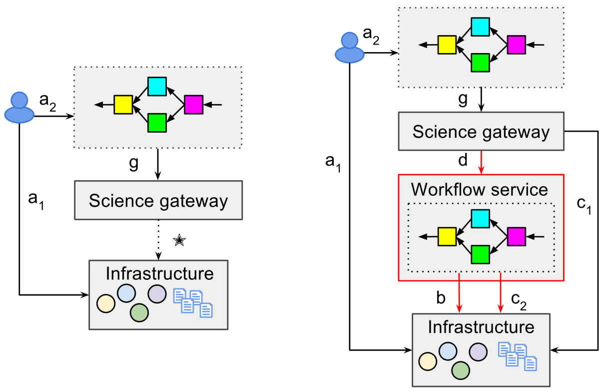
(III) Task encapsulation



(IV) Pool



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



(VI) Workflow conversion. Left: abstract model. Right: instantiation with service invocation.

Figure 2: Architectures

Architecture	Systems
Tight	Catania Science Gateway Framework [3], Distributed application runtime environment (DARE [28]), DECIDE [3]^{c,n}, LONI Pipeline Environment^{n} [13]
Service	Apache Airvata [29], Neuroscience Gateway [37]^{w,n}, HubZero with Pegasus [30], MoSGrid [26]^{w}, System in [47], Vine Toolkit [41], Virtual Imaging Platform [16]^{n}, WS-PGRADE/gUSE framework [25], Science gateways in [24]^{w}
Task	CBRAIN and PSOM [18]^{n}, CBRAIN and FSL^{n}
Pool	SHIWA pool [36]
Nested	SHIWA Simulation Platform (Coarse-Grained Interoperability [43])^{w}, HubZero with Pegasus (via hierarchical workflows) [12], Tavaxy [1].
Conversion	SHIWA Simulation Platform (Fine-Grained Interoperability) [34]^{w}, Tavaxy [1], system in [9].

Table 1: Classification of science gateways based on the architecture used to integrate workflow engines. ^{c}: based on the Catania Science Gateway Framework. ^{w}: based on WS-PGRADE/gUSE. ^{n}: used for neuroimaging.

(d) Workflow control: operations to control workflow execution in an engine, including: workflow submission, monitoring, termination, etc. Workflow control can be coarse-grained (black box) or fine-grained (white box). In a coarse-grained model, the various tasks created by a workflow execution 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, dependencies 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 of a sub-task is usually available and used for additional control. For instance, the parent task may wait for all its sub-tasks to complete before finishing, and conversely all the sub-tasks may be terminated if the parent task is killed.

(f) Pool-agent: specific to the pool architecture described in Section 2.5. 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. A similar type of interaction is used in pilot-job systems [45] and other agent-based computing models.

(g) Workflow conversion: translation from one workflow language into another. This interaction may not be available or possible for every workflow language. It has been developed mostly for translation between well-structured and relatively simple workflow languages such as GWorkflowDL and Scufl [32], and for translation among the 5 workflow systems in the SHIWA project: Askalon, MOTEUR, Taverna, Triana and WS-PGRADE.

2.2. Tight integration

See Figure 2(I). 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 model in a Ruby on Rails application. The workflow engine and the science gateway usually share a database where applications, 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 [3] (see specific documentation on workflows[3]), in the Distributed application runtime environment (DARE) [28], in Galaxy [19], and in the LONI Pipeline Environment [13]. Note that tightly integrated architectures may provide advanced workflow edition features which are not covered by our analysis.

2.3. Service invocation

See Figure 2(II). The workflow engine is available externally to the science gateway, as a service. The science gateway controls the service through a specific interaction (d) that might be implemented as

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

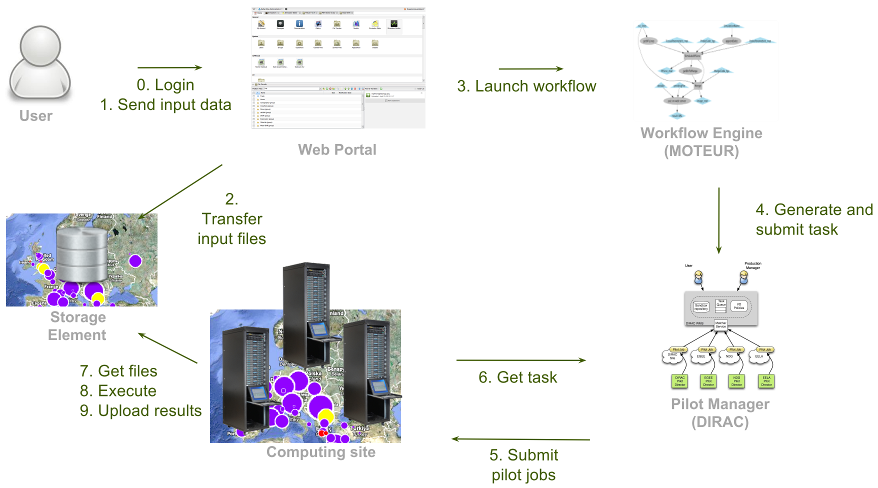


Figure 3: Architecture of the Virtual Imaging Platform (service invocation). Step 3 corresponds to interaction `d` in Figure 2(II), step 2 corresponds to interaction $\mathtt{c}_1$, step 4 maps to interaction `b`, and steps 7 and 9 map to $\mathtt{c}_2$ (in VIP, workflow data transfers are performed by the tasks and embedded in their descriptions).

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 tasks, 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 ($\mathtt{c}_2$). User data is usually managed through the science gateway ($\mathtt{c}_1$), although in some variations of the architecture (not represented in Figure 2) 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 [29], Vine Toolkit [41], Virtual Imaging Platform [16], the systems in [47, 38] and the WS-PGRADE/gUSE framework [25]. The integration between the HubZero science gateway and the Pegasus workflow engine performed in [30] also uses a service architecture, where the `d` interaction is implemented as a set of calls to Pegasus' command-line tools (e.g., pegasus-status, pegasus-plan, etc.). Figure 3 shows the architecture of the Virtual Imaging Platform, where the MOTEUR workflow engine [17] is invoked through a service.

2.4. Task encapsulation

See Figure 2(III). The workflow engine is wrapped in 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 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 cluster managers (see for instance attribute `depend` of option `-W` in Torque[4]).

The science gateway also transfers both user and workflow data across the infrastructure, so that interactions $\mathtt{c}_1$ and $\mathtt{c}_2$ are both covered by `c`.

Task encapsulation is implemented in CBRAIN [40] where it is used to integrate the FSL toolkit [23] and the PSOM workflow engine [6]. The CBRAIN-FSL integration allows leveraging FSL pipelines written in low-level workflow languages (Linux executables and scripts) that submit tasks uniformly through a specific tool called fsl_sub. This integration is shown in Figure 4(I), where 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`.

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

The CBRAIN-PSOM integration [18] is shown in Figure 4(II). The PSOM workflow engine adopts a pilot-job architecture [45] where a master coordinates workflow execution by submitting workers and establishing direct communication channels with them. Note how this peculiar execution model is well supported by task encapsulation.

2.5. Pool model

See Figure 2(IV). Workflows are submitted by the science gateway to a central pool through interaction `d`. Agents connect to the pool asynchronously to retrieve and execute workflows through interaction `f`. Agents may be started according to various policies, for instance to ensure load balancing. Workflow engine controls tasks and data on the infrastructure through interactions `b` and $\texttt{c}_2$, science gateway transfers user data through interaction $\texttt{c}_1$, and administrator installs workflows through interaction $\texttt{a}_1$ and $\texttt{a}_2$.

The pool model was implemented in the SHIWA pool [36] diagrammed in Figure 5, where agents can wrap different types of workflow engines to execute workflows expressed with different languages.

2.6. Nested workflows

See Figure 2(V). In nested workflows (Figure 2(V)-Left), a workflow engine is integrated as a *child* process of a *parent* workflow engine. Parent and child engines might use different languages and might run on different infrastructures. A parent workflow is also a meta-workflow. The science gateway communicates with the parent engine through abstract interaction $*_1$. The science gateway also communicates with the infrastructure to transfer user data through abstract interactions $*_2$ and $*_3$. Both workflow engines communicate with the infrastructure through abstract interactions $*_4$ and $*_5$. The parent engine communicates with the child engine through abstract interaction $*_6$. Administrator installs workflows through interactions $\texttt{a}_1$ and $\texttt{a}_2$.

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 (Figure 2(V)-Right) as this is the most used architecture. In the 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 into a single one with a `d` interaction with itself. An example of such interaction is the use of hierarchical workflows in Pegasus [12]. Workflow engines communicate with infrastructures using `b` and $\texttt{c}_2$. Science gateway transfers user data to infrastructures using $\texttt{c}_1$ interactions.

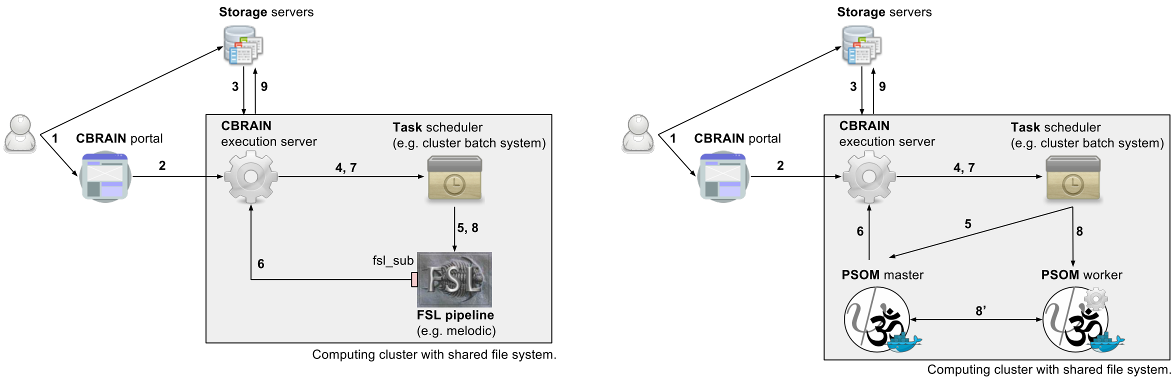
Nested workflows have long been available in workflow engines, for instance in the Taverna workbench [31]. They are also used implicitly in several platforms where workflow engines are wrapped in workflow tasks as any other command-line tool. Nested workflows were notably used by the SHIWA Science Gateway to implement so-called Coarse-Grained workflow interoperability [43], i.e. to integrate various workflow engines in a consistent platform. Figure 6 shows the architecture used in the SHIWA Simulation Platform for nested workflow execution with service invocation.

2.7. Workflow conversion

See Figure 2(VI). This is an abstract model instantiated with the service invocation architecture for consistency. Workflow engines are integrated in the science gateway through workflow format conversion from a native format to the science gateway format. Workflow conversion is usually an offline process that is not involved in workflow execution. A few systems have implemented workflow conversion. In the SHIWA Simulation Platform, it is implemented through the IWIR language, which provides a common language for portability across grid workflow systems [34] and allows conversion among n workflow languages using $2n$ interactions instead of n^2. Tavaxy [1] enables the import, merging and execution of Taverna [31] and Galaxy [19] workflows; when some workflow parts cannot be imported, workflows are run by Tavaxy as nested workflows using their native engine. The work in [9] describes workflow conversion from KNIME [7] to WS-PGRADE/gUSE and from Galaxy [19] to WS-PGRADE/gUSE.

3. Evaluation

The architectures described in Section 2 are evaluated in Table 2 using five main criteria: integration complexity, robustness, extensibility, scalability and other specific features. These are based on science gateway properties introduced in Section 1.2. Criteria break down to specific metrics where *lower value indicates better performance.*



(I) CBRAIN-FSL integration.

(II) CBRAIN-PSOM integration. Figure reproduced from [18].

Figure 4: CBRAIN architecture for workflow engine integration (task encapsulation). 1: User sends data and workflow 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 workflow engine (FSL tool or PSOM master) via task scheduler. 6: Workflow engine submits sub-tasks to execution server (FSL tasks or PSOM agents). 7-8: Execution server starts sub-tasks through task scheduler (FSL sub-tasks or PSOM workers). FSL sub-tasks will run locally instead of being submitted again to CBRAIN through interaction 6. 8': (PSOM only) PSOM master and workers execute workflow. 9. Execution server transfers results to storage server(s). Interaction `b` in Figure 2(III) is implemented by steps `4`, `5`, `7` and `8` (regular interactions with batch manager). Interaction `c` is implemented by steps `3` and `9`. Interaction `e` is implemented by step `6` (for FSL: through a modified version of the fsl_sub script available in `https://github.com/aces/cbrain-plugins-neuro`; for PSOM: through a specific development in PSOM).

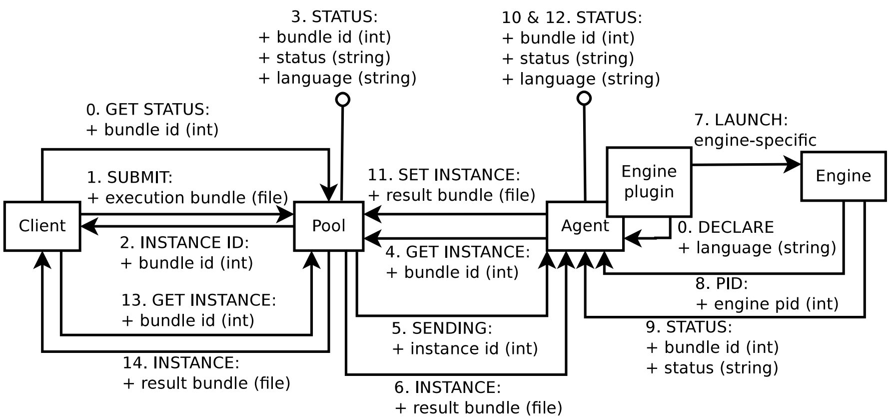


Figure 5: Architecture of the SHIWA pool. Circle-terminated arrows indicate messages that are broadcast to all pool clients. Interaction `d` of Figure 2(IV) is implemented by 3 different calls to the pool: workflow submission (`1` & `2`), workflow status retrieval (`0` & `3`), and workflow retrieval (`13` & `14`). Interaction `f` is implemented through 2 types of calls: workflow instance retrieval (`4`, `5` and `6`), and workflow instance update (`11` and `12`). Workflow instance retrieval is used by the agents to fetch work from the pool. Workflow instance update is used by the agents to update workflow statuses. Calls `0`, `7`, `8` and `9` are used by workflow engine plugins to declare their supported language and to launch engines, and by workflow engines to report their status to the agent. These calls are specific to the SHIWA Pool implementation of the `agent` component and therefore have no corresponding representation in Figure 2(IV). Figure reproduced from [36].

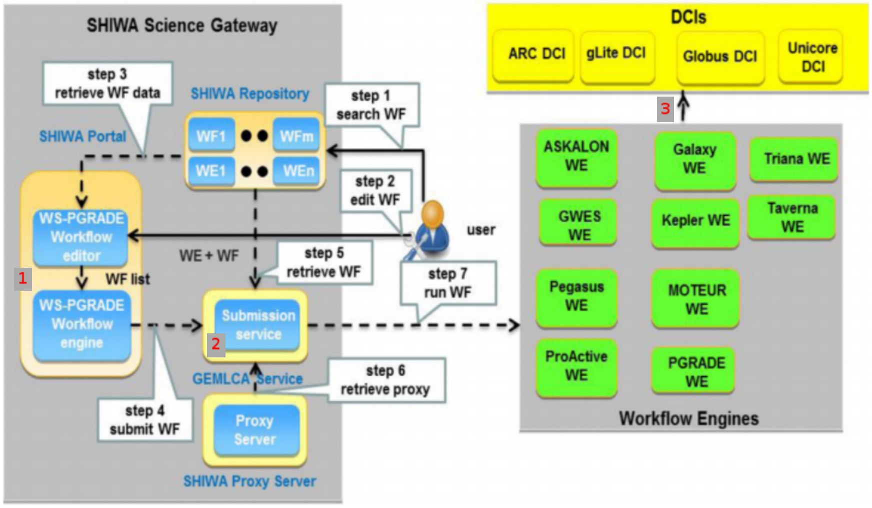


Figure 6: Nested workflow execution through the SHIWA Science Gateway. The parent workflow engine is `WS-PGRADE/gUSE`, invoked as a service in the Science Gateway (step `1`, interaction `d` on Figure 2(V)-Right). Ten different child engines can be used by nested workflows, invoked through the `Submission service` (step `2`, interaction `d`). Each of these engines can submit tasks and transfer data to a distributed computing infrastructure (`DCI`, step `3`, interactions `b` and $\mathtt{c}_2$). User data interactions ($\mathtt{c}_1$) and application porting ones (`a`) are not represented. Figure reproduced from [43] with permission of the first author.

Colors in Table 2 represent normalized scores, as explained in the Table caption. For each criterion, a global score is computed by summing up the individual scores for each metric. We ensure that the different metrics used to calculate the global score for a criterion are comparable, so that they can be combined meaningfully. The criteria and metrics are explained hereafter.

3.1. Evaluation metrics

Science gateway *integration complexity* is measured through the number of software components and their interactions. It breaks down to the following 2 metrics:

- Total components ($\mathtt{I}_1$): total number of components in the architecture.
- Total interactions ($\mathtt{I}_2$): total number of interactions in the architecture.

One may wonder whether infrastructure should be counted in $\mathtt{I}_1$, since it is usually an external entity that is not developed nor maintained by the groups who integrate workflow engines in science gateway. We support its inclusion here because integrating an infrastructure into the system usually requires some technical effort (e.g., account creation, software installation, APIs, etc.), therefore increasing complexity. The two metrics are summed to obtain a global score that measures the total number of software pieces affected by integration.

Robustness of workflow execution measures the likelihood that the workflow execution fails due to errors in the components or in the interactions of the software architecture. Errors coming from the infrastructure (e.g., unavailable data or terminated tasks) or workflows (e.g., wrong user input or application errors) are not covered, since they do not stem directly from the software architecture. Robustness is measured here as a consequence of global complexity, since complex architectures tend to be more prone to failure. More precisely, robustness is determined by the number of software components and interactions represented in red in the architecture diagrams in Figure 2, which are specific to workflow execution. Two metrics are used:

- Specific components ($\mathtt{R}_1$): number of specific components involved in workflow executions. The Science gateway component, for instance, is *not* specific to workflow execution since it is used to authenticate users, add new workflows, transfer user data, etc., and
- Specific interactions ($\mathtt{R}_2$): number of specific interactions involved in workflow executions. Data control (interaction `c`), for instance, is *not* specific to workflow execution since it is also used by the science gateway to transfer user data.

	Tight	Service	Task	Pool	Nested	Conversion
Integration complexity						
Total components – I_1	2	3	2	4	5	3
Total interactions – I_2	5	6	5	7	11	7
Total (total software pieces)	**7**	**9**	**7**	**11**	**16**	**10**
Robustness						
Specific components – R_1	0	1	0	2	2	1
Specific interactions – R_2	2	3	1	4	6	3
Total (execution software pieces)	**2**	**4**	**1**	**6**	**8**	**4**
Extensibility						
New engine type – E_1	3	4	2	3	4.5	1
New engine version – E_2	1	0	1	0	0	1
New workflow – E_3	2	2	2	2	2	3
New infrastructure – E_4	4	4	3	4	6	4
Total (difficulty to extend)	**10**	**10**	**8**	**9**	**12.5**	**9**
Scalability						
Multiple engine instances – S_1	2	1	0	0	1	1
Distributed engines – S_2	1	1	1	1	0	1
Task scheduling – S_3	0	0	1	0	1	0
Total (scalability issues)	**3**	**2**	**2**	**1**	**2**	**2**
Specific features						
Meta-workflow – O_1	1	1	0	1	0	0
Fine-grained debugging – O_2	0	1	1	1	1	1
Total (missing features)	**1**	**2**	**1**	**2**	**1**	**1**

Table 2: Architecture evaluation. Lower values (brighter colors) indicate better performance. Cell color is set as follows: (1) on each row, metric values are normalized between 0 (best value) and 1 (worst value): $m' = \frac{m - m_{\min}}{m_{\max} - m_{\min}}$ where m is the metric value, $m_{\min}$ and $m_{\max}$ are the minimum and maximum values for all architectures; (2) the RGB hexadecimal color code of the cell is `#99XX99`, where X=`round`(F-6m') (`round` rounds a number to the nearest integer).

R_1 and R_2 are summed to obtain the global score for this criterion, which measures the total number of software pieces that are specifically involved in the workflow execution.

Extensibility measures the difficulty to replace or add elements to the architecture. It is determined by the number of interactions and components that need to be modified when a new element is added. Modification of a component is required when its code needs to be updated or recompiled (science gateway or workflow service), or when a new piece of software has to be installed (infrastructure only). Modification of an interaction is deemed necessary when the parameters involved in this interaction are modified. Extensibility breaks down to 4 metrics depending on the type of element (engine, workflow or infrastructure) that has to be added or replaced:

- New engine (E_1): number of interactions or components that need to be modified to integrate a new type of workflow engine in the architecture. Workflow engines belong to different types when they cannot be invoked using the same implementation of the interactions.

- Engine version upgrade (E_2): number of interactions or components modified to integrate a new version of a workflow engine in the architecture, assuming that another version of the same engine type is already available. Different versions of a workflow engine can be invoked using the same interaction implementation(s). When this is not the case, the different versions are considered as different engine types. Components are not treated equally regarding engine version updates. Components whose only function is to host the workflow engine, i.e. workflow services and agents, are not counted in E_2 because updates in such components are assumed to be straightforward. On the contrary, modifications of components that have other functions than wrapping the engine, i.e. science gateway and infrastructure, are counted in E_2 because updates in these components require more effort, e.g. new release of the science gateway, gaining administrative privileges on the infrastructure, etc. In practice, E_2=0 when the workflow engine is wrapped in a service or in an agent, and E_2=1 when it is wrapped in another component.

- New workflow (E_3): number of interactions required to integrate a new workflow in the architecture. Adding a new workflow is a very common operation that does not require modifying software components or interactions, assuming that the engine type and version required to execute this workflow are already available. In most cases, adding a new workflow requires only interactions $\mathtt{a}_1$ and $\mathtt{a}_2$, but $\mathtt{g}$ is required as well when workflow conversion is used.

• New infrastructure ($\mathtt{E}_4$): number of interactions or components affected for the integration of a new type of infrastructure in the architecture. Adding a new infrastructure typically aims at providing more computing or storage power, enabling access to specific types of resources (e.g., GPUs, clouds), or enforcing execution policies (e.g., constrain data to remain in a particular network domain).

The four metrics are summed to obtain a global score that measures the difficulty to extend the architecture. Note that some extensions may involve several metrics in practice. For instance, adding a new type of engine may help integrating new infrastructures when interactions $\mathtt{b}$ and $\mathtt{c}_2$ are already present for the new engine and the infrastructure.

Scalability corresponds to the ability of an architecture to cope with high workloads through the following mechanisms: starting multiple engine instances, distributing engines and task scheduling. This is measured by assessing the potential scalability difficulties due to (partial) absence of a given feature in the architecture. Features are evaluated using a 3-level metric: 0 means that the feature is very easy to enable, 1 means that it can be implemented but with some difficulty, and 2 means that the feature cannot be implemented in a realistic scenario. Three different features are identified:

• Multiple engine instances ($\mathtt{S}_1$): measures the potential for a single gateway instance to use more than one engine instance simultaneously. Workflow engines may require important amounts of resources when several workflows, or large workflows, are executed. At some point, it may be required for the science gateway to distribute the load among several engines. $\mathtt{S}_1$=0 means that adding a new engine instance is inherently supported in the architecture. In this case, elastic engines can be implemented through some kind of auto-scaling mechanism to control the number of engine instances in the architecture. $\mathtt{S}_1$=1 means that multiple engine instances requires specific developments in the science gateway or workflow engine. $\mathtt{S}_1$=2 means that new engine instances cannot be added.

• Distributed engines ($\mathtt{S}_2$): measures the potential of distributing the execution *of a single workflow* among different engine instances. In our scope, this feature focuses on the capabilities of the architecture rather than these of the workflow engine. $\mathtt{S}_2$=0 means that distributed engines are enabled by the architecture, and $\mathtt{S}_2$=1 that they require specific developments in the workflow engine.

• Task scheduling ($\mathtt{S}_3$): measures the complexity added by the architecture to task scheduling on the infrastructure. Task scheduling typically depends more on the implementation of specific algorithms in the science gateway, workflow engine and infrastructure than on the architecture used to integrate the workflow engine in the science gateway. Some architectures, however, complicate the task scheduling problem by introducing additional software layers or creating tasks with specific characteristics. $\mathtt{S}_3$=0 means that the architecture does not add any additional complexity to the scheduling problem, and $\mathtt{S}_3$=1 otherwise.

These three metrics are summed to obtain a global measure of the scalability potential of the architecture.

Specific features include:

• Meta-workflow ($\mathtt{O}_1$): measures the ability to describe meta-workflows from existing workflows. Meta-workflows offer an additional level of flexibility to build workflows from reusable components. $\mathtt{O}_1$=0 means that meta-workflows are intrinsically enabled by the architecture (i.e. they can be implemented using the components and interactions already in place), amd $\mathtt{O}_1$=1 means that they may be implemented with some development effort.

• Fine-grained debugging ($\mathtt{O}_2$): availability of fine-grained debugging information about workflow tasks (white-box workflow). Fine-grained information about workflow tasks is required to properly troubleshoot workflow executions. Its availability and accuracy usually depends on the number of software layers between the science gateway and the workflow engine [33]. $\mathtt{O}_2$=0 means that the information is directly accessible in the science gateway, $\mathtt{O}_2$=1 means that it is obtained through one or more software interactions.

$\mathtt{O}_1$ and $\mathtt{O}_2$ are summed to obtain a total number of missing specific features in the architecture.

The architectures described in Figure 2 are evaluated along these metrics in the remainder of this Section.

3.2. Tight integration

Integration complexity. This architecture does not require any component in addition to the science gateway and infrastructure ($\texttt{I}_1$=2) and it involves 5 interactions: $\texttt{a}_1$, $\texttt{a}_2$, $\texttt{b}$, $\texttt{c}_1$ and $\texttt{c}_2$ ($\texttt{I}_2$=5).

Robustness. No component is specific to workflow execution ($\texttt{R}_1$=0), but interactions $\texttt{b}$ and $\texttt{c}_2$ are ($\texttt{R}_2$=2).

Extensibility. Integrating a new type of workflow engine requires modifying the science gateway as well as interactions $\texttt{b}$ and $\texttt{c}_2$ ($\texttt{E}_1$=3). Updating a workflow engine version requires modifications in the science gateway ($\texttt{E}_2$=1). Inserting a new workflow is done through interactions $\texttt{a}_1$ and $\texttt{a}_2$ ($\texttt{E}_3$=2). Adding a new infrastructure generates updates in interactions $\texttt{a}_1$, $\texttt{b}$, $\texttt{c}_1$ and $\texttt{c}_2$ ($\texttt{E}_4$=4).

Scalability. Adding a new engine instance requires a new instance of the complete science gateway ($\texttt{S}_1$=2). Specific IT setups such as load-balancing between web server instances might be used to create new gateway instances, but they would not allow a single gateway instance to use multiple engine instances, which is the point here. Distributed engines are not available by default ($\texttt{S}_2$=1). The architecture does not add any particular complexity to the task scheduling problem since the workflow engine may implement any kind of scheduling policy ($\texttt{S}_3$=0).

Specific features. Supporting meta-workflows requires specific developments in the workflow engine or science gateway to allow workflows to execute other workflows ($\texttt{O}_1$=1). For instance, specific Java objects may be implemented in the science gateway framework to execute different types of workflows and link them together. The science gateway can retrieve fine-grained debugging information from the workflow engine directly ($\texttt{O}_2$=0).

3.3. Service invocation

Integration complexity. Service invocation requires a workflow service in addition to the science gateway and infrastructure ($\texttt{I}_1$=3). The architecture involves 6 interactions: $\texttt{a}_1$, $\texttt{a}_2$, $\texttt{b}$, $\texttt{c}_1$, $\texttt{c}_2$ and $\texttt{d}$ ($\texttt{I}_2$=6).

Robustness. The workflow service is a component specific to workflow execution ($\texttt{R}_1$=1). Workflow execution also involves 3 specific interactions: $\texttt{b}$, $\texttt{c}_2$ and $\texttt{d}$ ($\texttt{R}_2$=3).

Extensibility. Adding a new type of workflow engine requires implementing the corresponding workflow service, modifying interaction $\texttt{d}$, and implementing interactions $\texttt{b}$ and c_2 ($\texttt{E}_1$=4). New engine versions can be added by updating the workflow service without modifying any interaction or component ($\texttt{E}_2$=0). Updating the engine version in a workflow service does not count in $\texttt{E}_2$ since the only goal of this component is to wrap the engine. New workflows are added in the science gateway or in the workflow engine through interactions $\texttt{a}_1$ and $\texttt{a}_2$ ($\texttt{E}_3$=2). Adding a new type of infrastructure requires updates in interactions $\texttt{a}_1$, $\texttt{b}$, c_1 and c_2 ($\texttt{E}_4$=4).

Scalability. The service architecture in principle supports multiple engine instances through multiple workflow services. Adding a new engine instance, however, requires specific developments in the science gateway ($\texttt{S}_1$=1). Distributing the execution of a single workflow in multiple engines is usually not possible unless the workflow engine has specific abilities ($\texttt{S}_2$=1).

Specific features. Meta-workflows require specific developments in the workflow engine or science gateway to allow workflows to execute other workflows. ($\texttt{O}_1$=1). The science gateway needs to invoke interaction $\texttt{d}$ to retrieve dine-grained debugging information from the workflow service ($\texttt{O}_2$=1).

3.4. Task encapsulation

Integration complexity. Task encapsulation requires only 2 components ($\texttt{I}_1$=2). It involves 5 interactions: $\texttt{a}_1$, $\texttt{a}_2$, $\texttt{b}$, $\texttt{c}$ and $\texttt{e}$ ($\texttt{I}_2$=5).

Robustness. No component is specific to workflow execution ($\texttt{R}_1$=0), and only interaction $\texttt{e}$ is necessary ($\texttt{R}_2$=1).

Extensibility. Integrating a new type of workflow engine requires developing interaction $\texttt{e}$ and installing the engine on the infrastructure ($\texttt{E}_1$=2). Updating an engine version in the architecture shares the same mechanism as version updates of other tasks, which requires an update on the infrastructure ($\texttt{E}_2$=1). New workflows are integrated by creating a new task in the science gateway through interactions $\texttt{a}_1$ and $\texttt{a}_2$ ($\texttt{E}_3$=2). Adding a new infrastructure requires updating interactions $\texttt{a}_1$, $\texttt{b}$ and $\texttt{c}$ in the science gateway ($\texttt{E}_4$=3).

Scalability. New engine instances are spawned and executed on the infrastructure as any other task upon user submission ($\mathtt{S}_1$=0). This is a major interest of task encapsulation. Distributed engines are not supported by default ($\mathtt{S}_2$=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 ($\mathtt{S}_3$=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 the sub-tasks because workflows are by definition more complex than their sub-tasks: errors on sub-task parameter estimations accumulate in the workflow, and additional control constructs such as tests and loops may further increase the uncertainty. Such parameter estimation errors may generate issues such as selection of wrong batch queues on clusters or task termination due to exceeded quotas. Finally, the interdependencies between the workflow task and its sub-tasks may create deadlocks when there is contention. For instance, if only 1 computing resource is available for the science gateway and if 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.

Specific features. Task encapsulation explicitly enables meta-workflows ($\mathtt{O}_1$=0). Indeed, workflow engines may invoke other engines using the exact same interaction used to submit regular sub-tasks (interaction `e`), without any additional development. Fine-grained debugging information is obtained through interaction `b` ($\mathtt{O}_2$=1).

3.5. Pool model

Integration complexity. The pool model requires a workflow pool and an agent in addition to the science gateway and infrastructure ($\mathtt{I}_1$=4). It involves 7 interactions: $\mathtt{a}_1$, $\mathtt{a}_2$, `b`, $\mathtt{c}_1$, $\mathtt{c}_2$, `d` and `f` ($\mathtt{I}_2$=7).

Robustness. The workflow pool and agent are specific to workflow execution ($\mathtt{R}_1$=2). Interactions `b`, $\mathtt{c}_2$, `d` and `f` also are ($\mathtt{R}_2$=4).

Extensibility. Adding a new engine type requires wrapping the engine into the agent and updating interactions `b` and $\mathtt{c}_2$ ($\mathtt{E}_1$=3). Updating the version of an engine is transparent ($\mathtt{E}_2$=0) since it only requires updating the agent, which is a component dedicated to the engine. Integrating a new workflow is done through interactions $\mathtt{a}_1$ and $\mathtt{a}_2$ ($\mathtt{E}_3$=2). Integrating a new infrastructure requires updates in interactions $\mathtt{a}_1$, `b`, $\mathtt{c}_1$ and $\mathtt{c}_2$ ($\mathtt{E}_4$=4).

Scalability. By design, new engine instances only require new agents, which can be easily automated ($\mathtt{S}_1$=0). For instance, auto-scaling rules can be implemented to start new agents when the workload in the science gateway exceeds a certain threshold [27]. Distributed engines are not available by default ($\mathtt{S}_2$=1) and the architecture does not add any complexity to the task scheduling problem ($\mathtt{S}_3$=0).

Specific features. Meta-workflows require specific developments in the workflow engine to enable workflow submission ($\mathtt{O}_1$=1). Debugging information is accessed through interactions `d` and `f` ($\mathtt{O}_2$=1).

3.6. Nested workflows with service invocation

Integration complexity. Setting up a nested workflow architecture with service invocation requires a science gateway, 2 workflow services and 2 infrastructures ($\mathtt{I}_1$=5). The architecture involves 11 interactions ($\mathtt{I}_2$=11): $\mathtt{a}_1$ (twice), $\mathtt{a}_2$, `b` (twice), $\mathtt{c}_1$ (twice), $\mathtt{c}_2$ (twice) and `d` (twice).

Robustness. The two workflow services are specific to workflow execution ($\mathtt{R}_1$=2). Interactions `b` (twice), $\mathtt{c}_2$ (twice) and `d` (twice) also are necessary ($\mathtt{R}_2$=6).

Extensibility. Adding a new type of *parent* engine requires implementing the corresponding service, implementing interactions `b` and $\mathtt{c}_2$ in the parent engine, and implementing interaction `d` in the science gateway and in the parent service ($\mathtt{E}_1$=5). Adding a new type of *child* engine only requires implementing the corresponding service, developing interactions `b` and $\mathtt{c}_2$ in the child engine, and implementing interaction `d` in the parent service ($\mathtt{E}_1$=4).

We use $\texttt{E}_1\texttt{=4.5}$ in Table 2 to reflect both conditions. Adding a new version in the parent or child engine only requires modifying this engine ($\texttt{E}_2\texttt{=0}$). Adding a new workflow is done through interaction $\texttt{a}_1$ and $\texttt{a}_2$ ($\texttt{E}_3\texttt{=2}$). Adding a new infrastructure requires re-implementing interactions $\texttt{a}_1$, $\texttt{b}$ and $\texttt{c}_2$ twice, and interaction $\texttt{c}_1$ once so that it can be supported by both workflow engines ($\texttt{E}_4\texttt{=6}$).

Scalability. As in the service architecture, adding a new engine instance requires specific developments in the science gateway (instance of a parent engine), or in the parent engine (instance of a child engine) ($\texttt{S}_1\texttt{=1}$). Similarly, elastic engines are difficult to achieve. Distributed engines can be implemented through meta-workflows ($\texttt{S}_2\texttt{=0}$). Task scheduling is more complex than in other architectures though, due to the fact that workflow execution is split in different engines ($\texttt{S}_3\texttt{=1}$).

Specific features. Meta-workflows can be implemented, which is one of the main interest of this architecture ($\texttt{O}_1\texttt{=0}$). Note, however, that the complexity of the architecture increases with the number of engine types involved in meta-workflows: for instance, a meta-workflow with child workflows executed by 2 different engine types would require an additional workflow service and the corresponding interactions. Debugging information is accessed through interaction $\texttt{d}$ (invoked twice) ($\texttt{O}_2\texttt{=1}$).

3.7. Workflow conversion with service invocation

Integration complexity. Workflow conversion with service invocation requires the same components as for service invocation ($\texttt{I}_1=3$), and an additional $\texttt{g}$ interaction ($\texttt{I}_1=7$).

Robustness. Since workflow conversion is not involved in the execution (it is an offline process), the scores are the same as for the service architecture ($\texttt{R}_1\texttt{=1}$, $\texttt{R}_2\texttt{=3}$).

Extensibility. Since adding a new type of workflow engine aims at supporting more workflows, we consider that it only requires re-implementing interaction $\texttt{g}$ in this architecture ($\texttt{E}_1\texttt{=1}$). Note, however, that implementing interaction $\texttt{g}$ can require substantial work depending on the complexity of the workflow language used by the new engine. Similarly, adding a new engine version only requires modifying interaction $\texttt{g}$ ($\texttt{E}_2\texttt{=1}$). Adding a new workflow is done through interactions $\texttt{a}_1$, $\texttt{a}_2$ and $\texttt{g}$ ($\texttt{E}_3\texttt{=3}$). As in the service architecture, interfacing with a new infrastructure requires modifications in interactions $\texttt{a}_1$, $\texttt{b}$, $\texttt{c}_1$ and $\texttt{c}_2$ ($\texttt{E}_4\texttt{=4}$).

Scalability. Since workflow conversion is not involved in the execution (it is an offline process), the score is the same as for the service architecture ($\texttt{S}_1\texttt{=1}$, $\texttt{S}_2\texttt{=1}$, $\texttt{S}_3\texttt{=0}$).

Specific features. Meta-workflows are available after conversion, by connecting workflows in the language used in the science gateway ($\texttt{O}_1\texttt{=0}$). Debugging information is accessible as in the service invocation architecture ($\texttt{O}_2\texttt{=1}$).

4. Discussion

4.1. Comparison between architectures

Tight integration and task encapsulation are the simplest architectures to integrate, followed by service integration, workflow conversion (with service invocation) and pool. Nested workflows (with service invocation) require more integration than the other architectures. Robustness roughly leads to the same ordering of architectures, with tight integration and task encapsulation in the top group, service integration and workflow conversion (with service invocation) close behind, pool in a third group, and nested workflows (with service invocation) at the end. This ordering is consistent across metrics; it reflects the global complexity of the architectures.

Regarding extensibility, most architectures are overall comparable, except nested workflows (with service invocation) which are significantly behind. This is explained by the complexity of the nested workflow architecture, with 2 infrastructures and 2 workflow services. Task encapsulation and workflow conversion (with service invocation) perform slightly better when integrating new engines ($\texttt{E}_1$), which makes them useful architectures for science gateways that do not focus on a particular engine. However, task encapsulation, workflow conversion (with service invocation) and tight integration perform worse than the others when adding engine versions ($\texttt{E}_2$), due to the need to update infrastructure (task encapsulation), science gateway (tight integration), or workflow converter (workflow conversion). Workflow conversion performs worse than the others when integrating new workflows ($\texttt{E}_3$) because of the language conversion step. All architectures except nested workflows are equivalent when integrating new infrastructures ($\texttt{E}_4$).

The pool architecture is overall the most scalable, which is not surprising since it is designed precisely for scalability. Tight integration is the least scalable and all the other architectures perform the same overall. The global scalability score, however, should not conceal the unique characteristics of architectures regarding this criterion. Nested workflows are the only architecture that can easily accommodate distributed workflow execution, which can be critical in some cases. At the same time, the scheduling constraints created by task encapsulation and nested workflows may become problematic depending on the type of targeted infrastructure. Non-reliable infrastructures, for example, could hardly cope with workflow engines being wrapped in computing tasks as done in task encapsulation. The availability of multiple engine instances could also become a critical feature for science gateways with important workloads, which would favor task encapsulation and pool.

Differences in other specific features should not be neglected. Task encapsulation, nested workflows (with service invocation) and workflow conversion (with service invocation) are the only architectures that intrinsically support meta-workflows. Besides, tight integration is the only architecture that allows fine-grained debugging. The availability of fine-grained debugging information may be critical to efficient user support.

Table 3 provides an overall comparison between architectures, based on the metrics in Table 2. Note that this analysis is only meant for illustration purposes, since it assumes that all criteria have equal weights while real systems would favor some of them. Overall, task encapsulation performs a bit better than the other architectures and nested workflows stand a bit behind for the reasons mentioned previously.

4.2. Limitations

A few limitations should be considered when using the results of our evaluation. First, our evaluation methods aimed to provide comparative metrics, however the high-level of abstraction used to derive these metrics hinders the complexity of real systems to some extent. In particular, the presented architectures are abstract patterns that may be mixed together in actual systems. The distinction between tight integration and service invocation, for instance, may not always be that clear in practice. Service invocation may also be combined with task encapsulation in some cases. However, the criteria discussed in this work would still apply to broadly compare and categorize such cases of hybrid architectures.

Moreover, all the interactions involved in the architectures were treated equally, whereas some of them are obviously more complex than others. For example, interaction `g` (workflow language conversion) is clearly more complex than interaction `d` (service invocation), and may challenge reliability more strongly. However, without entering into the details of a particular system, quantification of the robustness or the amount of development associated with each interaction and software component will hardly be precise.

The particular case of interaction `g` (workflow language conversion) requires particular attention when implementing a real system since this interaction may not be easily generalizable to any workflow language. For instance, converting FSL, PSOM or Nipype pipelines to any other workflow language is problematic because these engines rely on general-purpose programming languages such as Bash, Octave and Python, which are much richer than scientific workflow languages.

The abstract nested workflows and workflow conversion patterns were instantiated with service invocation so that they can be analyzed in the same framework as the other architectures. Other types of instantiation, for instance nested workflows with task encapsulation, could also be envisaged. We chose to limit ourselves to instantiations with service invocation because the resulting architectures are implemented in real systems, and because service invocation is largely used. Nevertheless, it could be interesting to explore other types of instantiations.

The particular technical or historical context of a science gateway project may obviously influence the choice of an architecture to integrate workflow engines. For instance, many workflow engines are already available as web services, which tends to favor service invocation, and other science gateways may have strongly tested task and data control features (interactions `b` and `c`), which would favor task encapsulation. Similarly, adding a new type of engine may facilitate integration of a new infrastructure when interactions `b` and $\mathtt{c}_2$ are already available. The migration cost between architectures has been ignored as well.

Finally, workflow engines are only one of the many aspects involved in the design of a science gateway. Other properties definitely influence the

	Tight	Service	Task	Pool	Nested	Conversion
Integration (global score)	0.00	0.22	0.00	0.44	1.00	0.33
Robustness (global score)	0.14	0.43	0.00	0.71	1.00	0.43
Extensibility (global score)	0.44	0.44	0.00	0.22	1.00	0.22
Scalability (global score)	1.00	0.50	0.50	0.00	0.50	0.50
Specific features (global score)	0.00	1.00	0.00	1.00	0.00	0.00
	1.6	**2.6**	**0.5**	**2.4**	**3.5**	**1.5**

Table 3: Overall evaluation. Brighter colors and lower scores indicate better performance. Scores are obtained by summing the normalized global scores (m' values) of each criterion in Table 2 and the colors are obtained as in Table 2.

design of a software architecture, for instance collaborative features, data visualization, search, authentication, accounting and so on.

5. Related work

There is abundant literature describing specific workflow engines and systems; some of the main references are cited in Section 1.1. A few works described the architecture and features of workflow system at an abstract level. For instance, the early work [48] proposes a generic architecture for grid workflow systems that is based on the workflow reference model defined by the Workflow Management Coalition[5]. Deelman et al. [10] have further characterized the features of workflow systems. These two works, however, do not mention science gateways.

Numerous science gateways have been described, as mentioned in Section 1.2. However, only a few works focused on science gateway architectures. Shahand et al. [39] presents the Science Gateway Canvas, which is a business reference model where the most relevant functional components are organized into functional groups. Although workflow management is mentioned as a possible component to coordinate distributed computations in science gateways, the paper does not comment on architecture to achieve this.

Olabarriaga et al. [33] presents a user-centered view of the ecosystem of science gateways with focus on workflow management. Their proposed layered system architecture includes gateway, workflow engine and distributed infrastructure components, similarly to the "service invocation" pattern. However other patterns are not identified.

6. Conclusion

We have systematically reviewed architectures used to integrate workflow engines in science gateways. These architectures are described in a system-independent framework suitable for comparison, illustrated on real systems, and evaluated using novel quantitative metrics that allow simple comparison across architectures. We have discussed the pros and cons of all the presented architectures based on these metrics.

To the best of our knowledge, our work is the first to systematically review and compare software architectures used to integrate workflow engines into science gateways. So far, the literature on science gateways and workflow engines has focused on the description of particular systems, or on the presentation of a particular architecture. Instead, our analysis abstracts and evaluates architectural patterns independently from any particular system. It provides general insight to integrate science gateways and workflow engines, which we hope will be useful for software architects.

7. Acknowledgments

This work has been made possible with the support of the Irving Ludmer Family Foundation and the Ludmer Centre for Neuroinformatics and Mental Health. The integration between PSOM and CBRAIN was supported by a Brain Canada Platform Support Grant, as well as the Canadian Consortium on Neurodegeneration in Aging (CCNA), through a grant from the Canadian Institute of Health Research and funding from several partners. This work is in the scope of the LABEX PRIMES (ANR-11- LABX-0063) of Université de Lyon, within the program "Investissements dAvenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR). This work also falls into the scope of the scientific topics of the French National Grid Institute (IdG). The VIP team thanks the site administrators of the European Grid Initiative and the GGUS support for their help related to the VIP platform. The CBRAIN team is grateful for the computing cycles, storage, and support obtained from Compute

[5] `http://www.wfmc.org`

Canada (https://computecanada.ca) and platform development program from CANARIE (http://www.canarie.ca/). We also acknowledge the Dutch national e-Infrastructure with the support of SURF Cooperative, the Dutch national program COMMIT/ and the High Performance Computing and Networking (HPCN) Fund of the University of Amsterdam for their support to the science gateway actitivies at the AMC. We are also grateful to the financial support provided by FP7 E-INFRASTRUCTURE program for financial support to SCI-BUS, SHIWA and ER-flow projects.

References

[1] Mohamed Abouelhoda, Shadi Alaa Issa, and Moustafa Ghanem. Tavaxy: Integrating Taverna and Galaxy workflows with cloud computing support. *BMC Bioinformatics*, 13(1):1–19, 2012.

[2] Y. Ad-Dabbagh, D. Einarson, O. Lyttelton, J.-S. Muehlboeck, K. Mok, O. Ivanov, R.D. Vincent, C. Lepage, J. Lerch, E. Fombonne, and A.C Evans. The CIVET Image-Processing Environment: A Fully Automated Comprehensive Pipeline for Anatomical Neuroimaging Research. In *Proceedings of the 12th Annual Meeting of the Organization for Human Brain Mapping*, 2006.

[3] Valeria Ardizzone, Roberto Barbera, Antonio Calanducci, Marco Fargetta, E Ingrà, Ivan Porro, Giuseppe La Rocca, Salvatore Monforte, R Ricceri, Riccardo Rotondo, et al. The DECIDE science gateway. *Journal of Grid Computing*, 10(4):689–707, 2012.

[4] John Ashburner. SPM: A history. *NeuroImage*, 62(2):791 – 800, 2012.

[5] Bartosz Balis. Hyperflow: A model of computation, programming approach and enactment engine for complex distributed workflows. *Future Generation Computer Systems*, 55:147–162, 2016.

[6] Pierre Bellec, Sébastien Lavoie-Courchesne, Phil Dickinson, Jason Lerch, Alex Zijdenbos, and Alan C Evans. The Pipeline System for Octave and Matlab (PSOM): a lightweight scripting framework and execution engine for scientific workflows. *Frontiers in neuroinformatics*, 6(7), 2012.

[7] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. *Data Analysis, Machine Learning and Applications: Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation e.V., Albert-Ludwigs-Universität Freiburg, March 7–9, 2007*, chapter KNIME: The Konstanz Information Miner, pages 319–326. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[8] S. P. Callahan, J. Freire, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and Huy T. Vo. Managing the Evolution of Dataflows with VisTrails. In *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, pages 71–71, 2006.

[9] Luis de la Garza, Johannes Veit, Andras Szolek, Marc Röttig, Stephan Aiche, Sandra Gesing, Knut Reinert, and Oliver Kohlbacher. From the desktop to the grid: scalable bioinformatics via workflow conversion. *BMC Bioinformatics*, 17(1):1–12, 2016.

[10] Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

[11] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G Bruce Berriman, John Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[12] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17 – 35, 2015.

[13] Ivo D Dinov, John D Van Horn, Kamen M Lozev, Rico Magsipoc, Petros Petrosyan, Zhizhong Liu, Allan MacKenzie-Graham, Paul Eggert, Douglas S Parker, and Arthur W Toga. Efficient, distributed and interactive neuroimaging data analysis using the LONI pipeline. *Frontiers in Neuroinformatics*, 3(22), 2009.

[14] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. Toward an open cloud standard. *Internet Computing, IEEE*, 16(4):15–25, 2012.

[15] Thomas Fahringer, Radu Prodan, Rubing Duan, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong-Linh Truong, Alex Villazon, and Marek Wieczorek. Askalon: A grid application development and computing environment. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 122–131. IEEE Computer Society, 2005.

[16] 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, and D. Friboulet. A virtual imaging platform for multi-modality medical image simulation. *IEEE Transactions on Medical Imaging*, 32(1):110–118, 2013.

[17] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployment of data-intensive applications on grids with MOTEUR. *Journal of High Performance Computing and Applications*, 22(3):347–360, 2008.

[18] T. Glatard, P.O. Quirion, R. Adalat, N. Beck, R. Bernard, B.L. Caron, Q. Nguyen, P. Rioux, M-E. Rousseau, A.C. Evans, and P. Bellec. Integration between PSOM and CBRAIN for distributed execution of neuroimaging pipelines. In *Meeting of the Organization for Human Brain Mapping, Geneva, Switzerland*, 2016.

[19] Jeremy Goecks, Anton Nekrutenko, James Taylor, et al. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.

[20] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Gregor Von Laszewski, Craig Lee, Andre Merzky, Hrabri Rajic, and John Shalf. SAGA: A Simple API for Grid Applications. High-level application programming on the Grid. *Computational Methods in Science and Technology*, 12(1):7–20, 2006.

[21] Krzysztof Gorgolewski, Christopher D Burns, Cindee

Madison, Dav Clark, Yaroslav O Halchenko, Michael L Waskom, and Satrajit S Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in Python. *Frontiers in Neuroinformatics*, 5(3), 2011.

[22] Ákos Hajnal, Zoltán Farkas, and Péter Kacsuk. Data avenue: remote storage resource management in WS-PGRADE/gUSE. In *6th International Workshop on Science Gateways (IWSG)*, pages 1–5. IEEE, 2014.

[23] Mark Jenkinson, Christian F. Beckmann, Timothy E.J. Behrens, Mark W. Woolrich, and Stephen M. Smith. FSL. *NeuroImage*, 62(2):782 – 790, 2012.

[24] Peter Kacsuk. *Science Gateways for Distributed Computing Infrastructures.* Springer, 2014.

[25] Peter Kacsuk, Zoltan Farkas, Miklos Kozlovszky, Gabor Hermann, Akos Balasko, Krisztian Karoczkai, and Istvan Marton. WS-PGRADE/gUSE Generic DCI Gateway Framework for a Large Variety of User Communities. *Journal of Grid Computing*, 10(4):601–630, 2012.

[26] Jens Kruger, Richard Grunzke, Sandra Gesing, Sebastian Breuers, André Brinkmann, Luis de la Garza, Oliver Kohlbacher, Martin Kruse, Wolfgang E Nagel, Lars Packschies, et al. The MoSGrid science gateway–a complete solution for molecular simulations. *Journal of Chemical Theory and Computation*, 10(6):2232–2245, 2014.

[27] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12, 2012.

[28] Sharath Maddineni, Joohyun Kim, Yaakoub El-Khamra, and Shantenu Jha. Distributed application runtime environment (DARE): a standards-based middleware framework for science-gateways. *Journal of Grid Computing*, 10(4):647–664, 2012.

[29] Suresh Marru, Lahiru Gunathilake, Chathura Herath, Patanachai Tangchaisin, Marlon Pierce, Chris Mattmann, Raminder Singh, Thilina Gunarathne, Eran Chinthaka, Ross Gardler, et al. Apache Airavata: a framework for distributed applications and computational workflows. In *Proceedings of the 2011 ACM workshop on Gateway computing environments*, pages 21–28. ACM, 2011.

[30] Michael McLennan, Steven Clark, Ewa Deelman, Mats Rynge, Karan Vahi, Frank McKenna, Derrick Kearney, and Carol Song. HUBzero and Pegasus: integrating scientific workflows into science gateways. *Concurrency and Computation: Practice and Experience*, 27(2):328–343, 2015.

[31] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[32] S. Olabarriaga, T. Glatard, A. Hoheisel, A. Nederveen, and D. Krefting. Crossing HealthGrid Borders: Early Results in Medical Imaging. In *HealthGrid'09*, pages 62–71, Berlin, jun 2009.

[33] Silvia Olabarriaga, Gabrielle Pierantoni, Giuliano Taffoni, Eva Sciacca, Mahdi Jaghoori, Vladimir Korkhov, Giuliano Castelli, Claudio Vuerli, Ugo Becciani, Eoin Carley, et al. Scientific workflow management–for whom? In *e-Science (e-Science), 2014 IEEE 10th International Conference on*, pages 298–305. IEEE, 2014.

[34] Kassian Plankensteiner, Radu Prodan, Matthias Janetschek, Thomas Fahringer, Johan Montagnat, David Rogers, Ian Harvey, Ian Taylor, Ákos Balaskó, and Péter Kacsuk. Fine-Grain Interoperability of Scientific Workflows in Distributed Computing Infrastructures. *Journal of Grid Computing*, 11(3):429–456, September 2013.

[35] Sylvain Reynaud. Uniform access to heterogeneous grid infrastructures with JSAGA. In *Production grids in Asia*, pages 185–196. Springer, 2010.

[36] D. Rogers, I. Harvey, T.T. Huu, K. Evans, T. Glatard, I. Kallel, I. Taylor, J. Montagnat, A. Jones, and A. Harrison. Bundle and pool architecture for multi-language, robust, scalable workflow executions. *Journal of Grid Computing*, 11(3):457–480, September 2013 2013.

[37] Shayan Shahand, Ammar Benabdelkader, Mohammad Mahdi Jaghoori, Mostapha al Mourabit, Jordi Huguet, Matthan WA Caan, Antoine HC Kampen, and Sílvia D Olabarriaga. A data-centric neuroscience gateway: design, implementation, and experiences. *Concurrency and Computation: Practice and Experience*, 27(2):489–506, 2015.

[38] Shayan Shahand, Mark Santcroos, Antoine H.C. vanKampen, and Sílvia Olabarriaga. A grid-enabled gateway for biomedical data analysis. *Journal of Grid Computing*, 10(4):725–742, 2012.

[39] Shayan Shahand, Antoine H. C. van Kampen, and Sílvia D. Olabarriaga. Science Gateway Canvas: A business reference model for Science Gateways. In *Proceedings of the Science of Cyberinfrastructure: Research, Experience, Applications and Models*, SCREAM'15, Portland, OR, USA, 2015.

[40] Tarek Sherif, Pierre Rioux, Marc-Etienne Rousseau, Nicolas Kassis, Natacha Beck, Reza Adalat, Samir Das, Tristan Glatard, and Alan C Evans. CBRAIN: a web-based, distributed computing platform for collaborative neuroimaging research. *Frontiers in Neuroinformatics*, 8(54), 2014.

[41] Dawid Szejnfeld, Piotr Dziubecki, Piotr Kopta, Michal Krysinski, Tomasz Kuczynski, Krzysztof Kurowski, Bogdan Ludwiczak, Tomasz Piontek, Dominik Tarnawczyk, Malgorzata Wolniewicz, Piotr Domagalski, Jaroslaw Nabrzyski, and Krzysztof 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), 2010.

[42] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The Triana workflow environment: Architecture and applications. In *Workflows for e-Science*, pages 320–339. Springer, 2007.

[43] Gabor Terstyanszky, Tamas Kukla, Tamas Kiss, Peter Kacsuk, Ákos Balaskó, and Zoltan Farkas. Enabling scientific workflow sharing through coarse-grained interoperability. *Future Generation Computer Systems*, 37:46–59, 2014.

[44] Peter Tröger, Roger Brobst, Daniel Gruber, Mariusz Mamonski, and Daniel Templeton. Distributed resource management application API Version 2 (DRMAA). In *Technical report, Open Grid Forum, January 2012. Also available online: `http://www.ogf.org/documents/GFD.194.pdf`*, 2012.

[45] Matteo Turilli, Mark Santcroos, and Shantenu Jha. A

comprehensive perspective on the pilot-job abstraction. *arXiv preprint arXiv:1508.04180*, 2015.

[46] V. Korkhov, D. Vasyunin, A. Wibisono, V. Guevara-Masis, A.Belloum, C. de Laat, P. Adriaans, L.O. Hertzberger. WS-VLAM: Towards a Scalable Workflow System on the Grid. In *Proceedings of the 2nd workshop on Workflows in Support of Large-Scale Science (WORKS07), 16th IEEE International Symposium on High Performance Distributed Computing*, pages 63–68, 2007.

[47] Wenjun Wu, Thomas Uram, Michael Wilde, Mark Hereld, and Michael E Papka. Accelerating science gateway development with Web 2.0 and Swift. In *Proceedings of the 2010 TeraGrid Conference*, page 23. ACM, 2010.

[48] J. Yu and R. Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, 2005.

[49] Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor Von Laszewski, Veronika Nefedova, Ioan Raicu, Tiberiu Stef-Praun, and Michael Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE Congress on Services*, pages 199–206, 2007.

[50] A. P. Zijdenbos, R. Forghani, and A. C. Evans. Automatic "pipeline" analysis of 3-D MRI data for clinical trials: application to multiple sclerosis. *IEEE Transactions on Medical Imaging*, 21(10):1280–1291, Oct 2002.