A Framework for Domain-Specific Science Gateways

Joanna Kocot, Tomasz Szepieniec, Piotr Wójcik, Michał Trzeciak, Maciej Golik, Tomasz Grabarczyk, Hubert Siejkowski, and Mariusz Sterzel

AGH University of Science and Technology, ACC Cyfronet AGH,
ul. Nawojki 11, 30-950 Kraków, Poland
{j.kocot,t.szepieniec,p.wojcik,m.trzeciak,m.golik,t.grabarczyk,
h.siejkowski,m.sterzel}@cyfronet.pl

Abstract. While modern Federated Computing Infrastructures – Grids, Clouds and other technologies – continuously increase their computing power, their use for research still stays lower than desired. The authors' diagnosis of this problem is a technology barrier hard to overcome to people who want to focus only on science. The federated infrastructures are difficult to use not only due to the physical distribution of the resources and, thus, need for remote access, but, mainly, due to the fact that everyday patterns of interaction with a computer cannot be directly used for these resources. The way of performing computing operations on them is different than the usual way the scientists do their research using laptops or personal computers.

The authors claim that the key to increase the use of modern federated infrastructures for science is making the processing on these infrastructures resemble using a personal computer. The paper collects requirements from different scientific use cases and, from these requirements, derives a processing model that could satisfy all of them, thus, allowing to build a system, in which computations on large infrastructures can be similar to everyday work. This model is implemented by a framework for creating Science Gateways – InSilicoLab.

Keywords: federated infrastructures, processing model, science gateway, framework, *in silico*.

1 Introduction

Modern Federated Computing Infrastructures offer almost endless computing power. The capacity provided by Grids, Clouds and other technologies is huge, however, the level of their use for research remains lower than desired. The reason for this might be that, for a researcher, moving from an everyday computation done on a personal computer to a large parallel machine is a very time-consuming process. It involves transferring data to and from a remote server, launching a computation – usually without any user interface other than command-line, managing the computation status, etc. Most of this is only a burden for a regular researcher, who usually possesses some kind of programmatic skills, but

M. Bubak et al. (Eds.): PLGrid Plus, LNCS 8500, pp. 130–146, 2014.

they are not the essence of his work. The amount of required purely technical operations might not be an obstacle for large communities, like LHC, who are able to invest in dedicated teams of computer scientists and programmers, but it almost eliminates smaller communities. For the latter, the inability to use the federated infrastructures might cause not only making slower progress – while running computations on a PC or laptop takes longer than on a parallel machine – but also prevent some of the research at all – as larger (usually more important) datasets could take years to be calculated on a standard computer. Apart from the difficulties related to technical specifics of the computations management, there are some objective obstacles that make using distributed computing infrastructures difficult, e.g.:

- overhead of running a computation – acceptable for very large ones, but too big for smaller tasks,
- problematic data transfer – e.g. for Grid: inefficient and difficult, for Clouds: expensive,
- inability to interact with a running computation.

The stakes are high, as often the research conducted in smaller or more dispersed communities is of great importance to humanity – take bioengineering or seismology as an example.

Fortunately, with successive identification of the aforementioned barriers, new tools have started to be created, leading also to evolution of Science Gateways – portals that aim at facilitating the operation on large computer infrastructures by providing interfaces adjusted to a given community, and offering many other tools that might be of use to researchers coming from these communities.

This paper presents a framework for creating Science Gateways – InSilicoLab – that provides processing-related solutions tailored to specific domains of science. The gateways created with this framework, to facilitate the usage of the system, provide a user interface fully customized to a specific science domain, including the data types, possible operations and construction of the experimentation flow. On the other hand, they aim at reducing side effects of utilizing a computer infrastructure – not only the difficulties of performing operations on the infrastructure, but also indirect costs like time overheads and lack of interactivity (batch-style computations). The requirements for development of the gateway framework were based on several use cases concerning different domains of science or different areas of these domains.

The paper is constructed as follows: Section 2 gives an overview of computation use cases coming from different domains of science; summary of the requirements for a processing model identified on the base of these use cases is given in Section 3. Section 4 proposes an implementation of this model, while Section 5 specifies a framework for Science Gateways using it. The current status of work on the framework is summarized in Section 6. Section 7 gives an overview of the related work in the field of Science Gateways, and, in general, in computation models and tools for science. The article is summarized in Section 8.

2 Use Cases

This section presents several use cases from different domains of science or communities gathered around specific classes of problems within these domains.

2.1 Computational Chemistry

Computational chemistry is one of the domains that heavily utilize the resources of advanced computing infrastructures. Its computational methods are usually used to explain or predict empirical effects. Large computational power is necessary to improve accuracy of the simulation outcomes, extend the timescale of simulations or, sometimes, just to make modelling a large system possible. Computational cost of these methods can be the time needed to complete the calculations as well as large memory consumption and disk storage use.

Similar modelling is also used in other sciences, like nanotechnology or biological sciences. The significance of this work was recently confirmed by the Nobel Foundation who awarded the 2013 Nobel Prize in chemistry for "the development of multiscale models for complex chemical systems" [1,2].

A classic pattern for Quantum Chemistry computations is comparing similar systems (geometries), e.g. optimizing their energy or finding other properties. This requires preparing input data for different systems, launching computations for all of them and, then, repeatedly checking if the results are available. The results are text files containing the log of the whole computation process – sometimes including even the license and the 'message of the day'. Therefore, analyzing these files is difficult and tiresome, but often required, also while the computation is running – to control whether the processing is going to be convergent. If not, the process should be stopped and rerun with slightly different parameters. This is especially important in case of very time-consuming optimization procedures, that could otherwise run for days or weeks not giving eventually any correct results.

If the computations are launched on a distributed computing infrastructure, the user additionally has to transfer data to a suitable storage service, specify the computation description (usually, a file written in a description language specific to a given infrastructure), monitor the state of the jobs, and, afterwards, transfer the result data from a storage service to their computer. Even if it would further be used for next computations (which is a common scenario), the transfer to a local computer is required to analyze the contents of the file. What is more, in such case, watching the result (log) file during the computation is usually impossible or very difficult – and, in case of many parallel jobs – unmanageable [3].

Another scenario useful for the computational chemistry domain is an interactive "play" with a molecular system like Molecular Dynamics trajectory, where several parameters of the system reduction are chosen and tested on one of the trajectory frames. After the reduction is considered successful, the same operations can be applied to the whole trajectory or to a set of its chosen frames. In case of this scenario, implementing it using a traditional batch system would

bring more inconvenience than benefits, as the operations required to run the program and transfer files take longer than the actual computation.

2.2 CTA

The Cherenkov Telescope Array (CTA) [4] is a large project from astrophysics domain that aims at building the next generation ground-based telescopes to observe very high energy gamma-rays. As the project is still in its preparatory phase, large simulations are needed to model the telescopes response with respect to different parameters – ranging from the telescopes size, placement and geographical location to the electronic trigger configuration. At this stage, heavy Monte Carlo (MC) computations are run to find the optimum values of these parameters, considering as well individual telescopes as the whole telescope array. The simulations often have a character of a parameter sweep – to thoroughly test the possible values of a range of parameters, usually also for different input files – which produces a very large number of parallel simulations. The simulations are based on data containing simulated atmospheric events, which tend to be very large (usually several GBs). Such large data cannot be stored on personal computers, and even their transfer from a dedicated storage takes considerable time. The duration of the computations themselves also depends on the data size as well as on the parameter values.

Within the next few years, CTA will enter its operational phase, when it will serve as an open observatory to a wide astrophysics community and will provide a deep insight into the non-thermal high-energy universe. The MC simulations will be needed in a limited number then, and the software will have to be targeted at analysis of the data produced by the telescopes. This has to be prepared earlier, to calibrate both the telescopes and the software parameters to provide reliable information to the astrophysics community. Such analysis requires multiple transformations of the telescope data to extract relevant information and to reconstruct the direction and the energy of the original atmospheric event, which are then compared against the simulated input values. This is done by several tools organized in a pipeline, for which some elements might be exchanged (e.g. for verification, some of the analyses have to be run twice, using different methods – only if they give similar outcomes, the resulting information can be considered valid).

Another important aspect of the telescope array operation will be consuming the data that come from the instrument. This will be streams of large data coming continuously – not being divided into individual files. To process data of such character, neither batch systems nor interactive applications are enough. Such data require completely different approach that would consider data not as files but as streams.

2.3 MHD Simulations in Astrophysics

Magnetohydrodynamic (MHD) simulations are an important way of modelling processes and studying their evolution in many astrophysical objects and

environments. The codes used for this modelling have the highest requirements with respect to the computing infrastructure – e.g., one of the scientists performing these simulations every year reaches the first place in statistics of the resource consumption in the PL-Grid national grid infrastructure[1]. An important aspect of these simulations is monitoring the status of the physical system being modelled. Therefore, it is convenient to produce a significant amount of pictures delivered on the fly with their number and names not known prior to the calculations.

It is also useful to do a compilation test before the computation is submitted to the infrastructure. This allows the user to tune the compilation parameters and test whether they are valid and work with the application. This testing is of large significance, as the application could run for even as long as several months. A test compilation could be done several times until satisfactory configuration is reached, which requires that its results are returned almost instantly. Only the final, validated parameters are used for a real compilation in the infrastructure, which is done before the program starts.

2.4 Bioinformatics

The bioinformatics calculations are known for their complexity with respect to the number of different (often small) programs that have to be run in a specific order (see e.g. the Gromacs application [5]) – usually one taking output from the other as input. This order might be different for different classes of problems, therefore, different researchers have their own patterns of using these programs. A natural way of facilitating work with them would be connecting the individual programs in workflows or pipelines. However, automating that is very difficult due to the lack of formal semantic description. What is more, some of these programs cannot be run in batch mode, as they interactively ask the user for input. The interactivity is also often required on the level of constructing the workflow or pipeline – by dynamically choosing the subsequent processing steps. This is due to the fact that the decision on the next step of the experiment is often based on early obtained results of the previous step – e.g. the user observes a plot generated by one of the programs and decides whether to continue the experiment or recalculate the current step with different parameters.

The bioinformatics community has also created a well-developed set of protein databases (like Protein Data Bank [6]) or publication databases that publish a service API, which enables them to be used through service method calls (e.g. REST [7] operations).

3 Processing Model Based on Scientific Use Cases

The requirements gathered on the base of the presented use cases show that scientific computations not only need large computational resources, but also

[1] According to the statistics presented on the forum of the PL-Grid Consortium.

immediate response and interactivity. Therefore, a comprehensive computing model targeted at research communities has to, on one hand, enable or facilitate access to large computer infrastructures, and, on the other, consider interactivity and responsiveness as key functionality.

We can summarize the requirements for such computing model by introducing the following three types of processing that have to be addressed:

- *Batch operations* – which usually take a significant amount of time to complete, therefore, their immediate result is not important. These operations often involve managing large datasets. There are also two features of such computations that are usually neglected by traditional computational systems:
  - Interaction with the application – the computational task run on an infrastructure might execute an application that, at some point, requires user input or simple decision.
  - Continuous preview of the application results – partial results should be available for preview during the computation (if the application produces such partial results before it completes).
- *Immediate operations* – requiring immediate response from the system. They would be usually used jointly with other types of processing – e.g. to test parameters for larger batch computation or to deploy a service.
- *On-demand deployment of services* – operations that deploy services to be later used by other components. The service endpoints will be distributed across the user infrastructure. The use of such services could enable processing data streams – which could not be handled by any other mechanism.

All of the above-listed types of processing require interactive mechanisms, where not only outputs are available immediately, but also processing units of all kinds can prompt for a user decision on a specific matter. This should be presented to the user in their interface or sent as a notification in case they are off-line.

Implementing this concept of performing computations brings us closer to the main aim a modern processing model for computing infrastructures should follow – i.e., preserving the same functionalities and usage patterns that the users have on their personal computers. Such model is already partially available through contemporary grid and cloud technologies (IaaS, PaaS), however, it is not yet accessible in a unified way. The authors believe that combining all these in a single system, which is a heart of a domain-specific portal solution, might be a vast change in the way of performing computing-aided research. A portal empowered with a system based on the presented processing model, on one hand, gives access to powerful processing mechanisms, and on the other, can offer a personalized user interface containing all necessary tools and applications required by a specific domain of science. We call it a Science Gateway.

4 Implementation of the Processing Model

To implement the computation model represented by the three classes described in Section 3, the following system architecture was designed.

The central component of the system (see Fig. 1) is a message broker (implemented as a message queue) that coordinates requests of the other actors: a requestor and a worker. The requestor is an entity (usually a portal interfacing the user) that defines the computation to be performed, whereas the worker is an actual program that performs the computation. Both the requestor and the worker can be deployed in multiple instances, either providing different functionality or duplicating an existing component – the latter providing means for load balancing. The message broker ensures a reliable asynchronous communication between all the actors. Fig. 1 shows this communication scheme.

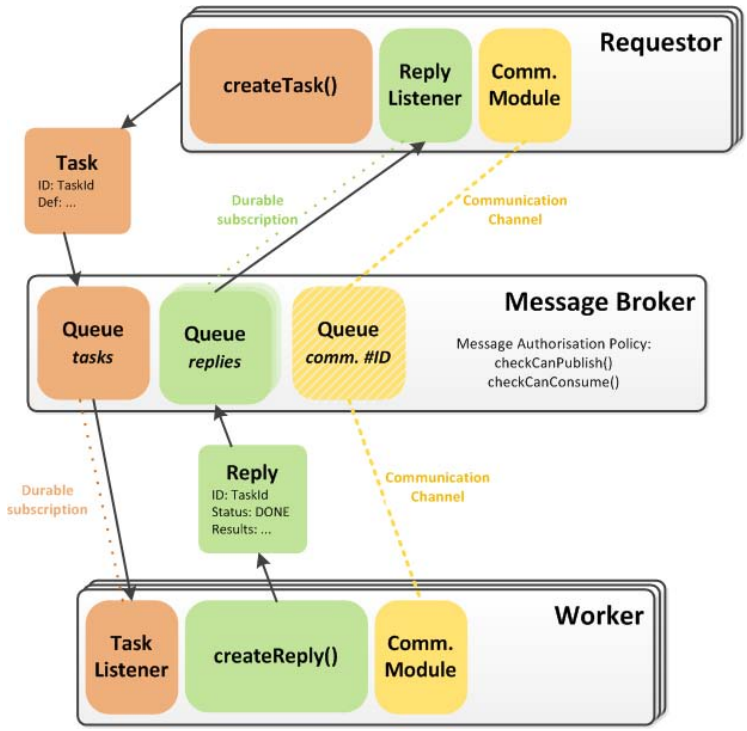

Fig. 1. Schema of the communication in the system: Requestor and Worker instances interacting through a Message Broker

Workers start their execution by subscribing to the queue and declaring what type of specific task they are able to process (e.g., a specific program, dynamically deployed visualization service, etc.). As the workers communicate with the rest of the system only via the message queue, they can be launched on any computer capable of running the specific task – regardless whether it is a node of an advanced distributed infrastructure or just a personal computer. Likewise, they can be implemented in any software technology or programming language. Workers of adequate type (able to process certain task type) can be spawned or terminated according to the system load.

The requestor is responsible for creating task definitions based on the user input, and submitting them to the message queue. The tasks are portions of work that have to be delegated from the requestor (e.g. portal) to workers. They are usually a specific set of parameters and/or input files for a concrete program run by the worker. When the requestor submits a task of a certain type to the queue, the message broker sends it to one of the workers, that are capable of processing it. The workers are usually built for a concrete application, and, therefore, need only a limited configuration and input files passed with the task description to start the computation. Such approach limits their use to only computations of a specified kind, however, the behavior of such workers is much more predictable and can be more easily handled and communicated to the user. Furthermore, implementing a general-purpose worker that would be able to execute an arbitrary script is also possible, to complement the system functionality.

At any time, the worker may send messages back to the requestor – also through the message queue. Such messages can include e.g. task results (also partial ones), execution status or error information. What is more, when receiving a task from the queue, the worker subscribes to a new, temporary queue, which serves as a channel for additional message exchange with the requestor (concerning only the given task). In this way, a two-way communication is possible, in which both sides may initiate the interaction and both sides await messages – e.g., the requestor can send additional data required by the task during its execution, or the worker may notify the requestor that it needs an input from the user. If a worker is deploying a service with an external interface, this communication channel will still be active, therefore, allowing the worker to communicate with the requestor in the usual manner (e.g. enabling the requestor to send a shutdown message), while external entities will be accessing it according to the service protocol and interface.

Relying on asynchronous message passing as the communication layer, adds an important quality to the system, namely, the persistence. Appropriate message broker policies ensure that task descriptions are not lost in case of worker failure or connection problems. Similarly, should the requestor side fail, all messages from the workers are persisted by the message broker and delivered as soon as the requestor restarts/reconnects. The message broker policies also ensure the integrity of the system – e.g., that one task request will not be consumed by many workers at a time.

The latency added by the communication through a queue is minimal, therefore, instant response of the system is assured assuming there is a free worker to perform the requested task, and that, in turn, may be managed by appropriate worker deployment and spawning policy.

The workers are owned by users who run them either manually – by executing them directly or starting through a dedicated interface – or automatically, by triggering a computation of a certain type. The owner can also set the worker policy to allow other users to utilize it. In this way, light, permanent workers can be deployed on dedicated resources to realize small computations and be

available to all the users. Such workers can perform immediate operations, as they are constantly ready to receive and perform their tasks. Once they complete, the result is instantly put to the queue and dispatched to the requestor, which is waiting for it.

All workers are managed and monitored by a separate component – Worker Manager, which is aware how many workers, of what kind, and belonging to what users are alive, and what is their state (e.g. busy, waiting for reply, unoccupied). The manager would automatically adjust the pool of permanent workers to ensure the immediate operations are performed fast, and trigger notifications to the user whenever the user-managed pool of workers is overloaded.

5 Science Gateway Framework

To create a unique solution tailored to the needs of a specific research community, the generic computation model should be built into a larger framework that would specify how to create, on one side, the user interfaces, and, on the other, the connection to the underlying infrastructures.

5.1 Framework Architecture

The schema presented in Section 4 specifies the model of communication between the system components. This section specifies the architecture of a framework – InSilicoLab – that uses this communication model to build advanced Science Gateway solutions. It is composed of three layers:

- **Domain Layer** is the layer of contact with the user, providing specified tools for use in a concrete domain of science. These are usually exposed through a web portal to facilitate access from remote destinations. As responsible for the contact with the users, the Domain Layer has to assure maximum usefulness to them. Therefore, it operates only on actions and objects from the researchers domain or common to the research community – like Experiment, Analysis, Results, etc.
- **Mediation Layer** is responsible for performing the actions triggered in the user interfaces of the Domain Layer. The requests are handled with use of the InSilicoLab Core Services, that are able of interacting with the resources of the Resource Layer.
- **Resource Layer** is the layer of the computational and storage resources. The system interacts with these resources by calling resource-specific commands through the resource interfaces – e.g., submitting a computational job or saving entities to a database. The Workers (labelled with 'W' in the picture Fig. 2) reside on the members of this layer, serving as an additional interface for contact from the Core Services.

The user interaction with the system is realised through components of a Science Gateway (in the Domain Layer). It is responsible for taking the user input and sending it to the components of the Mediation Layer. They, in turn, translate

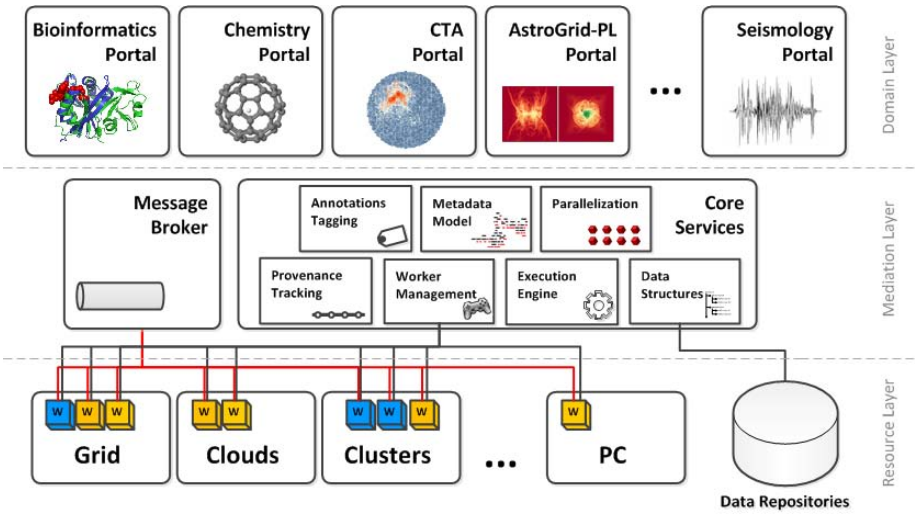

Fig. 2. Architecture of the InSilicoLab framework: Domain Layer, Mediation Layer with its Core Services, and Resource Layer. In the Resource Layer, Workers ('W') of different kinds (marked with different colors) are shown.

this request to task descriptions (in case of triggering a computation) or other messages (e.g. abort request message or response to an input request issued by a worker) and dispatch them to the appropriate queue. On the Resource Layer the workers process the messages compatible with their capabilities, and, whenever they need to communicate with the user or the system, they submit a message to another (or the same – depending on the communication purpose) queue. The Mediation Layer decides what should be further done with the message and, if needed, communicates it to the Domain Layer, which displays it correctly to the user.

The design of the framework makes it independent of the underlying infrastructure as customized workers may be deployed on any computing resources. This could prove useful also in situations where licensing issues prevent the researchers from running computations (with use of a proprietary software) on other computers than their own.

5.2 Integration Platform

The framework architecture enables creating Web components customized to the needs of a specific research domain. They are usually integrated in a standalone portal for the domain, however, they may also be composed into a larger integration platform – e.g., a Liferay portal [8]. In such case, apart from the usual InSilicoLab components (e.g. Experiment Management, Result Management, Data Management), the community would be able to use additional functionality provided by the integration platform – like Wiki pages, blogs, social networking,

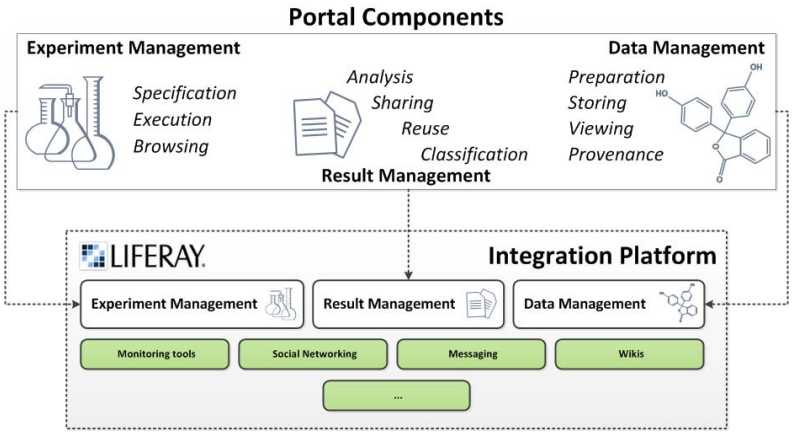

Fig. 3. Model of integrating the InSilicoLab components into an Integration Platform

etc. Furthermore, such platform as Liferay would also be capable of integrating more advanced services built for the research community – e.g. infrastructure monitoring service or other custom applications.

5.3 Science Gateway Instance Development Model

The most important aspect of developing a Science Gateway for a science community is understanding the way its researchers work. This is crucial to make the resulting solution useful and user-friendly. This cannot be done without analyzing the everyday work of a researcher, in order to extract common patterns, data flows and types, as well as tools they use. Having such knowledge, the work patterns can be recorded as science experiments understandable for a machine – algorithms. These algorithms become a base for the domain-related components required to create a new Science Gateway – i.e.: specialized workers, experiment logic (the algorithm of the experiment), and experiment-specific user interface.

The workers have a defined API and a set of utility methods they can use. Therefore, the development of a specialized worker requires only implementing several methods responsible for executing a domain application and transferring its results. The communication with the message queue, including serialization and deserialization of the messages, is implemented in the core classes each worker uses, therefore, it does not have to be separately handled by the worker developers. Likewise, communication with the Worker Manager, allowing for monitoring of the workers, is provided as a standard mechanism automatically included in each worker.

Experiment logic controls how the experiment is conducted – e.g., what operations are delegated to workers, what happens when a worker sends a reply or requests a user input. As in case of workers, this all can be implemented based on

a common API and utility methods provided by Core Services of the Mediation Layer (see 5.1).

The last component of a new experiment in a Gateway is the user interface. It should provide forms for the experiment input, interaction with it, and the results display (e.g. tables, plots). Most of the user interface components – like input fields, upload forms, charts, etc. are provided with the InSilicoLab framework, therefore, for simpler experiments, the construction of the interface requires only to create a layout of a set of ready components.

To simplify the creation of a new Science Gateway, its developers are provided with a template project containing all the required components with empty methods to fill in with custom operations.

6 Current Status

This section provides information on the current status of the implementation of the framework itself as well as the status of development of gateways based on it.

6.1 Implementation Status

An implementation of the InSilicoLab framework architecture already resulted in several running gateway instances (see 6.2) – both exposed as standalone portals and as Liferay-integrated solutions. The implementation of the processing model was already tested for batch and immediate operations, with a proof-of-concept implementation of the mechanisms for interaction with the user.

The reference implementation uses Java-based workers, started manually and monitored through the message queue interface.

6.2 Running Instances

There are three production instances of gateways based on the InSilicoLab Science Gateway framework:

- CTA Science Gateway (`http://cta-sg.grid.cyfronet.pl`) – a portal for the Cherenkov Telescope Array community, integrated along with other CTA-specific services within Liferay. The gateway can be accessed by all CTA consortium members.
- InSilicoLab for Chemistry (`http://insilicolab.grid.cyfronet.pl/`) – a standalone portal for computational chemistry, featuring Quantum Chemistry experiments with the use of standard QC packages as well as a custom library for interactive management for Molecular Dynamics trajectories.
- InSilicoLab for Astrophysics (`http://insilicolab.astro.plgrid.pl/`) – a standalone portal for MHD simulations run by the community of astrophysicists in the PLGrid Plus project [9]. The portal was developed by one of the AstroGrid-PL scientists.

Three other Science Gateway instances are currently under development: a Thematic Node for Induced Seismicity for the EPOS project [10], and two portal instances for different communities in bioinformatics.

As the core components of the system are common to all gateway instances, the process of creating a new instance is simplified, and requires providing only domain-specific components (see also 5.3). Due to this fact, a new gateway instance can be created by domain experts, with only little help and guidance from the framework developers – which was already proved by InSilicoLab for Astrophysics and the Bioinformatics portals.

7 Related Work

This section describes other science gateway framework solutions (7.1) as well as works related to individual parts of the framework described in this paper.

7.1 Science Gateways and Science Gateway Frameworks

gUSE/WS-PGRADE and SCI-BUS. gUSE/WS-PGRADE (Grid And Cloud User Support Environment) is a science gateway framework developed by Laboratory of Parallel and Distributed Systems at MTA SZTAKI (`http://www.lpds.sztaki.hu/`) [11]. SCI-BUS (SCIentific gateway Based User Support) is a project aiming at providing a science gateway customization methodology based on gUSE/WS-PGRADE for different user communities and National Grid Infrastructures (NGIs) [12]. gUSE/WS-PGRADE provides a generic purpose, workflow-oriented graphical user interface to create and run workflows on various Distributed Computing Infrastructures (DCIs) including clusters, Grids, desktop Grids and Clouds. The technology is based on Liferay [8] and Liferay portlets, which enables it also to use the Liferay's community and social functionality.

The advantages of the system are undoubtedly large range of the supported DCIs and possibility to use and share common Liferay portlets. However, such tight integration with a portal technology might also be a disadvantage in the future. What is more, generality makes it relatively easy for a developer (a computer scientist) to create a new gateway in the gUSE/WS-PGRADE technology, however, still extremely difficult for a regular researcher (see e.g. [13]).

SCI-BUS offers also an extensive support for workflows, compatible with SHIWA [14], including interactive workflow nodes that enable users to have influence on the run of a workflow. Nevertheless, usually developing a priori workflows is not convenient for regular users – they would need help of workflow developers. What is more, interaction on the level of a workflow may not be enough, as the user would often like to interact with the computational job itself – thus, the requirement of interactivity is only partially met here and the framework allows only for broad range of batch operations.

Catania Science Gateway Framework. Catania Science Gateway Framework (CSGF) [15] is a tool for Science Gateway creation, developed by INFN, Division of Catania (Italy).

Like gUSE/WS-PGRADE, the Catania Science Gateway Framework is based on Liferay, however, the authors claim that the technology is compliant with JSR-168 [16] and JSR-268 [17] portlet standards – which could make the technology less Liferay (as specific solution) dependent. Nevertheless, as in case of gUSE/WS-PGRADE, CSGF's portal installation as well as portlet development is a process that only an application developer (not a researcher) can handle easily.

CSGF offers several authorization mechanisms, including SAML-compliant solutions as well as using one of the Social Networks account.

Unlike gUSE/WS-PGRADE, which use their custom layer – gUSE, the underlying technology of CSGF gateways for communication with DCIs is JSAGA [18], which makes the technology more standards-compliant.

The computing model of CSGF is very similar to the one of gUSE/WS-PGRADE, offering mainly batch processing with little interaction.

Apache Airavata. Apache Airavata is "a software framework for executing and managing computational jobs and workflows on distributed computing resources including local clusters, supercomputers, national Grids, academic and commercial Clouds" [19].

It provides a suite for building gateway solutions that are oriented mainly at workflow construction and execution. The framework is interoperable with many existing workflow standards. Nevertheless, as mentioned before, the mechanism of constructing workflows to run them later is difficult to realize by a regular scientist.

The architecture of the solution is based on Web Services communicating over SOAP protocol, what makes it a very flexible solution. However, in its current shape the framework is focused on heavy, large-scale applications, not implementing any support for lighter (instantaneous) operations, not mentioning any interactivity on the computation level.

7.2 Grid/Cloud Computing Frameworks

DIRAC. DIRAC (Distributed Infrastructure with Remote Agent Control) [20] is a software framework for distributed computing providing a level of abstraction above known middleware providing access to distributed resources – it is thus called Interware. DIRAC attempts to bridge some of the flaws of the existing middleware by providing a custom Workload Management System with Pilot Jobs, and therefore providing more reliability and efficiency of multi-job computations.

DIRAC provides also a custom file catalogue (DIRAC File Catalogue – DFC). The Catalogue proved to be more efficient [21] than the widely used LFC catalogue. DIRAC File Catalogue offers also metadata support, which largely improves its search capabilities and, as a consequence, its usability.

The DIRAC functionality is exposed through a command-line interface or through a web portal (`http://dirac.ub.edu`) as well as through a RESTful API. The portal provides functionality allowing for job and pilot control, file catalogue search, reporting, creating dashboards and many more. All DIRAC interfaces are job-oriented, which makes them difficult to use by a science community member with no knowledge about distributed computing. Creating job descriptions is largely facilitated in the DIRAC Web Portal, but, still, the users have to operate on scripts, files, servers, etc. instead of the concepts related to their research.

The range of the interfaces DIRAC provides, along with the advantages of the DFC catalogue and the system good reliability, makes it a good candidate to use as one of the means for running the workers of InSilicoLab, however, interaction of the computation with a user has to be provided by the InSilicoLab framework as DIRAC itself does not provide any support for such model of processing.

JSAGA. JSAGA [18] is an implementation of Simple API for Grid Applications (SAGA) [22], which is an attempt to provide a common interface for grid middleware. JSAGA can be used either as a command-line tool or as a library for a Java application. It offers a flexible mechanism of plugins (adaptors) that allows the users to include only the components that are required for a concrete application. The range of the available adaptors is large and still growing thanks to community support. It includes i.a. adaptors for different middleware like gLite, DIRAC and Unicore, as well as SSH and SFTP adapters.

JSAGA was already tested and used in InSilicoLab for access to the LFC catalogue and for submission of the workers. As DIRAC, it does not provide any means for interaction with a running job, but is a good candidate for submitting batch operations.

7.3 Specific GUIs

Another group of solutions related to the functionality that Science Gateways cover, are user interfaces provided only for a concrete science domain or even dedicated to a single application. They cannot be considered as broader frameworks, however, their interfaces can be used as a point of reference for the interfaces offered in gateways.

One of such solutions is WebMO [23] – a Web interface for computational chemistry packages – supporting several of them. It allows users to draw structures in a 3D java editor, run calculations, and view results from their web browser. Although the interface is very comprehensive as a tool for chemistry, it is not adjustable to other domains of science. Its more advanced features as well as the ability to compute larger systems are available only in commercial versions – WebMO Pro and Enterprise.

There are also many other Graphical User Interfaces supporting specific applications – like GaussView [24]. However, their use limits the user to only one application, not allowing to compare results or choose different computation method for the same input data.

8 Summary

Although the needs of different science communities are very different with respect to computing, they can be summarized by putting into several classes of processing patterns. To create a fully functional and intuitive to use processing model, all of these patterns have to be implemented. Without this, the resulting tools and frameworks for computing would serve only a limited set of research activities. InSilicoLab, on one hand, implements the processing model with consideration to all the aforementioned classes, and, on the other, provides a framework for creating domain- or community- specific gateways that enable the research communities to take advantage of this model with a specialized interface.

The framework was already used for development of several Science Gateways in different science domains.

References

1. The Nobel Prize in Chemistry (2013), http://www.nobelprize.org/nobel_prizes/chemistry/laureates/2013/
2. Eilmes, A., Sterzel, M., Szepieniec, T., Kocot, J., Noga, K., Golik, M.: Comprehensive Support for Chemistry Computations in PL-Grid Infrastructure. In: Bubak, M., Kitowski, J., Wiatr, K. (eds.) PLGrid Plus. LNCS, vol. 8500, pp. 250–262. Springer, Heidelberg (2014)
3. Kocot, J., Szepieniec, T., Harężlak, D., Noga, K., Sterzel, M.: InSilicoLab – Managing Complexity of Chemistry Computations. In: Bubak, M., Szepieniec, T., Wiatr, K. (eds.) PL-Grid 2011. LNCS, vol. 7136, pp. 265–275. Springer, Heidelberg (2012)
4. The Cherenkov Telescope Array project web site, http://www.cta-observatory.org/
5. Gromacs web site, http://www.gromacs.org/
6. Protein Data Bank, http://www.pdb.org/
7. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures, dissertation. Chapter 5 (2000)
8. Liferay portal web site, http://www.liferay.com/
9. PLGrid Plus project web site, http://www.plgrid.pl/projekty/plus
10. EPOS web site, http://www.epos-eu.org/
11. gUSE web site, http://guse.hu
12. SCI-BUS project web site, http://www.sci-bus.eu/
13. Gottdank, T.: Cookbook for Gateway Developers. Solutions and Examples for Advanced Usage and Development of gUSE Components and Related Programming Interfaces (2013)
14. SHIWA web site, http://www.shiwa-workflow.eu/
15. Catania Science Gateways web site, http://www.catania-science-gateways.it/
16. JSR 168: Portlet Specification, https://jcp.org/en/jsr/detail?id=168
17. JSR 286: Portlet Specification 2.0, https://jcp.org/en/jsr/detail?id=286
18. JSAGA web site, http://grid.in2p3.fr/jsaga/
19. Apache Airavata web site, http://airavata.apache.org/
20. DIRAC web site, http://diracgrid.org/

21. Tsaregorodtsev, A., Poss, S.: DIRAC File Replica and Metadata Catalog. In: International Conference on Computing in High Energy and Nuclear Physics 2012 (CHEP 2012). Journal of Physics: Conference Series 396, 032108. IOP Publishing (2012), doi:10.1088/1742-6596
22. Goodale, T., Jha, S., Kaiser, H., Kielmann, T., Kleijer, P., Merzky, A., Shalf, J., Smith, C.: A Simple API for Grid Applications, SAGA (2013), http://www.ogf.org/documents/GFD.90.pdf
23. WebMO web site, http://www.webmo.net/index.html
24. GaussView web site, http://www.gaussian.com/g_prod/gv5.htm