Part One
WS-PGRADE/gUSE Science Gate-way Framework

1 Introduction to Science Gateways and Science Gateway Frameworks

Péter Kacsuk

Abstract. This chapter gives a short introduction to the basic architecture and functionalities of science gateways, as well as their development methods. It then briefly describes the EU FP7 SCI-BUS project that is developing a core science gateway framework called as WS-PGRADE/gUSE. A large number of various user communities have developed application-oriented science gateways by adapting and customizing the WS-PGRADE/gUSE gateway framework. The chapter also explains the vision of SCI-BUS on a collaboration-based SG instance development methodology. Finally, it gives a guide on how to read the rest of the book.

1.1 Science Gateway Frameworks and Instances

More and more scientific communities use distributed computing infrastructures (DCI) including grids and clouds. Unfortunately, directly using these infrastructures is not easy; it requires a lot of expertise and skill, and a good understanding of the working mechanisms of these infrastructures. Typical scientists like chemists, biologists, etc., do not have this required skill, and hence they require a high-level, scientific domain-specific user interface that hides all the details of the underlying infrastructure and exposes only the science-specific parts of the applications to be executed in the various DCIs.

Science gateways are the typical environments that realize these needs. They are typically provided as a web interface that can be accessed from everywhere in the world. They have the advantage that scientists do not have to install anything on their personal desktop machines or mobile devices and no matter where they travel (conferences, visiting other scientists, etc.), they can access the DCIs and run applications on them. Recognizing these advantages, more and more scientific communities have decided to build such gateways in order to simplify their use of the various DCIs.

Using the terminology introduced by the EGI Science gateway Virtual Team, science gateways (SG) can be divided into two main categories [Lovas/2013]: SG frameworks and SG instances. **SG frameworks** or generic DCI gateway frameworks are not specialized for a certain scientific area, and hence scientists from many different areas can use them. National Grid Initiatives (NGIs) are good candidates to set up such gateways to support their very heterogeneous user communities. Typical gateways belonging to this category are the Catania Science Gateway [Rotondo/2012], GridPort [Thomas/2001], Vine Toolkit [Dziubecki/2012], and WS-PGRADE/gUSE [Kacsuk/2012]. These gateways usually expose a large set of low-level services for their users. On the one hand, this is an obvious advantage, but on the other hand in order to exploit their full power, scientists need a relatively long learning period to efficiently use all the available features. The powerful but complex functionalities offered by a generic SG may be too complicated for end-users but could represent the right abstraction level for IT specialists, who can develop DCI applications for the scientists.

SG instances or application-specific SGs target a well-defined set of scientists typically working in a specific field of science. They provide a simplified user interface that is highly tailored to the needs of the given scientific community. As a result, the scientists do not have to learn too much to use the functionalities provided by the gateway. On the other hand, these services are limited, and hence if a scientist needs a more complex service, for example, utilizing a new type of DCI, this cannot be easily created and managed by these gateways. There are two options in order to create such SG instances there are two options.

The first option is to write the gateway from scratch. Since the services needed for a particular community are typically limited, and there are good technologies for the construction of web portals, like Liferay, it is relatively easy to develop such SG instances (compared with the development of an SG framework). However, such simplified gateways typically support the use of only one particular DCI and possibly do not support some advanced features such as workflow execution. Some communities selecting this option may underestimate the required manpower and time to produce a robust gateway that can be provided as a production 24/7 service for the large number of members of the community. Problems that typically arise once the gateway goes into production and becomes successful are scalability (to cope with more users than initially planned) and flexibility (to add new functions requested by the users). Moreover, while building and maintaining such gateways, the different communities usually solve again and again the same technical issues independently from each other, which could be avoided by reusing and customizing solutions implemented by SG frameworks.

The other option is to customize an existing versatile SG framework according to the needs of a certain user community. In this case the full power of the underlying portal framework can be exploited, for example, by developing comprehensive and sophisticated workflows for the community and hiding these complex workflows behind a simplified application-specific user interface. The advantage of this approach is that the DCI access services are already solved and provided in a robust way by an SG framework, and hence the user communities can concentrate on producing their application-specific layers of the science gateway. In this way the redundancy of developing the same DCI access mechanisms by many different communities can be avoided. For the same reason, the development time of SG instances can be significantly reduced, and there is a good chance that within the lifetime of the requiring project the science gateway can be built and provided as a production service. Another advantage is that the cost of producing such a gateway is usually lower than in the case of the first approach. Since the gateway is a customization of an existing robust and scalable SG framework, the resulting production SG instance will also be robust and scalable. The sustainability of such an SG instance is more certain than in the case of the first method since the large set of user communities involved in the adaptation and maybe further development of the framework represents a strong lobbing force to get further funding for maintenance and development. It is also important that the gateway framework should be open source and should involve community members in the development and maintenance of the code. When the SG framework is sustainable, the community of the SG instance should maintain only a narrow set of user-specific services, and the rest should be maintained by the SG framework developer community.

1.2 Architecture of Science Gateways

In both SG frameworks and SG instances two main components should be distinguished:

- Front-end
- Back-end

The role of the front-end is to provide the necessary user interface. In the case of SG instances the interface is very much customized to the particular needs of the scientific user community. For example, chemists and biologists would like to see visualization tools for molecules, whereas meteorologists need various types of map visualizations. The major focus of SG instances should be to develop this kind of specialized user interface to provide the right front-end for the target user community. In the case of an SG framework the interface is typically more generic, providing user interface for generic features that might be needed for many different user communities and SG instances. For example, these could include user interfaces for certificate management, file and data management, job submission, workflow creation and management, monitoring, etc. These generic parts of the front-end could also be reused from an SG framework for the implementation of customized SG instances. Quality requirements for a front-end are as follows:

- **User-friendliness:** provides intuitive user interface.
- **Efficiency:** provides fast response time even for complex user requests.
- **Scalability:** provides fast response time even for a large number of simultaneous user requests.
- **Robustness:** keeps working under any circumstances and recovers gracefully from exceptions.
- **Extensibility:** it must be easy to extend with new interfaces and functionalities.

Notice that the main difficulty of building an SG front-end is not the pretty design of the user interface but the achievement of the quality requirements listed above. These become really important when the SG is used in production by a large number of scientists. Gateways created from scratch in many cases reach only the prototype level, or if they go into production, they face a lot of difficulties to meet these quality requirements.

The back-end provides the necessary DCI access mechanisms that are needed to realize the typical gateway functionalities like certificate management, file and data management, job submission, workflow management, monitoring, etc., for various DCIs. The back-end is typically generic, i.e., the same back-end can be used by many different SG instances. Therefore the main advantage of developing SG frameworks and deriving the SG instances for them appears in the field of developing the back-ends. If a generic back-end is developed in a robust way by an SG framework, all the SG instances derived from it can take the benefit from its robustness with no or little development effort. A good back-end can support several DCI types (clusters, grids, desktop grids, clouds, etc.); therefore one of the distinguishing features of SG frameworks is how many different DCIs they can support and how easily these DCIs can be accessed via the functionalities provided by the SG framework.

Quality requirements for a back-end are similar to the front-end requirements, although their meaning could be quite different since the front-end serves users and the back-end manages jobs and service calls:

- **Efficiency:** provides fast response time even for complex submitted jobs or service calls.
- **Scalability:** provides fast response time even for a very large number (even for millions) of simultaneously submitted jobs or services calls.
- **Robustness:** keeps working under any circumstances and recovers gracefully from exceptions.
- **Flexibility:** ability to manage many different types of DCIs and many concrete instances of DCIs.
- **Extensibility:** it must be easy to extend with the support of new types of DCIs, with new concrete DCIs, and new back-end services.

1.3 Functionalities of Science Gateways

A science gateway can have many different functionalities. In fact, each user community typically requires some new functionalities according to their specific needs compared to the original, generic functionalities of the SG framework from which they derive their own SG instance. Therefore here we show only the typical functionalities that are commonly used by many different SG frameworks and SG instances. These functionalities can be grouped according to their relationship to the users and the DCIs:

- **DCI-oriented functionalities:**
    - Certificate proxy management

    - Job submission
    - Data management
    - Workflow management
    - Monitoring the usage of DCIs
    - Accounting the usage of DCIs
- **User-oriented functionalities:**
    - User certificate management
    - Workflow editing
    - Job and workflow execution progress visualization
    - Scientific visualization where requested
    - User collaboration support

In many DCIs accessing resources requires user authentication, and, unfortunately, different DCIs require different types of authentication mechanisms. If a gateway is to support access to different kinds of DCIs, then it should support all the user authentication methods required by the different DCIs. These methods include, for example, X509 certificate management and certificate proxy management. Chapter 6 "WS-PGRADE/gUSE security" describes the major authentication methods and their support in the WS-PGRADE/gUSE SG framework.

Users typically want to submit jobs to the different DCIs, and hence the job submission mechanism is a basic service in every science gateway. Again, different DCI types implement different types of job submission protocols, and a generic gateway framework should be prepared to handle all these different kinds of protocols. The WS-PGRADE/gUSE SG framework contains a generic job submission service that can submit jobs to all the major DCI types. This service, called the DCI Bridge, is described in detail in Chap. 4. Other SG frameworks also support access to several DCIs, but in a much more restricted way than is supported in WS-PGRADE/gUSE.

Jobs require access to data storage when they are executed. In many cases the different DCIs apply different storage access protocols, which also cause difficulties for gateway developers who must cope with the variety of these protocols. Executing a job in a certain DCI can require access to data storage maintained in other DCIs. To solve these problems, SCI-BUS developed the Data Avenue service that enables access to the most important storage types, even if jobs running in other DCIs. This service and its use in the WS-PGRADE/gUSE SG framework is explained in Chap. 5. Other SG frameworks typically lack this generic approach of accessing various types of data storages. Recently, the EUDAT EU FP7 project also started to develop a generic solution for this problem [Riedel/2013]**.**

Beyond simple job submissions and service calls, applications solving complex problems like scientific simulations require the creation and execution of scientific workflows. To support these more advanced types of applications, SG frameworks should provide workflow editing and execution services. Recently, more and more SG frameworks have such workflow support. The WS-PGRADE/gUSE SG framework was designed from the very beginning to include workflow management. This capability of WS-PGRADE/gUSE is described in detail in Chap. 3.

As jobs and workflows are executed in the various DCIs, users should be able to observe how their execution is progressing. Therefore the gateway back-end should be able to collect execution monitoring information from the DCIs, and the front-end component should be able to present this information to the users in a comprehensive way. This is such a basic requirement that it is typically supported by every SG framework. On the other hand providing accounting information on how many resources for what price have been used during job and workflow execution is also an important service of science gateways but is frequently neglected and not supported. The WS-PGRADE/gUSE SG framework provides such accounting service for commercial clouds when it is used together with the CloudBroker platform. This facility is explained in Chap. 6.

User collaboration is needed both inside a user community and among several user communities. WS-PGRADE/gUSE provides an internal application repository for collaboration inside a user community, and access to the SHIWA Workflow Repository in order to help external collaboration among different user communities. These services of WS-PGRADE/gUSE are described in Chap. 9.

Tools for scientific visualization are typically provided by SG instances and not by SG frameworks since scientific visualization is application-dependent. Therefore such tools and services are described in Chaps. 10–15, where the SG instances derived from the WS-PGRADE/gUSE SG framework are introduced.

1.4 Developers and Users of Science Gateways

People involved in the creation, operation, and usage of gateways have different roles, and a good gateway should provide support for all the roles.

The first category is the *gateway developers*, who develop the gateways. Here we have to distinguish *SG framework developers* and *SG instance developers*. The primary goal of SG framework developers is to develop the SG framework back-end in a portable way that enables SG instance developers to use it without modifications. Their second goal is to develop the generic part of the front-end, and obviously also generate and maintain up-to-date documentation. Beyond these tasks directly related to the gateway framework development, they should also provide user support, including the evaluation of feature requests and further developing the gateway framework according to the new functionality requirements. Developing an SG framework requires very deep understanding of the underlying infrastructures and the required web technologies. Therefore, to develop an SG framework the developer community should invest in a long-running and constant learning process, which is very costly. As a result, there are only very few SG frameworks, and the number of gateway framework developers is also very low.

The main task of the SG instance developers is either to customize an existing SG framework for their user community, i.e., to extend the SG framework with new application-specific interfaces, or to develop the SG instance from scratch. In the former – and recommended – case SG instance developers can concentrate on the application domain-specific features of their SG. In the latter case, they need to learn all those aspects of the underlying DCI middleware and web technologies that are needed for the SG framework developers. As a result, they usually create the SG instance much more slowly and with more efforts than those SG instance developers who choose the customization development method. The number of SG instance developers is about an order of magnitude larger than the number of SG framework developers, but in the ideal case, the difference would be even two orders of magnitude. In the case of the WS-PGRADE/gUSE framework we get close to this ideal case, since the framework has been adapted by more than 90 different communities who develop SG instances based on the framework. The WS-PGRADE/gUSE framework helps this customization process by providing a special API called Application Specific Module (ASM) API by which existing workflows can easily be embedded in application specific portlets (see details in Chap. 3).

Once the SG frameworks or SG instances are developed, they should be set up and operated. Here the role of *gateway operators* comes into play. They should be able to deploy, configure, run, and maintain the gateway service for the user communities. For these purposes, good gateways provide complete and up-to-date documentation, installation and configuration wizards, user management support interfaces, etc. These can be developed in a generic way within an SG framework and just be used (and maybe adapted) by SG instances.

Once the SG frameworks or SG instances are set up and operating, they are ready for use. We must distinguish two user categories: *end-users* and *application developers*. In fact, they need different front-ends. The application developers develop DCI applications, for example, new workflows, which are used by the end-users. The application developers are typically IT people or scientists (chemists, etc.) with good understanding of the underlying IT technology. They should have relatively detailed information on the underlying DCIs, while this information could partially or completely be hidden from the end-users. Therefore, the SG frameworks are primarily targeted to the application developers, and the SG instances are typically designed for the end-users. Of course, this typical usage does not exclude the possibility that some SG frameworks can be used by end-users and SG instances can provide front-ends necessary for DCI application development. However, a good practice is the clear separation and support of these two user types, and WS-PGRADE/gUSE supports this concept. It provides a full-scale user interface for workflow developers (called power users) that enables the fast and efficient development of DCI-oriented workflows. On the other hand, its end-user interface concept enables the automatic creation of an end-user interface with limited functionality that can be easily used by scientists who do not know the underlying DCIs. This aspect of the WS-PGRADE/gUSE gateway framework is described in a more detailed way in Chaps. 2 and 8.

1.5 The SCI-BUS Project

As written in Sect. 1.1, the recommended way to develop SG instances is the customization methodology. This approach is followed by the SCI-BUS (Science Gateway Based User Support, https://www.sci-bus.eu) EU FP7 project that develops the WS-PGRADE/gUSE SG framework and also a customization technology by which a large number of scientific user communities can easily adapt the framework and develop their SG instance. The structure of the project and the related applied technologies are shown in Fig. 1.1.

The central component of the project is the WS-PGRADE/gUSE gateway framework. This is the basis of all the SG instances developed by project partners, subcontractors, and associated partners. During the project the WS-PGRADE/gUSE framework has been significantly further developed, including the following main features:

1. Cloud integration via the CloudBroker Platform (this is described in detail in Chap. 7) to access a large variety of commercial and academic clouds
2. Direct cloud integration to access academic clouds (see details in Chap. 4)
3. To provide robot certificates (see details in Chap. 6)
4. To provide an efficient and flexible data management system over various DCIs (see details in Chap. 5)
5. To extend the workflow management system with workflow debugging capabilities (see details in Chap. 2)

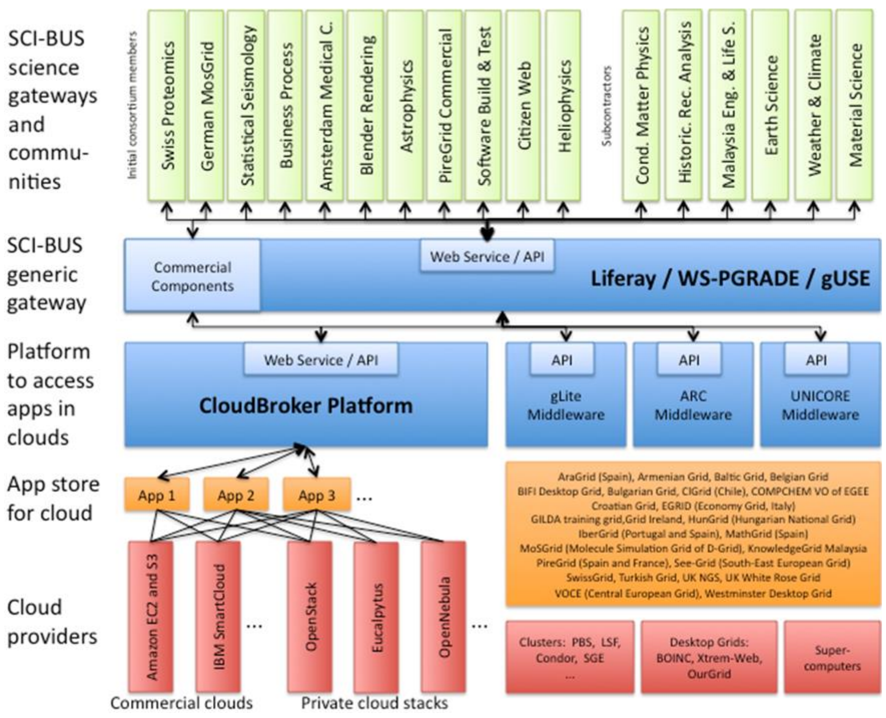
SCI-BUS science gateways and commu-nities
Initial consortium members
Swiss Proteomics
German MosGrid
Statistical Seismology
Business Process
Amsterdam Medical C.
Blender Rendering
Astrophysics
PireGrid Commercial
Software Build & Test
Citizen Web
Heliophysics
Subcontractors
Cond. Matter Physics
Historic. Rec. Analysis
Malaysia Eng. & Life S.
Earth Science
Weather & Climate
Material Science
SCI-BUS generic gateway
Commercial Components
Web Service / API
Liferay / WS-PGRADE / gUSE
Platform to access apps in clouds
Web Service / API
CloudBroker Platform
API
gLite Middleware
API
ARC Middleware
API
UNICORE Middleware
App store for cloud
App 1
App 2
App 3
...
AraGrid (Spain), Armenian Grid, Baltic Grid, Belgian Grid
BIFI Desktop Grid, Bulgarian Grid, CIGrid (Chile), COMPCHEM VO of EGEE
Croatian Grid, EGRID (Economy Grid, Italy)
GILDA training grid,Grid Ireland, HunGrid (Hungarian National Grid)
IberGrid (Portugal and Spain), MathGrid (Spain)
MoSGrid (Molecule Simulation Grid of D-Grid), KnowledgeGrid Malaysia
PireGrid (Spain and France), See-Grid (South-East European Grid)
SwissGrid, Turkish Grid, UK NGS, UK White Rose Grid
VOCE (Central European Grid), Westminster Desktop Grid
Cloud providers
Amazon EC2 and S3
IBM SmartCloud
...
OpenStack
Eucalyptus
OpenNebula
...
Clusters: PBS, LSF, Condor, SGE ...
Desktop Grids: BOINC, Xtrem-Web, OurGrid
Super-computers
Commercial clouds
Private cloud stacks

Fig. 1.1 **SG instance development methodology and required services (with permission of CloudBroker GmbH)**

Of course, not only was the functional extension a major goal in the project but it also made the framework robust and efficient in the sense that a large number of users (in the range of 100–1000) could simultaneously use it with short response times and the gateway should be able to handle even millions of simultaneous job submissions. Another important aspect was the improvement of the gateway installation procedure, for which an installation and a service wizard have been developed. The documentation of the framework was also significantly improved. It contains 14 documents in the following 4 series:

1. Blue series for end-users (2 documents)
2. Green series for gateway administrators (5 documents)
3. Red series for workflow developers (3 documents)
4. Orange series for general purposes (4 documents)

The gateway framework is published at SourceForge (https://sourceforge.net/projects/guse/) and has become very popular. There have been over 15,000 downloads as of the writing this book. The user forum is very active, and nearly 200 different topics are discussed by a large number of participants. The further development of the WS-PGRADE/gUSE gateway framework will not be stopped when the SCI-BUS project is over at the end of September 2014. The project has also developed a sustainability plan that, together with the large number of users, guarantees the further progress of the WS-PGRADE/gUSE gateway framework. A roadmap of development goals with their expected deadline is found on the SCI-BUS web page (http://www.sci-bus.eu), which will be maintained even after SCI-BUS project is finished.

As Fig. 1.1 shows, 11 communities as project partners have develop application-specific SG instances based on the WS-PGRADE/gUSE gateway framework. These SG instances are the following:

1. Swiss proteomics gateway
2. MoSGrid gateway (see details in Chap. 11)
3. Statistical seismology gateway (see details in Chap. 12)
4. Business process gateway
5. Computational neuroscience gateway developed by Amsterdam Medical Center (see details in Chap. 10)
6. Blender rendering gateway
7. VisIVO astrophysics gateway (see details in Chap. 13)
8. PireGrid commercial community gateway
9. Software building and testing gateway (see details in Chap. 19)
10. Document Archiving Gateway for citizen web community (see details in Chap. 19)
11. Heliophysics gateway (see details in Chap. 14)

Subcontractors of SCI-BUS have also developed SG instances as listed below:

1. Science gateway for condensed matter physics community (see details in Chap. 15)
2. Weather Research and Forecasting science gateway developed by University of Cantabria
3. Academic Grid Malaysia Scientific Gateway
4. AdriaScience Gateway developed by Ruđer Bošković Institute
5. Metal physics science gateway of the G.V.Kurdyumov Institute for Metal Physics
6. ChartEX Gateway developed by Leiden University

The condensed matter physics gateway is described in detail in Chap. 15 but the other subcontractors' gateways are not detailed in this book due to the size limitations of the book. The interested reader can find details of these gateways in the public deliverable D6.2 of SCI-BUS under the title "Report on developed and ported applications and application-specific gateways" that is accessible at the SCI-BUS web page. Figure 1.2 shows those communities who have some relationship with SCI-BUS to build their science gateway instances. Beyond these communities there are many others without any relationship with SCI-BUS that also intensively use the SCI-BUS gateway technology.

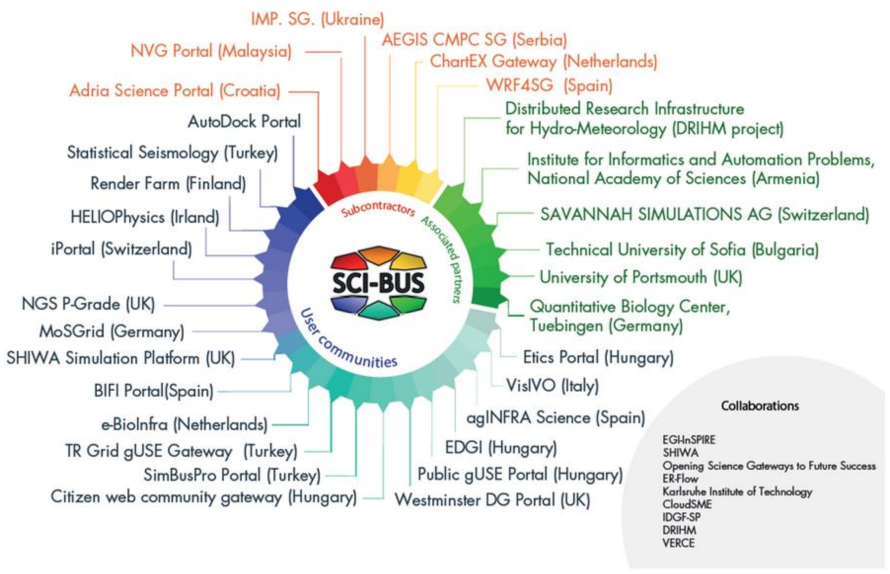
IMP. SG. (Ukraine)
NVG Portal (Malaysia)
AEGIS CMPC SG (Serbia)
ChartEX Gateway (Netherlands)
Adria Science Portal (Croatia)
WRF4SG (Spain)
Distributed Research Infrastructure for Hydro-Meteorology (DRIHM project)
AutoDock Portal
Statistical Seismology (Turkey)
Institute for Informatics and Automation Problems, National Academy of Sciences (Armenia)
Render Farm (Finland)
HELIOPhysics (Irland)
SAVANNAH SIMULATIONS AG (Switzerland)
iPortal (Switzerland)
Technical University of Sofia (Bulgaria)
University of Portsmouth (UK)
Subcontractors
Associated partners
SCI-BUS
User communities
NGS P-Grade (UK)
Quantitative Biology Center, Tuebingen (Germany)
MoSGrid (Germany)
SHIWA Simulation Platform (UK)
Etics Portal (Hungary)
BIFI Portal(Spain)
VisIVO (Italy)
agINFRA Science (Spain)
e-BioInfra (Netherlands)
TR Grid gUSE Gateway (Turkey)
EDGI (Hungary)
SimBusPro Portal (Turkey)
Public gUSE Portal (Hungary)
Citizen web community gateway (Hungary)
Westminster DG Portal (UK)
Collaborations
EGI-InSPIRE
SHIWA
Opening Science Gateways to Future Success
ER-Flow
Karlsruhe Institute of Technology
CloudSME
IDGF-SP
DRIHM
VERCE

Fig. 1.2 SG instance developer communities using SCI-BUS technology (with permission of Elisa Cauhé Martín)

1.6 Collaboration-based SG Instance Development Methodology

SCI-BUS technology helps the collaboration among the different types of people developing and using the gateway technology. As already mentioned, two different level repositories help collaboration between workflow developers and workflow users. Inside a community using the same gateway, the internal gUSE Application Repository can be used for workflow developers to publish the ready-to-use workflows, and scientists in the end-user mode of the gateway can import these ready-to-use workflows from the Application Repository. After parameterizing the workflows they can be executed in the target DCIs. Of course, the Application Repository can also be used to support collaboration between workflow developers. A workflow stored in the Application Repository can be taken by any workflow developer belonging to the same gateway's community and can extend or further develop the imported workflow. Similar activities are supported among workflow developers and end-users belonging to different gateway communities via the SHIWA Workflow Repository. Using the coarse-grained workflow interoperability technique developed in the SHIWA project, this repository and the WS-PGRADE/gUSE gateway enable collaboration even in cases when the different communities use different workflow systems (see details in Chap. 8).

Collaboration is supported not only among workflow developers and workflow users but also among gateway developers. For this purpose SCI-BUS developed and set up the SCI-BUS Portlet Repository. This enables the sharing of Liferay portlets between SG instance developers (see details in Chap. 9). This sharing of existing portlets can further accelerate the customization process of gateway frameworks.

In fact, these repositories, the SG framework stored in the open source SourceForge repository and the customization concept of SCI-BUS enable the introduction of a collaborative SG instance development methodology. Figure 1.3 shows the services required for the SG instance development methodology as well as the different types of developers and users related to the SG instance. The steps in developing an SG instance according to this SG instance development methodology are as follow:

- Step 1: An SG instance developer downloads the WS-PGRADE/gUSE framework from SourceForge and deploys it as a general purpose science gateway. It contains the major functionalities to develop and run workflows by the workflow developers and to run workflows by the end-user scientists.
- Step 2: An SG instance developer downloads several domain-specific portlets from the SCI-BUS Portlet Repository that are needed for the target user community. At this stage, without any development the community already has a domain-specific gateway. Although it may not be perfectly what they want, the users can start to work with it.
- Step 3: An SG instance developer downloads several domain-specific workflows from the SHIWA Workflow repository and develops new domain-specific portlets on top of them. At this stage, without any workflow development the community already has an improved domain-specific gateway; although it is not perfectly what they want, the users can have more portlets to work with. For the sake of mutual collaboration, the SG instance developer uploads the new portlets into the SCI-BUS Portlet Repository so other communities can take advantage of using these new portlets.
- Step 4: The workflow developer develops new domain-specific workflows and uploads them to the SHIWA Workflow Repository. She might download other workflows from the SHIWA Repository and use them to develop new workflows.
- Step 5: An SG instance developer develops new domain-specific portlets on top of the workflows developed in step 4. At this stage the domain-specific gateway is extended with new portlets specifically designed according to the needs of this community. For the sake of mutual collaboration, the SG instance developer uploads the new portlets into the SCI-BUS Portlet Repository so other communities can take advantage of using these new portlets.

Of course, steps 2-5 can be repeated in as many times as required. Every iteration results in a further improved and extended SG instance for the user community.

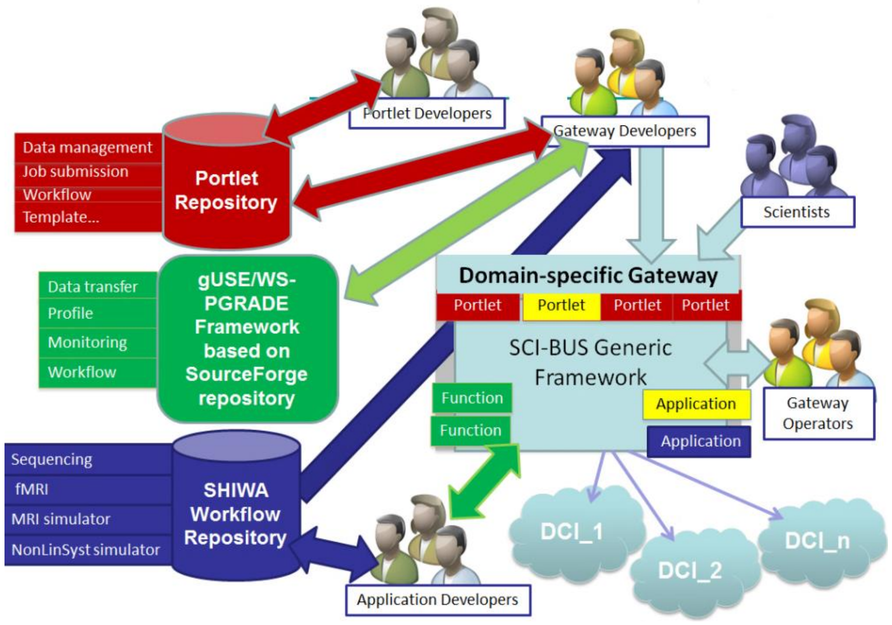

Fig. 1.3 Collaboration-based SG instance development methodology and required services

1.7 How to Read this Book?

The main goal of the book is to transfer the knowledge of building science gateways for those communities who would like to develop their own science gateway instance in the future or who would like to extend or improve their existing science gateway with new functionalities, services, portlets, and workflows. The book summarizes those technologies that we have developed in the SCI-BUS project concerning building general-purpose science gateway frameworks as well as customizing the framework toward domain- and application-specific science gateway instances. Since workflows play more and more important roles in IT-based scientific research, we also show how the SCI-BUS workflow technology can be used and extended with other workflows by using the workflow interoperability technology developed in the EU FP7 SHIWA project and currently actively used in the EU FP7 ER-Flow project (see Chap. 8).

The book is divided into three main parts. After the current chapter, the first part describes the core SCI-BUS gateway framework technology, WS-PGRADE/gUSE. Chapter 2 gives a generic introduction to WS-PGRADE/gUSE science gateway framework technology and summarizes the main features of WS-

PGRADE/gUSE. Since all the other chapters are built on the knowledge described in this chapter it is recommended that everyone read this chapter. Similarly, reading of Chap. 8 is also recommended for every reader since it explains all the major use-case scenarios where the gateway framework can be applied.

Chapter 4 describes the DCI Bridge service that enables access to a large set of DCIs via a common interface based on the OGF standard BES. Since any workflow systems and existing gateways can be extended to be able to exploit this service, any reader who is interested in extending their workflow system and gateway with access to such a large set of DCIs should read this chapter. Similarly, Chapter 7 describes the Data Avenue service that enables file transfer between different DCI storages having different protocols. This is a very generic service that can be used independently from WS-PGRADE/gUSE, and hence readers who would like to extend their workflow manager and gateway to exploit this service should read this chapter.

The following chapters should be read by those readers who are interested in learning more on the following aspects of WS-PGRADE/gUSE:

- Workflow concepts of WS-PGRADE/gUSE (Chap. 3)
- Executing WS-PGRADE workflows in various Distributed Computing Infrastructures and the DCI Bridge service (Chap. 4).
- Security aspects of WS-PGRADE/gUSE (Chap. 6)
- Integration of WS-PGRADE/gUSE and clouds via the CloudBroker Platform (Chap. 7)
- Data management in WS-PGRADE/gUSE and the Data Avenue service (Chap. 5)
- Usage scenarios by WS-PGRADE/gUSE (Chap. 8)
- Community activity support in WS-PGRADE/gUSE via the SHIWA technology and ER-Flow experience (Chap. 9)

The second part of the book contains concrete use cases that describe how the WS-PGRADE/gUSE gateway framework was customized by SCI-BUS project partners and subcontractors to develop a domain-specific science gateway instance. These chapters are completely independent from each other but they use different features of the WS-PGRADE/gUSE framework; hence they are built on information described in various chapters in the first part of the book. These chapters are very useful for those readers who also want to develop a domain-specific science gateway instances because here they can find many good ideas on how to adapt WS-PGRADE/gUSE gateway for their own purposes.

Some further gateway instance examples that were developed in other EU FP7 projects like agINFRA, DRIHM, and VERCE are shown in Chap. 17 in the third part of the book. Chapter 18 even shows how different user communities can come together and create a science gateway alliance based on the same gateway technology. Notice that there are many more science gateway instances developed based on the WS-PGRADE/gUSE gateway framework, but due to the restricted size of the book those are not described here. However, the interested reader can find those further use cases via the SCI-BUS web page. Part 3 also describes some further application areas of the SCI-BUS gateway technology. These include educational and commercial uses. Those readers who are interested in the use of SCI-BUS technology in university courses are recommended to read Chap. 16. The commercial use of SCI-BUS technology is also possible and was exploited by several companies in the SCI-BUS project; there are other companies that are currently working on the commercial applications inside the EU FP7 CloudSME project. These commercial applications of the SCI-BUS technology are described in Chap. 19.

The book ends with a short Conclusions and outlook in which the future of the SCI-BUS technology is covered.

1.8 Conclusions

The goal of the current book is to describe the WS-PGRADE/gUSE SG framework, its customization technology and to show use cases for several user communities where this technology was successfully applied to create application-specific SG instances. Within the SCI-BUS project 11 partner user communities established their own SG instances as production services, another 6 communities as sub-contractors have developed their gateways and 7 associated partners also use the SCI-BUS gateway technology. The WS-PGRADE/gUSE SG framework is an open source software that can be downloaded from SourceForge. The number of downloads is over 15.000 as of writing this book and constantly grows. There are more than 90 SG instances are deployed world-wide as shown by the google map at https://guse.sztaki.hu/MapService/. The technology therefore matured enough to be used by large number of user communities and hence the significance of this book is to disseminate this know-how for the scientific communities who are interested in building gateways based on such a matured technology that SCI-BUS can provide.