# A Methodology for Developing Scientific Software Applications in Science Gateways: Towards the Easy Accessibility and Availability of Scientific Applications

A thesis submitted for the degree of Doctor of Philosophy (Ph.D.)

By

Adedeji Oyekanmi Fabiyi

Department of Computer Science

November 2017

# Abstract

Distributed Computing Infrastructures (DCIs) have emerged as a viable and affordable solution to the computing needs of communities of practice that may require the need to improve system performance or enhance the availability of their scientific applications. According to the literature, the ease of access and several other issues which relate to the interoperability among different resources are the biggest challenges surrounding the use of these infrastructures.

The traditional method of using a Command Line Interface (CLI) to access these resources is difficult and can make the learning curve quite steep. This approach can result in the low uptake of DCIs as it prevents potential users of the infrastructures from adopting the technology. Science Gateways have emerged as a viable option that are used to realise the high-level scientific domain-specific user interfaces that hide all the details of the underlying infrastructures and expose only the science-specific aspects of the scientific applications to be executed in the various DCIs. A Science Gateway is a digital interface to advanced technologies which is used to provide adequate support for science and engineering research and education. The focus of this study therefore is to propose and implement a Methodology for dEveloping Scientific Software Applications in science GatEways (MESSAGE). This will be achieved by testing an approach which is considered to be appropriate for developing applications in Science Gateways.

In the course of this study, several Science Gateway functionalities obtained from the review of literature which may be utilised to provide services for different communities of practice are highlighted. To implement the identified functionalities, this study utilises the methodology for developing scientific software applications in Science Gateways. In order to achieve this purpose, this research therefore adopts the Catania Science Gateway Framework (CSGF) and the Future Gateway approach to implement the methods and ideas described in the proposed methodology, as well the essential services of Science Gateways discussed throughout the thesis. In addition, three different set of scientific software applications are utilised for the implementation of the proposed methodology. While the first application primarily serves as the case study for implementing the methodology discussed in this thesis, a second application is used to evaluate the entire process. Furthermore, several other real-life scientific applications developed (using two distinctly different Science Gateway frameworks) are also utilised for the purpose of evaluation. Subsequently, a revised MESSAGE methodology for developing scientific software applications in Science Gateways is discussed in the latter Chapter of this thesis.

Following from the implementation of both scientific software applications which sees the use of portlets to execute single experiments, a study was also conducted to investigate ways in which Science Gateways may be utilised for the execution of multiple experiments in a distributed environment. Finally, similar to making different scientific software applications accessible and available (worldwide) to the communities that need them, the processes involved in making their associated research outputs (such as data, software and results) easily accessible and readily available are also discussed.

The main contribution of this thesis is the MESSAGE methodology for developing scientific software applications in Science Gateways. Other contributions which are also made in different aspects of this research include a framework of the essential services required in generic Science Gateways and an approach to developing and executing multiple experiments (via Science Gateway interfaces) within a distributed environment. To a lesser extent, this study also utilises the Open Access Document Repository (OADR) (and other related technologies) to demonstrate accessibility and availability of research outputs associated with specific scientific software applications, thereby introducing the concept (and thus laying the foundation) of an Open Science research.

# Acknowledgements

First and foremost, I would like to thank God almighty for His sustenance and guidance throughout the course of this work, and for the grace to complete this programme.

My heartfelt thanks and appreciation go to my supervisor, Prof. Simon Taylor, for his attention, guidance, support and motivation from the beginning of this work. I count myself fortunate to be one of his students and to work under his supervision. Many thanks to him for always being there whenever his attention was needed. I would also like to show my appreciation to my second supervisor, Dr. David Bell, for his precious advice.

My deepest gratitude goes to my parents, Hon. Justice John Afolabi Fabiyi (Rtd) and Mrs Helen Taiwo Fabiyi, for the love, prayers, moral and financial supports. Without it, I would not have been able to achieve this dream. Also, I would like to thank my adorable wife, Princess Toluwalase Hannah Fabiyi, for her moral support, love and patience. Thank you ever so much for making this journey an even more beautiful experience.

I would also like to acknowledge my gratitude to my friends, colleagues and staff in the department of Computer Science at Brunel University. Special thanks go to Dr. Anastasia Anagnostou and Dr. Nouman Chaudhry for their support and advice at various times in the course of this work.

Last but not least, I would like to thank the Science Gateway development team at Catania University, Italy, for the support they provided in granting access to the computing resources utilised throughout the course of this work. Many thanks go to Prof. Roberto Barbera, Dr. Bruce Becker and Dr. Mario Torrisi (in particular) for all the technical support that was provided while deploying the portlets on the Science Gateway.

Thank you all…

# Declaration

I hereby declare that the research presented in this thesis is my own work except where otherwise stated, and has not been submitted for any other degree.

Adedeji Oyekanmi Fabiyi

Part of this thesis has been disseminated as below:

Fabiyi, A.O., Taylor, S.J., Anagnostou, A., Torrisi, M. and Barbera, R., 2016, September. Investigating a Science Gateway for an Agent-Based Simulation Application Using REPAST. In Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on (pp. 29-36). IEEE.

Taylor, S.J., Fabiyi, A., Anagnostou, A., Barbera, R., Torrisi, M., Ricceri, R. and Becker, B., 2016, September. Demonstrating open science for modeling & simulation research. In Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on (pp. 191-192). IEEE.

Taylor, S.J., Anagnostou, A., Fabiyi, A., Currie, C., Monks, T., Barbera, R. and Becker, B., 2017, December. Open science: Approaches and benefits for modeling & simulation. In Simulation Conference (WSC), 2017 Winter (pp. 535-549). IEEE.

# Abbreviations

| | |
|---|---|
| AJAX | Asynchronous JavaScript And XML |
| API | Application Programming Interface |
| ASC | Astrophysics Scientific Collaboration |
| ASM | Application Specific Module |
| BES | Basic Execution Service |
| CAS | Central Authentication Service |
| CLI | Command Line Interface |
| COA | Component Oriented Architecture |
| CoG | Commodity Grid Toolkits |
| COTS | Commercial-Off-The-Shelf |
| CPU | Central Processing Unit |
| CSGF | Catania Science Gateway Framework |
| DCI | Distributed Computing Infrastructure |
| DECIDE | Diagnostic Enhancement of Confidence by an International Distributed Environment |
| DEISA | Distributed European Infrastructure for Supercomputer Applications |
| DOI | Digital Object Identifiers |
| DSR | Design Science Research |
| EGEE | Enabling Grids for E-science in Europe |
| EGI | European Grid Infrastructure |
| EJB | Enterprise JavaBeans |

| | |
|---|---|
| EU | European Union |
| GCE | Grid Computing Environments |
| GENIUS | Grid Enabled web eNvironment for site Independent User Job Submission |
| GISELA | Grid Initiatives for e-Science virtual communities in Europe and Latin America |
| GrIDP | Grid Identity Pool |
| GPDK | Grid Portal Development Toolkit |
| GRB | Grid Resource Broker |
| GSI | Grid Security Infrastructure |
| GSP | Grid Service Provider |
| GUI | Graphical User Interface |
| gUSE | grid and cloud User Support Environment |
| HPC | High Performance Computing |
| HTTP | Hypertext Transfer Protocol |
| IaaS | Infrastructure-as-a-Service |
| ICT | Information and Communication Technologies |
| IdF | Identity Federations |
| IdP | Identity Providers |
| IM | Infection Model |
| JDK | Java Development Kit |
| JRE | Java Runtime Environment |
| JSDL | Job Service Description Language |

| | |
|---|---|
| JSP | Java Server Pages |
| JSR | Java Specification Requests |
| LDAP | Lightweight Directory Access Protocol |
| LEAD | Linked Environments for Atmospheric Discovery |
| MDS | Monitoring and Discovery System |
| MESSAGE | Methodology for dEveloping Scientific Software Applications in science GatEways |
| MPI | Message Passing Interface |
| MVC | Model View Controller |
| NIST | National Institute of Standards and Technology |
| NWS | Network and Weather Service |
| OADR | Open Access Document Repository |
| OGCE | Open Grid Computing Environment |
| OGF | Open Grid Forum |
| OKF | Open Knowledge Foundation |
| ORCID | Open Researcher and Contributor ID |
| ORM | Object Relational Mapping |
| PaaS | Platform-as-a-Service |
| PDA | Personal Digital Assistant |
| PRACE | Partnership for Advanced Computing in Europe |
| PS | Parameter Study |
| RAD | Rapid Application Development |
| RAM | Random Access Memory |

| | |
|---|---|
| REPAST | Recursive Porous Agent Simulation Toolkit |
| RR | Researcher Registries |
| SaaS | Software-as-a-Service |
| SAGA | Simple API for Grid Applications |
| SAML | Security Assertion Markup Language |
| SDLC | Software Development Life Cycle |
| SG | Science Gateway |
| SOA | Service Oriented Architecture |
| SSH | Secure Shell |
| SSO | Single Sign-On |
| SWMS | Scientific Workflow Management System |
| UNICORE | Uniform Interface to Computing Resources |
| USB | Universal Serial Bus |
| VAS | Virtual Application Service |
| VDS | Virtual Data Systems |
| VM | Virtual Machine |
| VO | Virtual Organisation |
| VOMS | Virtual Organisation Membership Service |
| VPN | Virtual Private Network |
| WAN | Wide Area Network |
| WEKA | Waikato Environment for Knowledge Analysis |
| WRF | Weather Research and Forecast |

| | |
|---|---|
| WSDL | Web Services Description Language |
| WS-PGRADE | Web Service Parallel Grid Runtime and Developer Environment |
| XSEDE | Extreme Science and Engineering Discovery Environment |

# Table of Contents

## Contents

# List of Tables

# List of Figures

# Chapter 1: INTRODUCTION

## *1.1 Overview*

The use of Distributed Computing Infrastructures (DCIs) can help to increase system performance and enhance the availability of scientific software applications for different communities of practice. Distributed Computing can be defined as a model in which components located on networked computers communicate and coordinate their actions by passing messages. This computing model ensures that scientists have infinite set of resources to execute their scientific jobs more efficiently. However, using interfaces such as the command line to access these infrastructures present a different set of challenges especially to scientisits in different fields who are not necessarily Information and Communication Technologies (ICT) expert users. Consequently, Science Gateways are now being used to realise a high-level scientific domain-specific user interfaces which hide all the details of the underlying infrastructures and exposes only the science-specific aspects of scientific applications to be executed in the various DCIs. These Science Gateways are digital interfaces to advanced technologies which are used to provide adequate support to science and engineering research and education.

The research presented in this thesis is an investigation of the methods and approaches that could benefit the development of scientific software applications in Science Gateways. In particular, this thesis has emphasised on the development of methodologies for creating scientific applications in Science Gateways which could ultimately be used (by different communities of practice) for the execution of scientific jobs in a distributed environment. These communities are network of peers with diverse skills and experience in an area of practice or profession who may need to utilise these resources to achieve a common goal. In addition, DCI resources (which are usually tightly coupled with associated Science Gateways) are used to address the needs of researchers for digital services in terms of networking, computing and data management. A variety of working methods based on the shared use of ICT tools and resources across different domains are used for scientific endeavours. The tools include high-speed research communication networks, powerful computational resources (dedicated high performance computers, clusters, large numbers of commodity PCs), grid and cloud technologies, data infrastructures (data sources, scientific literature), sensors, web based portals, scientific gateways and mobile devices. All the aforementioned ICT tools are collectively known as e-Infrastructures.

Ultimately, the methodology presented in this thesis will therefore address the challenges which developers may face in the process of developing scientific software applications in Science Gateways for the different communities of practice. In light of the above, the focus of this study is to understand and address the major issues, methods and processes involved in the development of scientific software applications in the context of Science Gateways and e-Infrastructures and thus establish a methodology which ultimately develops into a road map for creating scientific applications in Science Gateways for scientific endeavours. The proposed methodology is implemented and evaluated using a specific Science Gateway framework and several real life scientific applications which are discussed much later in the thesis.

In addition, this study also investigates and demonstrates an open science approach to research by using the research outputs associated with specific scientific software application (i.e an Infection Model simulation output results) used throughout this research. Therefore, similar to making scientific software applications accessible and readily available, a demonstration is presented to show how the associated research outputs can easily be accessible to the communities that need them. This demonstration is performed by using the Open Access Document Repository (OADR) and other enabling technologies such as Digital Object identifiers (DOI), and the Open Researcher and Contributor ID (ORCID).

For an overview, this Chapter briefly describes the research context and rationale as well as the research approach that is taken to conduct the study. This thesis investigates methods and approaches for developing scientific software applications in Science Gateways and examines alternative approaches to executing jobs in distributed environments, other than the traditional method of the command line interface, etc.  In view of this, this thesis introduces the evolution of distributed systems from the early days of mainframe computing to the more recent rise in the use of cloud computing. Furthermore, the research motivations, research aim and objectives as well as the adopted research methodology which will be used to fulfil the aim and meet the objectives are presented. This Chapter concludes with the structure of the remainder of the document which discusses an overview of each Chapter in this thesis.

## *1.2 Context and Rationale*

There is a steady shift (with respect to the evolution of distributed computing) from centralised/mainframe computing to cluster computing (in the 80's/90's) to grid computing (in the 90's and 00's) and to the recent use of cloud computing. This evolution was made possible due to the combined actions of the increased power of personal computers and the growing influence of the internet. These efforts coupled with the reduction of the cost of hardware/network allowed for such evolution. As a result, affordable computing resources and powerful machines with large storage and memory capacity have become readily available.

The evolution of these computing models therefore presents a new perspective for many communities of practice who may need to collaborate with their peers or perform experiments and obtain results promptly. For instance in Modelling and Simulation (M&S) experiments, as models increase in size and complexity there could be need to increase computational power as it may ultimately affect the execution time of an experiment. A scientific domain of this nature could benefit immensely from performing operations and executing scientific jobs in distributed environments (such as the cloud) as this will ensure that not only will scientific jobs complete in a timely manner, but the scientific software applications which are used to execute those jobs are easily accessible and readily available. The easy accessibility and availability of these resources can therefore be fostered by using high-level scientific domain-specific user interfaces which Science Gateways provide. In light of the above, this research therefore proposes a systematic approach to developing scientific software applications in Science Gateways for the execution of scientific jobs in distributed environments. This systematic approach is a conceptual plan which is formulated by analysing different use-case attributes, functionalities and requirements which were identified from the review of literature.

For many decades, several application domains have made use of high-performance computers to solve many scientific and technical problems and to execute complex calculations (Limet, Smari and Spalazzi, 2015). Among them is the field of M&S applications where scientific simulations generate complex codes where data size and computation time (if not properly managed) could become a scientific barrier (Coullon and Limet, 2015). Also, there has been a high increase in the use of High-Performance Computing (HPC) in the field of life sciences and healthcare and more recently, HPC infrastructures are used in the mining of biomedical data with a special focus on the analysis of genomics, interatomic data and the exploration of magnetic resonance images in neurosciences (Cannataro, Guzzi and Sarica, 2013). Similarly, this research will also make use of DCIs and HPC principles to execute an Infection Model (via a Science Gateway interface) and the results obtained from the Infection Model will be analysed using WEKA's J48 classifier.

DCIs are used to provide services such as computational services, data services, application services, information services and knowledge services, and it consists of features that are used to provide potential users with a seamless computing environment (Baker, Buyya and Laforenza, 2002). It is a process of federating the network, storage, and computing resources across different institutions and making them available via well-defined interfaces and protocols which are exposed by grid middlewares to various scientific communities (Laure and Edlund, 2011). This approach therefore ensures that organisations do not need to have their computing infrastructures situated at one location and can therefore perform their computation on scalable and heterogeneous systems and across multiple administrative domains. There are several general purpose Application Programming Interfaces (API) for distributed resources which are used to provide levels of abstractions over the different middleware implementations. Such APIs include Simple API for Grid Application (SAGA) which is used to provide a level of abstraction over the various middleware implementations by hiding resource access and complexities and enabling interoperability between the different tools and applications (Merzky, Weidner and Jha, 2015). A similar tool to SAGA which can also be used to provide levels of abstraction over the different middleware implementation is the Java Commodity Grid Kit (Java CoG Kit). According to von Laszewski et al. (2001), the Commodity Grid projects is working to bridge the gap between the different commodity distributed computing technologies and frameworks by creating what is known as the Commodity Grid Toolkits (CoG Kits) which provide mappings and interfaces between grid and specific commodity framework.

The use of distributed computing resources to perform experiments can help (potentially) in allowing authorised scientists to access and execute scientific applications and subsequently obtain results in a timely manner. However, the use of complex programming languages can cause the deployment and use of these resources to be complicated and could be a daunting experience in which users may have to cope with complex and sophisticated technologies such as job description languages, execution scripts, command line interface and management of personal digital certificates. These requirements are quite cumbersome to users (especially non-ICT expert users) and could make the learning curve to be very steep. These access requirements may consequently prevent potential users from adopting the technology.

To enable easy access and use of these resources (as well as improve the availability of scientific software applications), the concept of Science Gateway has emerged as an alternative solution. A Science Gateway is defined as a digital interface to advanced technologies which is used to provide adequate support to science and engineering research and education (Lawrence et al., 2015). They are used to provide access to community resources including software, sensors, data, instrumentation and high-performance computing. Science Gateways ensure that users do not have to worry about the complex programming languages and other technical details which are earlier mentioned. As such all the details are managed at a level where functions will take care of job execution and data management activities (on different DCIs) thereby exempting users from the details of the implementations of different middlewares.

The focus of this study therefore is to examine existing approaches for DCI access and to propose a new approach for developing scientific applications in Science Gateways. There are several portal frameworks in use which aid the rapid development of portlets in Science Gateways. These frameworks include but not limited to Liferay, GridSphere, and Jetspeed. The need to have these frameworks stem from the pervasiveness of many application specific portals as well as the inability to reuse code in the presentation layer (Novotny, Russell and Wehrens, 2004). Science Gateways are normally built on top of these portal frameworks in order to aid in rapid portlet development.

Today, there are several Science Gateway frameworks in use that can efficiently help in developing application specific Science Gateways for different scientific domain. A Science Gateway framework is a platform which is used to offer a set of high-level grid and cloud services in which interconnection between grids, clouds and scientific user communities is achieved. They are usually equipped with Application Programming Interfaces (APIs), or set of libraries to manage the use of the Science Gateways and the interaction with several different kinds of DCI (Grid, Cloud, Clusters, HPC, etc). Based on the review of literature, some of the commonly used Science Gateway frameworks include WS-PGRADE/gUSE Science Gateway framework, Catania Science Gateway Framework (CSGF), VineToolkit Science Gateway framework and InSilicoLab framework. Several instances of these Science Gateway frameworks have been implemented either as production or prototype gateway instances for the different communities of practice and thus help in saving valuable resources (such as development time and effort). These Science Gateway efforts are discussed in more detail in Chapter 2.

Furthermore, to develop scientific software applications in Science Gateways several aspects such as the functionalities and requirements of the scientific application needs to be considered. A case in point is to consider ways in which scientific software applications are developed in order to achieve some level of adequate performance gains. For instance, a simulation experiment consisting of a substantial population of agents may be slow to execute and the problem may be to find ways to keep execution time manageable. The use of distributed systems can therefore help to enhance system performance. In addition, it will be of immense benefit to investigate ways in which a Science Gateway may be used to execute multiple experiments in a distributed environments. As a consequence, investigating the different approaches to developing scientific software applications in Science Gateways and executing them on DCIs (for single and multiple experiments) for different communities of practice such as M&S is of great benefit.

## *1.3 Research Motivation*

Even though much effort has gone in ensuring that communities of practice and scientific domains can perform scientific experiments in an efficient manner and on a large scale or resources (using DCI resources), accessing such environments may present a different set of challenge most especially to scientists in different fields who are not necessarily ICT expert users. Consequently, the ease of access and use of such environments therefore become really important. However, the use of Science Gateways can help to improve accessibility of the DCIs. This research is therefore motivated by the advances made in the area of distributed computing resources and Science Gateways. Furthermore, several other distributed resources can also come in play such as making provision for storing associated research data/outputs as well as data obtained from the execution of scientific experiments for its easy find and access. Using distributed environment in this way then becomes really paramount not only for the ease of access and use of research software and applications but also for the accessibility and availability of other research elements such as research data and associated data obtained from the execution of scientific applications.

## *1.4 Research Aim and Objectives*

The aim of this research is to create a methodology for developing scientific software applications in Science Gateways. To achieve this aim, the following research questions will be addressed:

- (RQ1) What approach can be used to develop scientific software applications in Science Gateways?

- (RQ2) What Science Gateway framework and SDLC approach are appropriate for implementing the proposed methodology for developing scientific software applications in Science Gateways?

- (RQ3) Is the developed methodology effective?

Furthermore, in order to achieve the aim and subsequently address the aforementioned research questions, the following objectives will be met:

1. Conduct a literature review on the state of the art of Science Gateways, Web Portals, Workflow Management Systems, DCI resources and other technologies that lend themselves to the access and use of distributed systems.

2. Identify appropriate research methods for the entire research process as well as create a methodology for developing scientific software applications in Science Gateways.

3. Implement the proposed methodology for developing scientific software applications in Science Gateways in objective (2) using three different scientific software applications. The first scientific application is the case study used to develop and implement the ideas and methods described in the methodology, while the second scientific application is used for its evaluation.

4. Perform a thorough evaluation of both the proposed methodology in objective (2), and the development and implementation of the scientific software applications in objective (3), by comparing with the approach that was used in developing real life use-cases of two Science Gateway projects.

5. Based on the evaluation in objectives (4), revise the methodology for developing scientific software applications in Science Gateways which was proposed in objective (2).

## *1.5 Audience and Scope*

Science Gateways are typically used to realise the computing needs of different communities of practice. As such, to realise a fully fuctional Science Gateway, several level of expertise (for the creation, operation and usage of the Science Gateway) are required. The first category of people that will benefit from reading this thesis are therefore the end-users (or scientists) who may need to utilise the Science Gateway interface to execute their jobs and thus require a basic understanding of the use of Science Gateways. The second category to benefit are the Science Gateway developers. This category is further divided into Science Gateway framework developers and Science Gateway instance developers. Both set of developers will benefit immensly from the MESSAGE methodology presented in this thesis. The final category to benefit are the Science Gateway operators who are tasked with the deployment, configuration, running and maintenance of the Science Gateway services for different user communities.

The focus of this study is to examine existing approaches for DCI access and to propose a new approach for developing scientific applications in Science Gateways. Taking into consideration the time available, attempt will be made to apply this new approach to majority of the Science Gateway services identified in Chapter 2. The proposed methodology is a generic framework which is applicable to different scientific applications and utilised across different Science Gateway frameworks. However, due to time constraints, only one Science Gateway framework is utilised. In addition, only two of the Science Gateway services is implemented. Furthermore, the execution of jobs via the Science Gateway interface is limited only in the use of one DCI (i.e the cloud). Lastly, this thesis is only beneficial to end-users, Science Gateway developers and Science Gateway operators as earlier mentioned.

## *1.6 Research Methodology*

In this section, a brief introduction to the research methodology that is adopted in this work is explained and justified. According to Johnson (2006), if the final result of an experiment does not support a hypothesis, it may be due to some flaws in the overall method being used. As such, it is imperative that an appropriate research strategy is adopted. In addition, well-formed and meaningful research questions will act as a guide to the entire research. The aim of this research is to create a methodology for developing scientific software applications in Science Gateways. To achieve this aim therefore, the Research Questions (RQs) mentioned in Section 1.4 will be addressed.

According to Yin (2009), a case study is "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially where the boundaries between the phenomenon and context are not clearly evident". The case study methodology is suited to different kinds of software engineering research as the object of study are often contemporary and they help to provide deeper understanding of the phenomena under study (Runeson and Höst, 2009). This type of research method follows four (4) distinct steps such as Hypothesis generation, Method identification, Result compilation and Conclusion. While the ideas to be tested by the research are explicitly identified at hypothesis generation stage, the method identification stage clearly defines the methods that is used to establish the hypothesis. Also, at the result compilation stage, the results of the experiment based on the adopted method is presented. Lastly, conclusions (to support or reject the hypothesis) are drawn based on the results. Consequently, the four major steps in this research includes:

1. Conduct a thorough literature review, identify gaps and form the hypothesis.
2. Propose a research methodology that will help to evaluate and establish the hypothesis.
3. Formulate case studies that can be used to experimentally test/evaluate the methodology.
4. Present results of evaluations and draw conclusion.

The above approach is well suited to this study as it can help to progressively establish a hypothesis of this nature (i.e. it is feasible to create a methodology for developing scientific software applications in Science Gateways). Furthermore, the development of scientific software applications in Science Gateways (such as an agent-based simulation application) is the realisation of software-as-a-service for specific scientific domains. Therefore, the overall process will also comprise of the development of a software artefact.

Design research is used to create such new software artefacts and according to Vaishnavi and Kuechler (2009), the developed artefacts should upgrade on the existing practices and deliver useful and efficient services. The different stages in the Design Science Research (DSR) methodology include awareness of problem, suggestion, development, evaluation and conclusion. This is also similar to the four major steps which are required in DSR as discussed by Bilandzic and Venable (2011). These steps includes problem identification, design, implementation, and evaluation. This research approach is associated with human-made artefacts (in terms of construction and evaluation) in order to enhance systems element (Myers, 1997).

The different stages of the Design Research also complement some of the steps outlined by Johnson (2006) in the empirical research method. Therefore, a similar approach is used in developing the Science Gateway artefacts. Consequently the DSR methodology (which is discussed in great detail in Chapter 3) is adopted in this thesis.

## *1.7 Thesis Structure and Outline*

This thesis is divided into seven Chapters as follows:

**Chapter 1** presents the introduction to the thesis. It discussed the research context, research motivation and rationale of the research. Furthermore, it identifies the aim and objectives of the research and presents the research methodology that is adopted throughout the thesis to fulfil those objectives and meet the aim. Finally, an overview of the entire thesis is presented.

**Chapter 2** reviews the different technologies that lend themselves to the access and use of distributed infrastructures such as Science Gateways, Web Portals and Workflow Management Systems. It also surveyed the different DCI resources (such as cloud, grid and clusters) and the back-end which provide access to these resources. Moreover, the various projects which have been undertaken in the aspect of Science Gateways and Scientific Workflows are discussed. Furthermore, based on the different projects that were discussed, several requirements and high level functionalities of the systems emerged. These high level functionalities were therefore presented in Chapter 4 as the essential services of Science Gateways.

**Chapter 3** establishes the research methodology that is adopted in this study. It starts by discussing the different research perspectives and assumptions and how they relate to the study at hand. It also established the nature of the research by matching the research questions with the research objectives which consequently resulted in three distinct research types. A careful study reveals that a research methodology which is capable of incorporating all three identified research types is the DSR. Furthermore, the approach that is used to achieve the different research types is presented.

**Chapter 4** identifies the essential services of generic Science Gateways and creates a MESSAGE methodology for developing these services (for different scientific applications) in Science Gateways. The review of literature presents the high level functionalities required by different communities of practice. These high-level functionalities were referred to as the essential Science Gateway services. This therefore provides a framework of Science Gateway services which is discussed in detail and a methodology that may be used for developing scientific software applications in Science Gateways (otherwise known as MESSAGE methodology) is later presented.

**Chapter 5** discusses the development/implementation of three different case studies (the Infection Model portlet, WEKA - J48 portlet and the visualiser portlet) based on the MESSAGE methodology that was proposed in Chapter 4. It presents the sequential version of the scientific applications, i.e. portlets that could be used to execute single experiment in a distributed environment. It starts by giving a general description of the use-cases, their requirements, functionalities, core components and the adopted Science Gateway approach. Lastly, it discusses the design and implementation of both applications by utilising relevant aspects of a software development methodology such as the waterfall model.

In **Chapter 6**, the analysis and design of the parallel approach to job execution are discussed. This Chapter presents the parallel versions of both portlets, i.e. portlets that could be used to execute multiple experiments within a distributed environment. It introduces the different methods which the CSGF can use to execute jobs in parallel and discusses in great detail the design and implementation of this portlet and how it may be used to execute multiple jobs in distributed environments.

**Chapter 7** presents the evaluation of the proposed MESSAGE methodology which subsequently supports the aim of the thesis. Different approaches are utilised for the assessment of both portlets. The first approach involves comparing the method used in developing both the Infection Model portlet and the WEKA – J48 portlet. Secondly, the method used for developing both portlets was compared with several scientific software applications developed in two different Science Gateway projects (i.e. the Sci-GaiA project and the CloudSME project). Consequently, a revised MESSAGE methodology for developing scientific software applications in Science Gateways was proposed based on the evaluation that was performed throughout the Chapter. Furthermore, it considers the possibility that (more than the easy access to scientific software applications) communities of practice may also require access to other research outputs which are generated as a result of executing these scientific applications. This Chapter therefore concludes with the use of the Infection Model simulation output results to demonstrate the access and retrieval of simulation output results, using enabling technologies such as open access repositories, etc., and thus providing a basis for Open Science Research.

In **Chapter 8**, the summary and the conclusion of the thesis are discussed. A reflection is made on how the different objectives have been met in order to achieve the research aim in Chapter 1. More importantly, the contributions made throughout the research are highlighted. Finally, this Chapter analyse the limitation of the research and presented the future work to be undertaken.

## *1.8 Summary*

This research is motivated by the advances made in the aspects of DCIs and Science Gateways. Several scientific communities now make use of DCIs such as grids and clouds to enable easy accessibility and availability of their scientific applications and to execute jobs in more efficient ways. Science Gateways are used to provide user friendly interfaces that hide all the details of the underlying infrastructure and exposes only the science-specific parts of the applications to be executed in the various DCIs. This Chapter therefore discusses the context and the rationale which has motivated this research. It also establishes the aim of this research which is to create a methodology for developing scientific software applications in Science Gateways. To achieve this aim, it presents three research questions and five research objectives. In addition, the research audience such as the end-users, Science Gateway developers and Science Gateway operators as well as the scope of the research were discussed. The research methodology that is adopted in order to achieve the objectives is also briefly discussed. Lastly, it analyses the structure of the thesis with a brief description of each Chapter.

# Chapter 2: REVIEW OF LITERATURE

## *2.1 Overview*

The aim of this research is to create a methodology for developing scientific software applications in Science Gateways. This Chapter therefore provides the context of this aim through a discussion of Science Gateways and its enabling technologies. It starts with the commonly used distributed resources and their role in the support and execution of scientific applications. Furthermore, it analyses the more generic Science Gateway frameworks and how they are used to create application specific Science Gateways. In addition, it discusses the different Scientific Workflow Management System (SWMS) in use and their role in the execution of jobs in a distributed environment. This Chapter concludes with a summary of the important features and functionalities that Science Gateways should possess as discussed throughout the literature.

The increasing need to analyse huge amount of data, run large-scale simulations that require large amount of Central Processing Unit (CPU) and the high demand for computational power all points to a shift from a local or single system to a more geographically dispersed set up. Consequently, Section 2.2 discusses an overview of the distributed resources and places emphasis on the use of Grid and Cloud for the execution of scientific jobs. Section 2.3 presents the application programming interfaces which provide the necessary abstraction to the different middleware implementations and the underlying infrastructures. To create Science Gateways for different user communities, a developer either has to write the Science Gateway from scratch by making use of available portal frameworks such as Liferay or customise an existing Science Gateway framework according to the specific needs of each community. Consequently, Section 2.4 and Section 2.5 therefore discuss some portal frameworks in use as well as the Science Gateway frameworks and instances, respectively. Furthermore, Section 2.5 also reviews the early technologies and projects (such as the Open Grid Collaboration Environment (OGCE) and Grid Portal Development Toolkit (GPDK) which are used to provide access to distributed systems. More than just simple job submissions and service calls some applications solving complex problems such as scientific simulations require the creation and execution of scientific workflows. Therefore, Section 2.6 briefly describes the common types of SWMS which represents another technique being employed for the execution of scientific applications in distributed environments. Section 2.7 captures the attributes and functionalities common to all the technologies that help provide a transparent access to DCI resources as discussed throughout this Chapter. Lastly, Section 2.8 introduces the concept of open research and the major initiatives that support Open Science. A conclusion of the entire Chapter was done in Section 2.9.

## *2.2 An Overview of Distributed Computing Infrastructures (DCI)*

The need for more efficient approach to execute jobs and also to provide worldwide access to scientific software applications suggest a move from centralised computing to a more distributed environment. The management of the underlying physical resources of several distributed systems is therefore really critical. Over the years, many computing models have evolved to help with task execution and job management. The following section therefore provides a summary of the commonly used computing models.

Advances in networking and distributed computing techniques have given rise to the concept of Virtual Organisations (VO) where different organisations combine researches and resources across traditional administrative and organisational domains (Laure and Edlund, 2011). This description is closely related to the concept of the Grid which was defined by Foster, Kesselman and Tuecke (2001) as "the coordinated resource sharing and problem-solving in dynamic multi-institutional virtual organisations." Distributed environments are inevitable as the world is nearly fully connected. Also, simple computing tasks on small data sets where everything is local are a common place, however, as soon as more resources are required, the principles of distributed computing become paramount. Furthermore, network, storage and computing resources are made available across a big number of institutions to different scientific communities via well-defined protocols and interfaces and are exposed by software layers known as Grid middleware. The most popular and widely used Grid middleware includes globus toolkit, gLite, and unicore. Other commonly used Grid middleware also includes Legion and Condor.

Cloud computing (which is a more recent computing model) is usually provisioned over the internet where services are delivered on demand to customers. It has several deployment models (public, private, community, and hybrid) as well as service models (software-as-a-service, platform-as-a-service, and infrastructure-as-a-service). These service models are often overlaid above the hardware and network infrastructures and made available to end users through a web interface or a language interfacing Application Programming Interface (API).

Other equally important and widely used distributed computing infrastructures include Cluster computing and Supercomputing. However, while Cluster computing and Supercomputing are more focused on traditional non-service applications, Grid computing (on the other hand) overlaps with other computing concepts (as shown in Figure 2.1) and is considered of lesser scale than Super computers and the Cloud (Foster et al., 2008). The Science Gateway layer in Figure 2.1 shows an alternative user friendly interface that can be used to access all the different DCIs, which will be the main focus of this research.

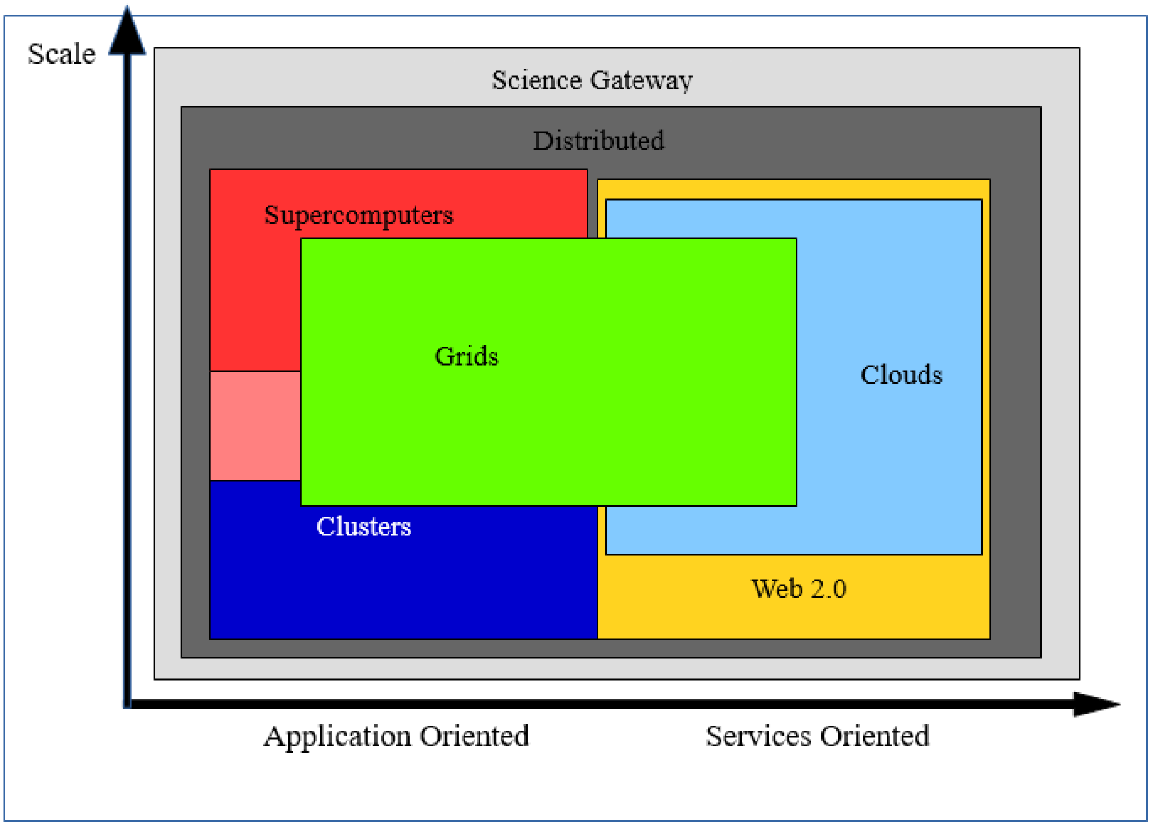
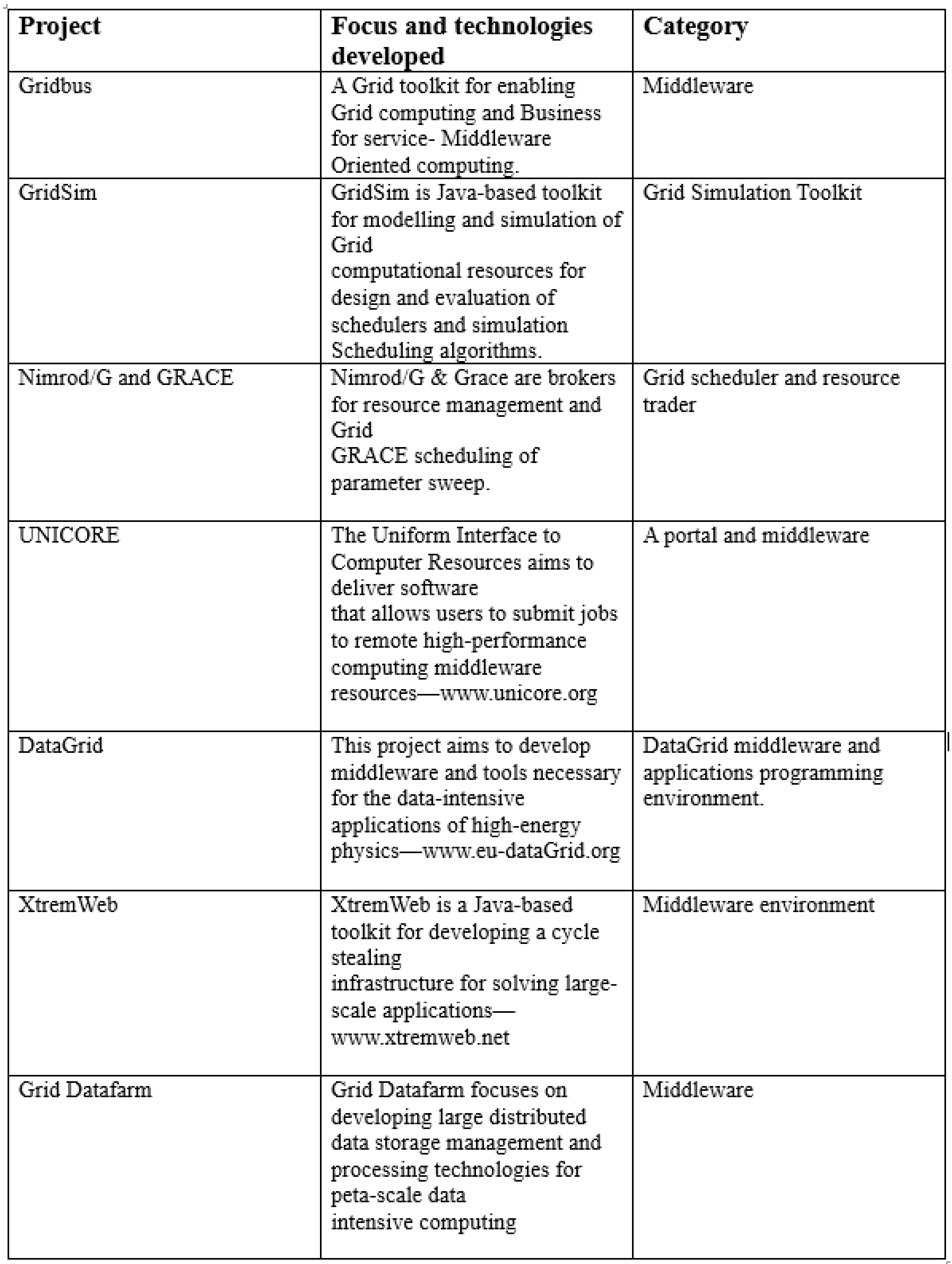


Figure 2. 1 A Section of the Distributed Systems and the Science Gateway Interface Layer. Adapted from (Foster *et al.*, 2008)

### 2.2.1 Grid Computing

Grid computing is a flexible, secure and coordinated resource sharing among dynamic collections of individuals, institutions and resources which are referred to as virtual organisations (Foster, Kesselman and Tuecke, 2001). Baker, Buyya and Laforenza (2002) discussed the state-of-the-art of grid computing, the different international efforts that have been made and the technologies used in achieving this model. Grids are used to provide services such as computational services, data services, application services, information services and knowledge services. There are different aspects that characterise and idealise the grid design features which are used to provide potential users with a seamless computing environment. These include multiple administrative domains and autonomy, heterogeneity, scalability and dynamicity/ adaptability. To realise a Grid, the integration of participating software and hardware, the deployment of low-level and user-level middleware as well as the development and optimisation of distributed application are considered to be key. In addition, other components which are required for the formation of the Grid as discussed by the authors include Grid fabric, core Grid middleware, user-level Grid middleware and Grid applications and portals. Several Grid efforts that were developed (internationally) were highlighted and thoroughly discussed by the authors. Some of these projects were also highlighted in the work of Sadashiv and Kumar (2011). Among these projects are globus (a type of Grid middleware that provides a set of API to the underlying services and resources), legion (which provides users with a single and coherent virtual machine), Nimrod-G and GRACE, GridSim, Gridbus, UNICORE, Information Power Grid and so on. Some of these projects are therefore summarised in Table 2.1 below.

Table 2.1 Grid Projects (Adapted from Baker, Buyya and Laforenza 2002)



| **Project** | **Focus and technologies developed** | **Category** |
|---|---|---|
| Gridbus | A Grid toolkit for enabling Grid computing and Business for service- Middleware Oriented computing. | Middleware |
| GridSim | GridSim is Java-based toolkit for modelling and simulation of Grid computational resources for design and evaluation of schedulers and simulation Scheduling algorithms. | Grid Simulation Toolkit |
| Nimrod/G and GRACE | Nimrod/G & Grace are brokers for resource management and Grid GRACE scheduling of parameter sweep. | Grid scheduler and resource trader |
| UNICORE | The Uniform Interface to Computer Resources aims to deliver software that allows users to submit jobs to remote high-performance computing middleware resources—www.unicore.org | A portal and middleware |
| DataGrid | This project aims to develop middleware and tools necessary for the data-intensive applications of high-energy physics—www.eu-dataGrid.org | DataGrid middleware and applications programming environment. |
| XtremWeb | XtremWeb is a Java-based toolkit for developing a cycle stealing infrastructure for solving large-scale applications—www.xtremweb.net | Middleware environment |
| Grid Datafarm | Grid Datafarm focuses on developing large distributed data storage management and processing technologies for peta-scale data intensive computing | Middleware |

## Grid Characteristics

According to (Bote-Lorenzo, Dimitriadis and Gómez-Sánchez, 2004) the origin of Grid computing concept could be traced to the early 90's when efforts were first made to link several supercomputing sites across the USA, by simply deploying gigabit testbeds. The term "Grid" was coined in the mid-90's. It was derived from the notion of the electric power grid to denote a proposed distributed computing infrastructure for advanced science and engineering. This view of the Grid ensures that computing becomes pervasive and users can therefore have access to computing resources such as storage, processor, data and applications. More so, access is made with little or no knowledge of where resources are located or what the underlying technologies, hardware and operating systems are.

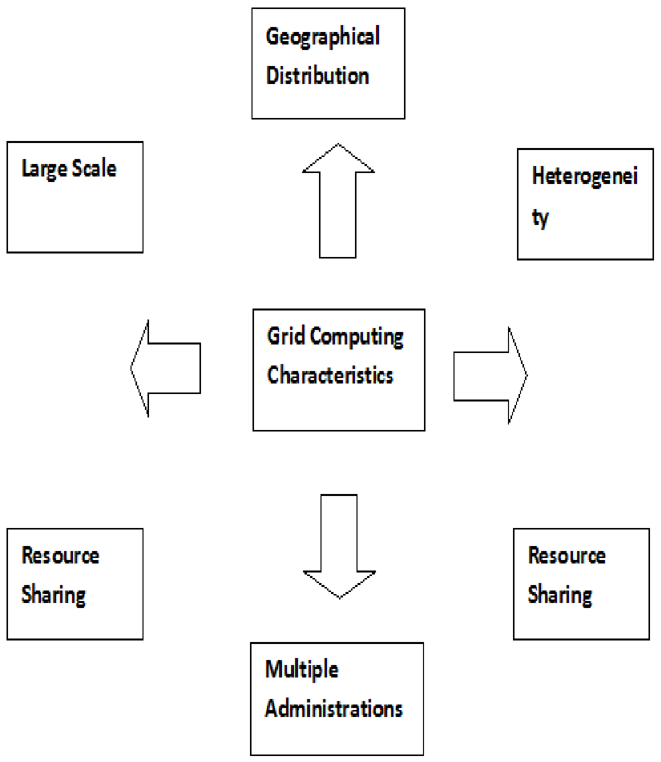


Figure 2. 2 Grid Computing Characteristics

Grid computing is defined as an environment that provides the ability to share and transparently access resources across a distributed and heterogeneous environment (Jacob *et al.*, 2005). The major problem that underlines the Grid concept is the coordinated resource sharing (i.e. direct access to computers, software and data) and problem-solving in dynamic, multi-institutional virtual organisations. Sharing in this way is highly controlled as resource providers and consumers define very clearly all the terms associated with it. These terms could include: what is shared, those allowed to share and the conditions under which sharing may take place. A set of institutions defined by such sharing rules are collectively known as a Virtual Organisation (VO). These VOs may differ tremendously in scope, purpose, structure and size. Classical examples of VO include the application service providers and storage service providers. In light of the above definition of Grid computing, (Bote-Lorenzo, Dimitriadis and Gómez-Sánchez, 2004) and (Baker, Buyya and Laforenza, 2002) summarised the Grid characteristics (as shown in Figure 2.2) as large scale, geographical distribution, heterogeneity, resource sharing, multiple administrations, resource coordination and (transparent/dependable/consistent/pervasive) access.

## Grid Computing Layers

VO was earlier defined as same set of institutions which are collectively bound by the same sharing rules. To support the creation of these VOs, a logical entity in which distributed resources are discovered and shared as if they are from the same organisation must be provisioned. In Grid computing, a set of standard protocols, middleware, toolkits and services are defined. Grids enable protocols and services at different layers as shown in the Grid protocol architecture in Figure 2.3.

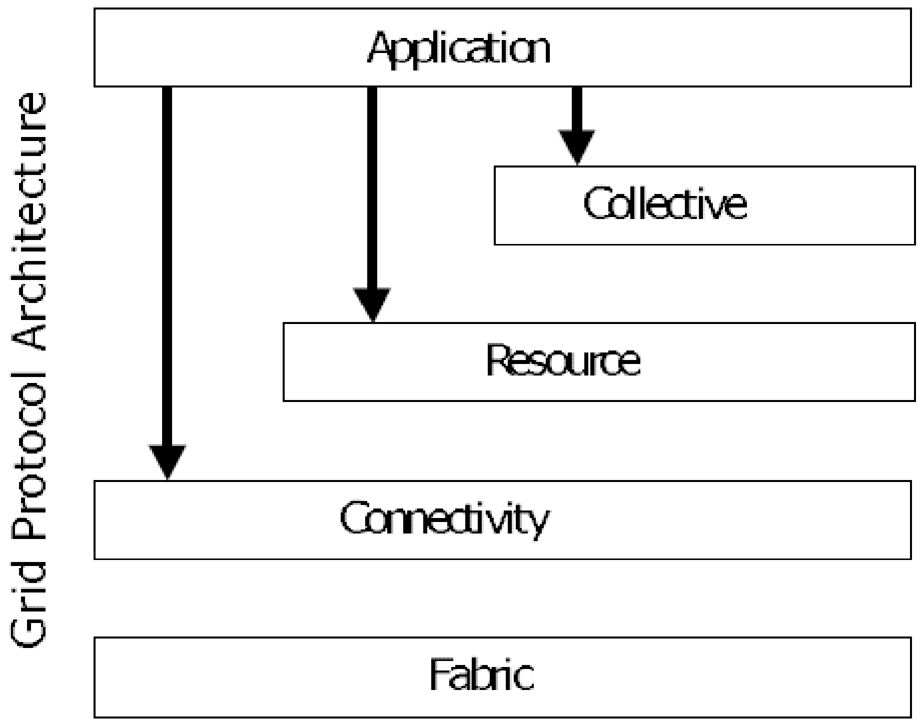


Figure 2. 3 Grid Computing Layers. Adapted from (Foster *et al.*, 2008)

**The Fabric Layer** is the interface to local control that enables the grid to facilitate and provide access to different resources such as computational resources, storage resources, network resources, catalogs and sensors. The fabric components are responsible for implementing the local and resource-specific operations on specific resources which may occur as a result of the sharing that happens at higher levels of operations.
**The Connectivity Layer** enables easy and secure communication and defines the core communication and authentication protocols which are required for grid-specific network transactions. Data exchange between Fabric layer resources is done via the communication protocols and authentication protocols that are built on their services (i.e. communication services) in order to provide cryptographically secured mechanisms for verifying the identity of users and resources (Foster, Kesselman and Tuecke, 2001).
**The Resource Layer** enables the sharing of single resources, and it defines protocols for the secure initiation, publication, discovery, negotiation, monitoring, accounting and control of sharing operations on individual resources. This layer is built over the Connectivity layer and is concerned primarily with individual resources only. The Resource Layer is categorised into two main classes namely the Information protocols (used to obtain information about the structure and state of a resource) and the Management protocols (used to negotiate access to shared resources).
**Collective Layer** enables the coordination of multiple resources as opposed to the single resource of the Resource Layer. It is built on top of the Resource Layer and the Connectivity layer, and it captures interactions across collections of resources. It implements a wide variety of Collective layer protocols and services (sharing behaviours) such as directory services, collaboratory services, monitoring and diagnostic services, etc.
**The Application Layer** that comprises of user applications which were built on top of the different protocols and APIs for the aforementioned layers. This Layer consists of the user applications that operate within a VO environment. Applications are built based on the services defined at each layer. Each layer may consist of well-defined protocols that enable access to services such as resource discovery, resource management, data access, etc.

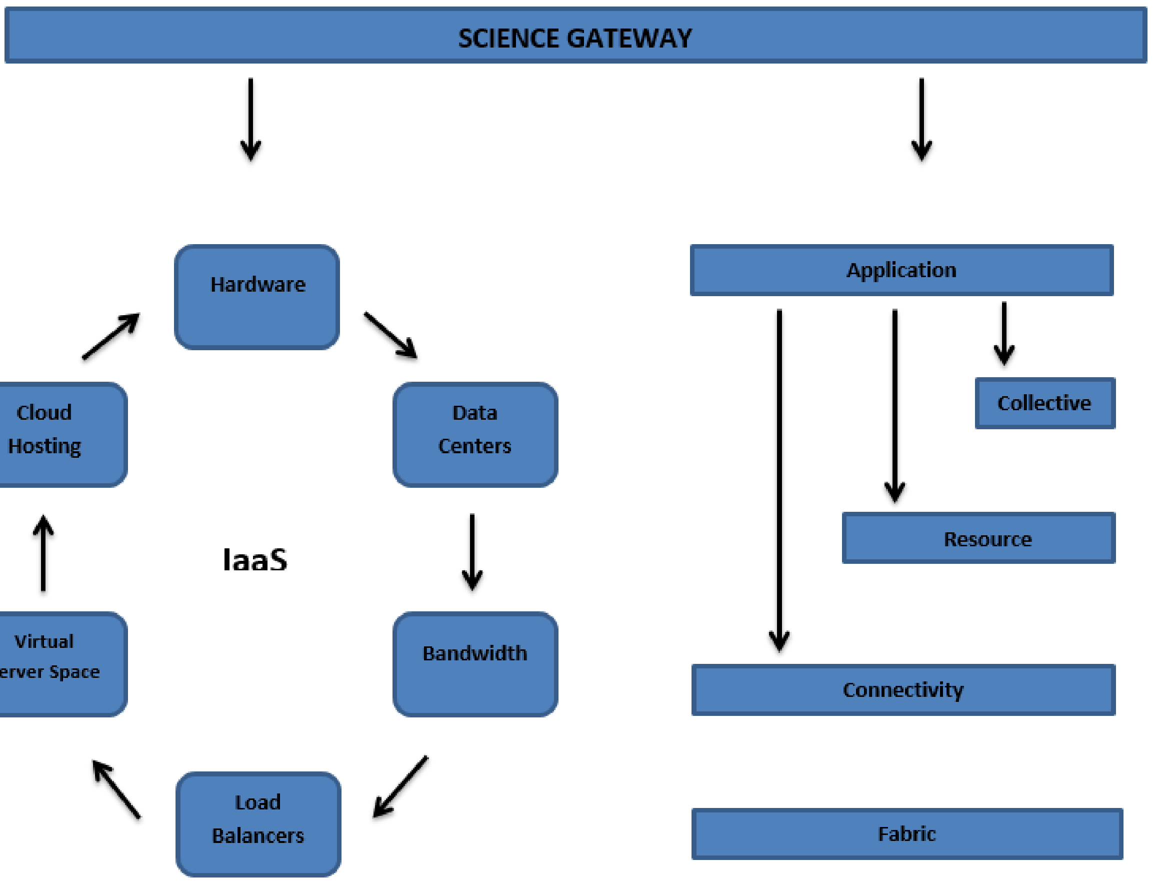


Figure 2.4 A view showing the Grid and Cloud elements with the Science Gateway Access Layer

### 2.2.2 Cloud Computing

The Cloud computing paradigm has been compared to utility services such as gas and electricity where such services are provided to the general public and made available individually yet simultaneously. Cloud computing hints at a future where computing is done not only on local computers but also on centralised facilities which are operated by third-party compute and storage facilities. According to the National Institute of Standards and Technology (NIST), Cloud computing is a model that is used to enable ubiquitous and on-demand network access to a shared pool of computing resources such as networks, servers, services, storage, and applications (Mell and Grance, 2011). They are usually provisioned and released rapidly with minimal management effort.

Foster et al (2008) defined Cloud computing as a large-scale distributed computing paradigm being driven by economies of scale whereby a host of “abstracted, virtualised, dynamically-scalable, managed computing power, storage, platforms and services are delivered on demand to customers over the internet”. Some key points have emerged from these definitions which show Cloud computing to be different from the traditional computing in that it is a specialised distributed computing paradigm that is massively scalable. In addition, it can serve as an abstract entity which is used to deliver different levels of services to customers. Lastly, its services are dynamically configured and delivered (on demand) and it is driven by economies of scale. The term "Cloud computing" became popular in 2006 after Eric Schmidt (the then CEO of Google) used it to describe the business model of providing services over the internet.

Cloud computing is similar to the earlier computing paradigm such as Grid in that it is used to increase reliability, reduce the cost of computing, and increase flexibility whereby computers are transformed from something that is bought and operated locally to one that is exploited by third parties. According to Zhang, Cheng and Boutaba (2010), Cloud computing is not a new technology but a rather new operations model which combine a set of existing technologies to execute business in a different manner. Cloud computing provides several features such as Zero up-front investment, low operational cost, highly scalable, easy access, and reducing business risks and maintenance expenses. According to the National Institute of Standards and Technology (NIST), there are some essential characteristics that a Cloud model should possess. Furthermore, Cloud computing enables different service models as well as deployment models which are briefly explained in the following sections.

## Cloud Computing Characteristics

There are five (5) essential characteristics of a Cloud model (NIST, 2011). These are on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service as shown in Figure 2.5 and are briefly described in the following section:

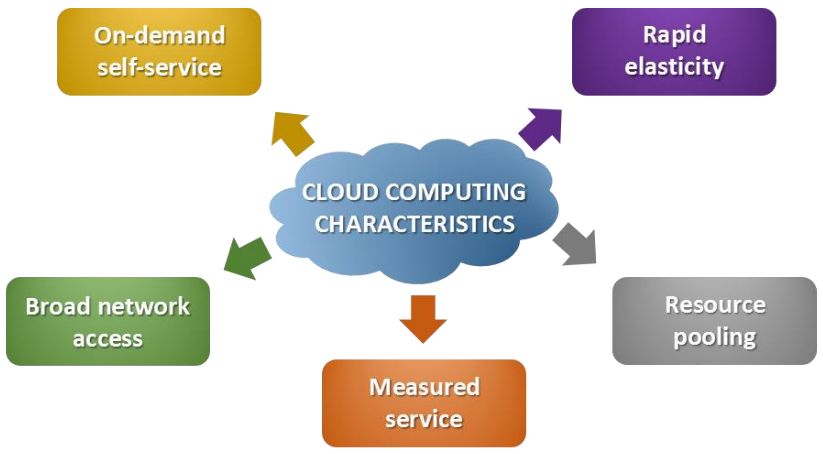


Figure 2. 5 Cloud Computing Characteristics

***On-demand self-service****:* A consumer can provision computing capabilities in a unilateral manner. These could involve the automatic provisioning of server time and network storage without the need for the user to interact with each service provider.

***Broad network access***: This refers to resources that are hosted in a private Cloud network which are usually operated within a company's firewall and are available for access from a wide range of devices. Client platforms such as mobile phones, laptops, tablets and workstations are utilised over the network. The Cloud computing features enable broad network access using standardised interfaces that operate with the aforementioned client devices.

***Resource pooling*** ensures the use of the same physical resources to service multiple customers by securely separating the resources at the logical level. As such, virtualisation techniques are used to pool cloud resources based on user requests. In light of this, different physical and virtual resources are dynamically assigned and reassigned according to consumer demand.

***Rapid elasticity*** enables the elastic provisioning and release of capabilities. The level of resource allocation may depend on the current need and it facilitates the changes to each allocation to be effected in an efficient manner.

***Measured Service***: This service ensures that cloud systems have an effective and automatic control and management of operational activities. It leverages on a metering capability at some level of abstraction that is suited to the type of service. Monitoring, reporting and resource usage are done transparently based on utilisation therefore users are charged based on the number of computing hours being used.

## Cloud Computing Service Models

In addition to the aforementioned five essential characteristics of a Cloud model, there are several service models which are used by Cloud communities to categorise Cloud services as shown in Figure 2.6. Above the hardware and network infrastructure, there are several end-user service models such as Infrastructure as a service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These services are available to end users via a web interface or language-interfacing Application Programming Interface (API) (Buyya *et al.*, 2009)

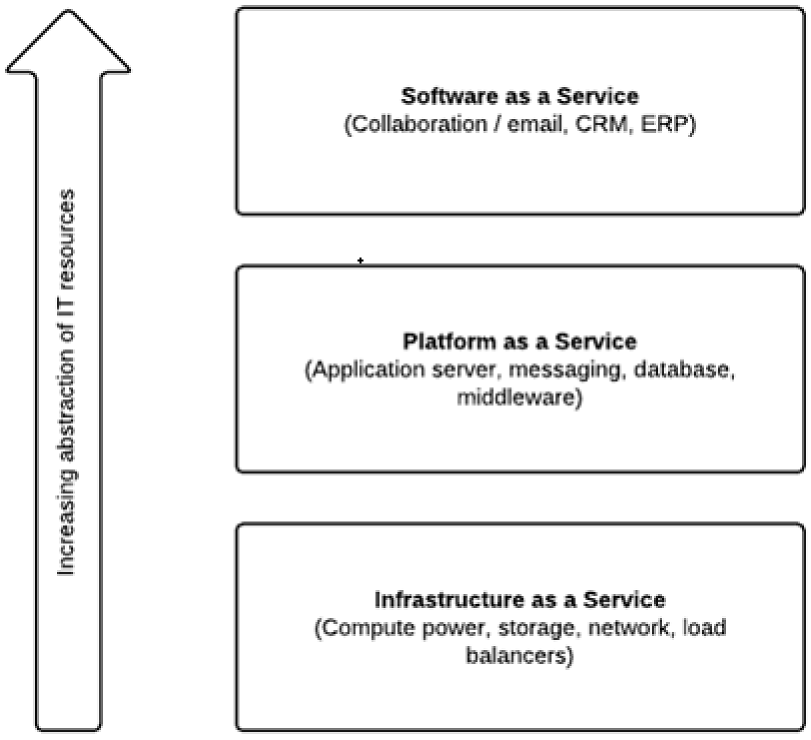

Increasing abstraction of IT resources
Software as a Service
(Collaboration / email, CRM, ERP)
Platform as a Service
(Application server, messaging, database, middleware)
Infrastructure as a Service
(Compute power, storage, network, load balancers)

Figure 2. 6 Cloud Computing Service Models

***SaaS*** enables Cloud consumers to release applications which are usually accessed through networks from various clients (e.g. PDA, web browser) in a hosting environment, therefore users do not need to install and run software on their devices. The overall management of computing activities such as applications, infrastructure and operating systems is the responsibility of the cloud provider. Consequently, the cloud consumers do not have control over the cloud infrastructure. Examples of Saas include Google Docs, SalesForce.com, Google Mail, etc.

***PaaS*** facilitates the development of cloud services and applications directly on the PaaS cloud by cloud consumers. It therefore provides a development platform which hosts both completed applications (similar to SaaS) as well as in-progress cloud applications. As a result, this layer consists of tools and programming languages and APIs for building and delivering web applications and services. This could entail the entire software lifecycle such as design, development, testing, deployment, hosting and maintenance. Examples of PaaS include Google AppEngine, Microsoft Azure, AppFog, etc.

***IaaS*** provisions the processing, storage, networks and other fundamental computing resources where users can deploy and execute arbitrary software. The aforementioned ICT infrastructures which are provided in the IaaS cloud are directly utilised by cloud consumers. In terms of flexibility, this model is more flexible than the PaaS and the SaaS as it gives customers higher control. Even though the cloud consumer does not manage or control the underlying cloud infrastructure, they can exercise control over the operating systems, storage, application deployment, etc. An example of IaaS is the Amazon's EC2.

## Cloud Computing Deployment Models

Cloud computing also has four deployment models, in addition to the essential characteristics and the service models that were discussed above. These models are categorised on the basis of the owner of the infrastructure, the management team and the location or the end user (Mell and Grance, 2011). These four main models (as shown in Figure 2.7) and discussed by the authors, as well as the NIST, are summarised below as follows:

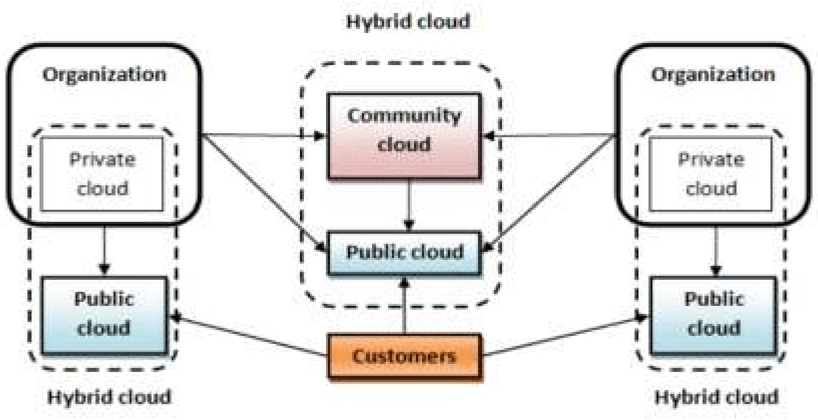


Figure 2. 7 Cloud Computing Deployment Model

**Private cloud** is operated and managed solely by a single organisation or by a third party. They are provisioned to be used exclusively by a single organisation. They are usually set up to maximise and optimise the utilisation of existing in-house resources. Furthermore, organisations that are more concerned with data privacy and trust often tend to adopt this kind of deployment model. It may either exist on or off the premises.
**Public cloud** infrastructure is provisioned for open use by the general public. This is the most common type of Cloud computing deployment model and the cloud service provider has the full ownership of its policy and charging cost. Examples of public Cloud services include AppEngine, EC2, and S3 and they exist on the premises of the Cloud provider.
**Community cloud** consists of several organisations which share the same Cloud infrastructure. This could also include organisations that share the same policies, requirements, concerns and values. It may be owned and operated by one or more organisations in the community or by a third party ownership. It may also either exist on or off the premises.
**Hybrid cloud** consists of a combination of two or more of the aforementioned Cloud deployment models, above. They are bound together by standardised or proprietary technology that enables data and application portability. To optimise resources and thus increase core competencies, organisations make use of the hybrid Cloud model. In this way, core activities are controlled on-premise by using private cloud, while the more peripheral business functions are marginalised onto other Clouds.

The above sections have discussed the most prominent distributed computing technologies that are used to execute scientific tasks. In particular, Cloud computing is the technology that is used to perform the execution of all the scientific application tasks in this research. Other variations of distributed systems include Cluster computing and HPC systems. According to (Limet, Smari and Spalazzi, 2015), HPC systems were introduced to solve computationally intensive, technical and scientific problems. In addition, several application domains now make use of HPC systems to execute complex calculations that require the analysis of huge amount of data in their terabyte and petabyte range. The historical developments of this system can be traced to Moore’s Law which states that “the number of transistors on a microprocessor chip would double every two years or so”.

### 2.2.3 Comparative study of Cloud and Grid Computing

In this section, a comparison of the two prominent distributed computing models (such as Cloud and Grid computing) is discussed. This comparison is based on the challenges of each computing model, the applications they use and some of the tools that are utilised for the design and development of such applications.

(Sadashiv and Kumar, 2011) defined Cloud computing as the application that is delivered over the internet in the form of services as well as the corresponding software and hardware used in providing those services. The challenges which may be experienced in Cloud computing include dynamic scalability, multi-tenancy, querying and access, standardisation, reliability/fault-tolerance, debugging/profiling, security/privacy, and power. Some of the initiative projects that have been developed in this model include CERN, unified Cloud Interface, Tclouds, OpenNebula, etc. In addition, various tools and products such as Zenoss, Spring Roo, CloudSim/CloudAnalyst and Cloudera are used to aid in the development of applications in the Cloud.

The authors defined Grid computing as the combination of different computers (from multiple administration domains) in order to achieve a common goal or solve a particular problem. Dynamicity, administration, development, accounting, heterogeneity, and programming are some of the challenges that were discussed which are common to Grid computing. Some of the applications and projects within Grid computing include advanced manufacturing, oil reservoir simulation, particle physics research, globus, EGI-InSPIRE, NASA Information Power Grid, etc. Also, tools listed by the authors for the implementation of Grid computing include paradyn, nimrod-G, condor-G, globus, gridbus, legion and gridsim.

In addition to the above, the authors also defined Cluster computing as a collection of parallel or distributed computers that are connected together using high-speed networks. They are used as an alternative to supercomputers due to their ability to grant relatively cheaper access to huge computational power. The challenges facing cluster computing include the type of middleware used, its program, elasticity and scalability. Also, several projects such as the Weather Research and Forecast (WRF), hadoop, nuclear simulation, etc., are said to be implemented on this computing model. Some of the tools which are used to implement cluster computing include nimrod, PARMON, condor, MPI/OpenMP, etc.

### 2.2.4 Blockchain-Based DCI

In addition to the above DCIs, there are other recent developments such as the emergence of blockchain-based DCls such as Ethereum. Ethereum is a decentralized virtual machine which runs programs called smart contracts upon user request. Contracts are written in a Turing-complete bytecode language, called Ethereum Virtual Machine (EVM) bytecode (Atzei, Bartoletti and Cimoli, 2017). In EVM, apart from several global parameters, most states are stored in accounts. EVM has a partial map from addresses (160-bit words) to account states. An account state contains code, storage, nonce, balance and the code is a sequence of bytes. The storage is a mapping from a machine word (an EVM machine word has 256 bits) to a machine word. A deployed Ethereum smart contract is public under adversarial scrutiny, and the code is not updatable. Users send transactions to the Ethereum network in order to perform activities such as create new contracts, invoke functions of a contract and transfer ether to contracts or to other users. All the transactions are recorded on a public, append-only data structure, called blockchain. The sequence of transactions on the blockchain determines the state of each contract, and the balance of each user. In EVM, apart from several global parameters, most states are stored in accounts. EVM has a partial map from addresses (160-bit words) to account states (Hirai, 2017).

## *2.3 Application Programming Interfaces (APIs) for Distributed Computing Infrastructures (DCIs)*

In the above section, the different distributed computing paradigms which are used for the execution of scientific applications are discussed. The following section will present the different Application Programming Interfaces (API) which are used to provide a level of abstraction over the different middleware implementations and their associated distributed resources.

### 2.3.1 Simple API for Grid Application (SAGA)

Simple API for Grid Application (SAGA) as described by Merzky, Weidner and Jha (2015) is a general purpose API for distributed resources used to provide a level of abstractions over the different middleware implementations by hiding resource access and complexities and enabling interoperability between the different tools and applications. The motivation for having SAGA stems from the heterogeneity of resources. Therefore, defining a simple and uniform access layer can ultimately provide support for diverse application requirements and usage scenarios over a shared set of resources. The design objectives of SAGA include managing system heterogeneity, support for common pattern usage and ease of use of the available resources. The scope of SAGA includes job and data management, resource information queries and communication abstractions. Several implementations of SAGA exist in programming languages such as Java, C++ and Python.

### 2.3.2 RADICAL Cybertools

RADICAL Cybertools are a set of integrated, abstraction-based suite of well defined capabilities which are developed for scalable, interoperable and sustainable support of science on a range of HPC systems. It is the design, development and use of abstraction-driven and standards-based solutions to large-scale compute and data-intensive scientific problems (Gesing, Nabrzyski and Jha, 2014). It consists of two components namely RADICAL SAGA and RADICAL-Pilot. While RADICAL-SAGA is a lightweight interface that provides standards-based interoperable capabilities to the most commonly used functionalities required to develop distributed appliocations, tools and services, RADICAL-Pilot (on the other-hand) is a scalable and flexible system that supports application-level resource management (Merzky, Weidner and Jha 2015).

### 2.3.3 A Java Commodity Grid kit

Another tool that is used to provide a level of abstraction to the different middleware implementation is the Java Commodity grid kit (Java CoG Kit). According to (von Laszewski et al., 2001), the Commodity Grid projects is working to bridge the gap between the different commodity distributed computing technologies and frameworks by creating what is known as the Commodity Grid Toolkits (CoG Kits) which provides mappings and interfaces between Grid and particular commodity framework. The authors believed that Grid technologies (which provides advance network services and multi-institutional environment for applications that requires the coordinated use of multiple resources) and commodity distributed computing technologies (that enables the rapid creation of client-server application) can combine for the benefit of advanced scientific software applications. The authors focused primarily on the Java CoG Kit and the use of Java for Grid computing. The Java CoG Kit has been categorised based on increased functionalities of each component where subsequent category reuses the lower-level components to facilitate an iterative development of future components. The different components which are classified based on their roles include low-level GUI interface, low-level utility components, common low-level GUI components and application specific GUI components. This has helped in defining rich set of classes that aid the developers with access to grid services and a range of GUI elements.

### 2.3.4 DCI Bridge

WS-PGRADE is primarily a workflow management system. To extend existing resources for job submission for the benefit of communities that do not need a workflow management system the DCI Bridge (which is a component of the WS-PGRADE/gUSE system) can be used (Balasko, Farkas and Kacsuk, 2013). This can transparently help to provide a standard interface for job submission to different middlewares. The DCI Bridge is deployed in the SHIWA and SCI-BUS projects and can potentially connect with any gateway via another service known as the Basic Execution Service (BES) interface to provide access to different set of DCIs, such as Grid, Cloud and Clusters. The DCI Bridge eliminates the need to use different submitters for each DCI configuration. The BES interface is used to submit and manage end-user workflows. Normally, several DCIs implement different types of job submission protocols and a generic Science Gateway framework should be able to handle all the different kinds of protocols. WS-PGRADE/gUSE Science Gateway framework uses the DCI Bridge for this purpose.

## *2.4 Portal/Web Application Frameworks*

A portal framework ensures the rapid development of portlets by enabling a generic portlet repository where a large set of ready-to-use portlets are found. However, they do not provide backends that support DCI access but they help in builing Science Gateway instances from scratch. The two commonly used portal frameworks such as Gridsphere and Liferay along with other web-based content management services are briefly discussed in the following section.

### 2.4.1 GridSphere

According to Novotny, Russell and Wehrens (2004), GridSphere portal framework was developed to improve on the lessons learnt from past Grid portal projects such as the Grid Portal Development Kit (GPDK) and the Astrophysics Scientific Collaboratory (ASC) portal. The authors developed a portal framework that will provide a model to ease the addition of new functionalities and increase collaboration within communities. They also developed an instance of the GridSphere portal (also known as the GridLab portal) to provide access to the available services within the GridLab project. The motivation to develop the GridSphere portal framework stems from the inability to reuse code in the presentation layer and also due to the pervasiveness of many application specific portals or stove pipe web applications that made code re-usability extremely difficult. Its architecture takes two forms: The first is the general portal framework for assisting virtual organisations (such as scientists and project developers) and the other architecture aids in the development of reusable modular components otherwise known as portlets. The GridSphere portlet API inherits much of their functionalities and methods from servlets such as init(), service() and destroy() methods and makes use of the standard portlet modes such as view, edit, configure and help. The GridSphere portal framework provides three primary core services such as the LoginService, the UserManagerService and the AccessControlManagerService and enables some default set of core portlets such as Login, User management, Account management, Account request, etc.

### 2.4.2 Liferay

Liferay is a popular open source framework that enables users to create attractive web portals (Sarang, 2009). It provides a runtime environment for hosting java based portal applications, otherwise known as portlets. The web portals which Liferay enables usually consist of a wide range of applications such as blogs, wikis, discussion forums, shared calendar, etc. According to (Ardizzone *et al.*, 2012), Liferay portal framework which offers an easy-to-use “web 2.0” interface using AJAX and other presentation layer technologies is an award winning portlet container. It has features such as GUI-based personalisation, drag-and-drop portlets, dynamic navigation and an instant-add portlet library.

### 2.4.3 Spring Framework

The Spring Framework provides a comprehensive programming and configuration model for modern java-based enterprise applications and on any kind of deployment platform. It can serve as the backbone for the business object layer, or middle tier, of a J2EE web application. Spring provides a web application context concept, a powerful lightweight container that seamlessly adapts to a web environment and can be accessed from any kind of web tier, whether Struts, WebWork, Tapestry, JSF, Spring web MVC, or a custom solution (Gupta and Govil, 2010). A key element of the Spring framework is infrastructural support at the application level. It focuses on the plumbing of enterprise applications so that teams can concentrate on application-level business logic, without unnecessary ties to specific deployment environment.

### 2.4.4 Application Hosting Environment

Application Hosting Environment (AHE) is a web services based environment for hosting scientific applications on the Grid. It is a lightweight, easily deployable environment designed to allow scientists to quickly and easily run unmodified, legacy applications on Grid resources, managing the transfer of files to and from the Grid resource and allowing the user to monitor the status of the application (Coveney *et al.*, 2007). The functionality provided by the AHE is application-centric: applications exposed as web services with a well-defined standards-compliant interface. This allows the computational scientist to manage application instances on a Grid in a transparent manner, thus greatly simplifying the user experience.

### 2.4.5 GridSpace

GridSpace, based on the component programming methodology and Semantic Grid initiative achievements, employs decomposition, dynamic organization and semantic comparison techniques in order to provide a new, abstract layer for programmers of Grid applications (Gubała and Bubak, 2005). It refers to a focused Service Space where a group of related Grid services forms a domain-specific Grid service community in order to facilitate dependable collaboration through trust-driven service selection and semantic-based service discovery. This approach sees the integration of Grid service, Semantic Grid, and Web2.0 to realise and support service-orientation towards a radical transformation from resources to services. GridSpace is therefore used as a platform for component-based Grid programming with use of various tools employing semantic Grid concepts.

## *2.5 Science Gateways*

A Science Gateway is defined as a digital interface to advanced technologies which is used to provide adequate support to science and engineering research and education (Lawrence et al., 2015). It is a set of tools, applications and data that has been integrated via a portal or suite of applications and customised to meet the needs of a specific community, usually in a graphical user interface. They are used to provide access to community resources such as software, data, and high-performance computing and can also be used for collaborative activities. More than just collection of applications, Science Gateways also enable users to store, manage, catalogue and share large data collections or novel applications that cannot be found elsewhere. There are different types of Science Gateways being used for specific purposes such as the application specific Science Gateways for which the portal framework was developed (to aid in the rapid development of portlets) and the user specific Science Gateways such as the ones provided by the OGCE. Some of the Science Gateway frameworks as well as the application specific Science Gateways are summarised in Table 2.2 and Table 2.3, respectively.

### 2.5.1 Science Gateway Frameworks

To develop Science Gateways for different scientific domains, it is important to adopt a Science Gateway framework with the required back-end that will provide developers with the necessary tools for building such web portals. In light of this, several frameworks have been developed for the rapid development of Science Gateways for different communities of practice. Such Science Gateway frameworks include but not limited to Catania Science Gateway Framework (CSGF), WS-PGRADE/gUSE, GridPort, P-Grade, and VinetoolKit (See Table 2.2). These aforementioned Science Gateway frameworks, most especially the CSGF and the WS-PGRADE/gUSE, represent the most commonly used Science Gateway framework in Europe. The following section therefore gives a brief discussion as follows:

#### 2.5.1.1 The Catania Science Gateway Framework (CSGF)

The Catania Science Gateway Framework (CSGF) developed by (Barbera, Fargetta and Rotondo, 2011) is one of the most commonly used Science Gateway framework in Europe. This framework is built on the Liferay portal due to its extensible architecture which makes it compatible and possible to be used with other authentication and authorisation frameworks employed by most organisations. This work centres on the integration of different technologies to develop Science Gateways for different user communities. They make extensive use of Certificate Authorities (CA) and IDentity Federations (IDFs) to support a common framework for managing access to resources by the different participating organisations. Also, they make use of shibboleth system, a tool based on the OASIS Security Assertion Markup Language (SAML), for the communication of user authentication, entitlement and attribute information to support cross-organisation Single Sign-On (SSO). This is to ensure the federation of different organisations with different authentication policies and to allow users to access Grid resources based on the organisation they belong. A big advantage of the CSGF is that (to submit jobs to the infrastructures) a user does not need to have personal digital certificates or belong to any virtual organisations. Proxy certificates which are mandatory to execute actions on the Grid are created using the robot certificates. These robot certificates (which are special kinds of X.509 certificates) are stored on the e-token server and (due to security reasons) can not be accessed by external parties. In addition to using the Shibboleth system and Robot certificates, a LDAP server where different user roles and groups are stored is also utilized. Also, the authors made use of a layer known as Simple API for Grid Application (SAGA) and the Java implementation of the SAGA standard (JSAGA) which abstracts the different middleware implementations. The framework for Science Gateways which was developed in Catania is fully web-based and adopts official worldwide standards and protocols through their most common implementations. Different standards that are supported include JSR 168 and JSR 286 also known as portlets 1.0 and 2.0 standards, OASIS Security Assertion Markup Language (SAML) standard and it's shibboleth and Simplephp implementations, the Lightweight Direct Access Protocol (LDAP) and its OpenLDAP implementation, the Cryptographic Token Interface Standard and it's Cryptoki implementation, and the Open Grid Forum (OGF) simple API for Grid applications (SAGA) standard and it's JSAGA implementation.

CSGF merges different levels of security mechanisms and (based on the credentials) users can access computing resources. It has been designed to address the needs of specific scientific communities by making provision of the existing standards for the distributed infrastructure needs and web/internet environments. The overall architecture of the CSGF comprises of three main components mainly the AAI (which manages user authentication and authorisation), web applications such as the JSR286 portlets and the Catania Cloud and Grid Engine. Each of these components is made up of a core functionality of the framework which adopts the most relevant standards to ensure the long term sustainability of the system. The CSGF provides a full-featured environment that allows the creation of high-level user interfaces which is able to submit e-Infrastructure jobs, access data and metadata content, provide both secure and anonymous access to e-Infrastructure services and help potential users in managing the technical details of the underlying infrastructures.

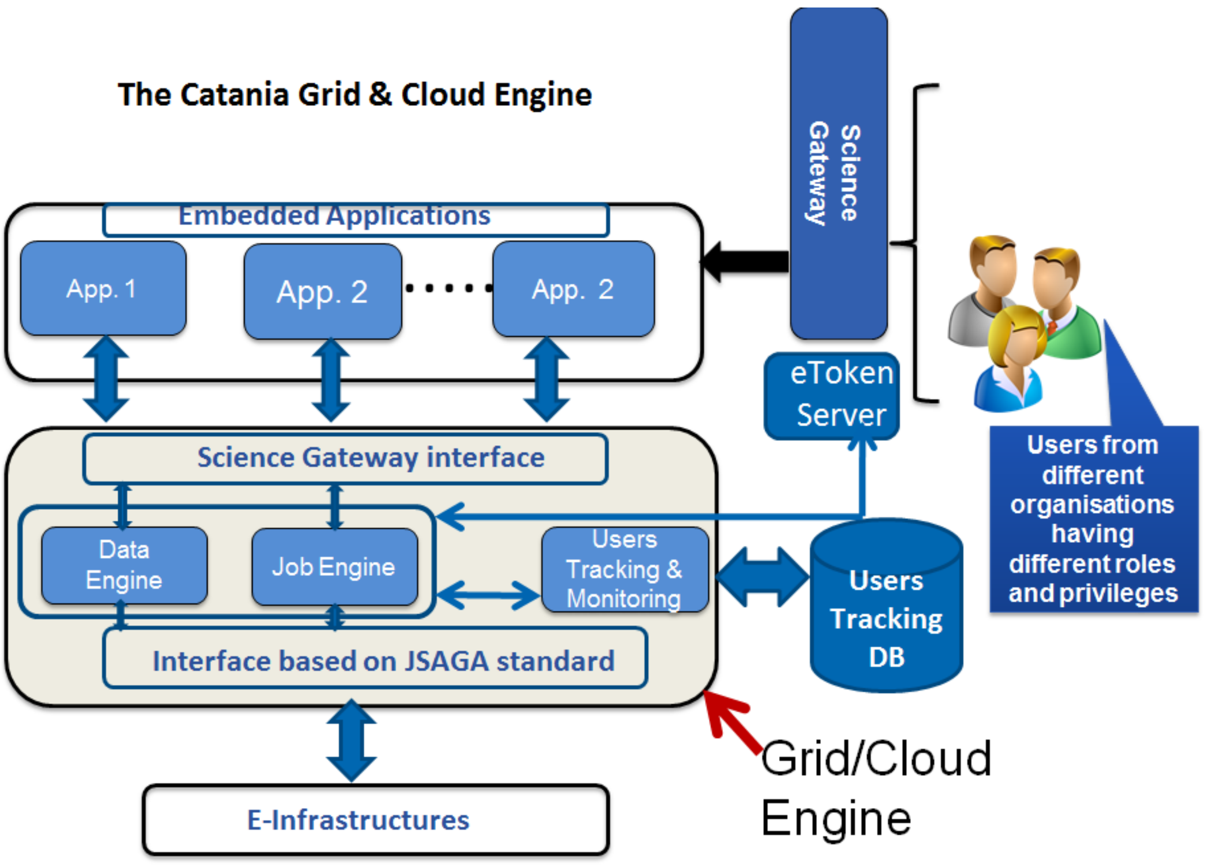

The Catania Grid & Cloud Engine
Science Gateway
Embedded Applications
App. 1
App. 2
App. 2
eToken Server
Users from different organisations having different roles and privileges
Science Gateway interface
Data Engine
Job Engine
Users Tracking & Monitoring
Users Tracking DB
Interface based on JSAGA standard
E-Infrastructures
Grid/Cloud Engine

Figure 2. 8 Catania Science Gateway Framework Adapted from (Fabiyi *et al.*, 2016)

***Science Gateway/Liferay portlets***

This presents an interface that contains the different scientific application which have been ported on the Science Gateway. For the development of the basic element of the Science Gateway, the JSR 286 standard (also known as "portlet 2.0") was adopted (Bruno et al., 2013b; Barbera et al., 2013). The award winning Liferay web portlet framework (which is the most popular framework for building Science Gateways) is the portlet container being utilised in the development process of all portlets in the CSGF. Some of its important features include its rich, easy-to-use web 2.0 interface using AJAX and other presentation layer technologies. It also features an instant-add portlet library, dynamic navigation, drag-and-drop portlets and GUI-based personalisation.

***eToken server***

For the different distributed infrastructures (and based on the GSI security) all transactions must be signed with proxies which are generated by standard X.509 digital certificates. To achieve this feat using the Science Gateway technologies the eToken server, which is a mechanism that generates on-the-fly proxies upon user request, was incorporated as part of the Science Gateway framework. This eToken server generates proxies known as robot certificates. The Robot certificates are standard digital certificates which are stored in USB smart cards otherwise known as eTokens. As a result, robot certificates are mapped to applications which users can then execute without making use of personal credentials since different proxies are created according to the roles and privileges that were stored in the LDAP registry.

#### *Users Tracking and Monitoring*

This module ensures that users who can access and execute jobs on the distributed resources can not flout the underlying rules of any distributed architecture. It achieves this by directly managing and tracking all user memberships and activities performed by each user of the infrastructure. This key feature serves as a means to help in realising the traceability policies that are required by the EGI specification which is one of the most important requirements of the Grid Security Infrastructure. This specification ensures the non-repudiability of all Grid operations made by the users of the infrastructure (Bruno et al., 2013b).

#### *Users Tracking Database*

This module (as shown in Figure 2.8) is tightly coupled with the users tracking and monitoring and is used to store information on all DCI usage by each user being captured. It controls the rate of DCI interactions being initiated via the Science Gateway. It is developed to adhere to the strict rules of the EGI VO Portal Policy and EGI Grid Security Traceability and Logging Policy (Ardizzone et al., 2012b).

### *Catania Grid and Cloud Engine*

The core module of the CSGF is the Catania Grid and Cloud Engine. It makes use of standard technologies to interact with the underlying distributed infrastructures via the Scientific Gateway presentation layer. It handles all e-Infrastructure transactions. It is a generic software module that connects the Science Gateway presentation layer with all underlying distributed infrastructures and middleware. It enables developers to build new Science Gateways simply by exposing an interface to the underlying infrastructures and middleware. The Catania Grid and Cloud Engine software layer are made up of the job and data engine.

Both functions (job and data engine) can interact with the underlying infrastructures by calling the JSAGA APIs for job and data management. By using standard technologies, the Catania Grid and Cloud engine can serve as a major link between the underlying distributed infrastructures and the Scientific Gateway presentation layer. By adopting standard technologies such as SAGA (and their JSAGA implementation) developers can build new Science Gateways in little or no time.

**Job Engine**

The Job engine is used to map job operations to JSAGA functions. It also allows users to fully exploit e-Infrastructure job management services (Ardizzone et al., 2012b). This is the most relevant aspect of the Catania Grid and Cloud Engine within the CSGF. Using the Job engine, the whole life cycle of job execution is managed from the submission of jobs to the retrieval of outputs. All Job requests received from the Science Gateway interface (for submission and execution purposes) are managed by the Job engine. In addition to job submission and management, it also handles job status checks and retrieval of outputs which are incorporated as part of the JSAGA functionalities. It handles preliminary operations for job execution such as the mapping of proxies to jobs, identifying available resource manager and so on. In addition, it facilitates an interface to the eToken server for generating proxies from robot certificates. To account for all user operations, it provides the User Tracking and Monitoring module with all necessary input information to control Grid interactions.

**Data Engine**

The Data engine module maps JSAGA functions to data operations and allows users to fully exploit e-Infrastructure data management services (Ardizzone et al., 2012b). For a data service to be considered relevant, it must provide users with the possibility of arranging files in folders and ordered in tree format in the same way file systems behave on a physical disk (Bruno et al., 2013b). The Data engine therefore enables a direct transfer of services between the Science Gateway and the DCIs. It provides client APIs that satisfy the standard protocol of interacting with different storage elements within the framework.

**SAGA/JSAGA interface**

The CSGF offers a customisable environment which tailors the need of different user communities. It was designed to address large user communities as well as support access to different kinds of distributed systems using the Grid and Cloud Engine APIs simply by exploiting the SAGA standards (Barbera et al., 2013). The SAGA/JSAGA enables the creation of unique interfaces to different middleware stacks so that Science Gateways can exploit resources of different DCIs. To perform middleware independent job and data management and access the underlying infrastructure, the SAGA and its JSAGA implementation is the adopted standard. Simple API for Grid Applications (SAGA) belongs to a family of related standards which was specified by the Open Grid Forum (OGF) that defines an application-programming interface (API) to incorporate specific distributed computing functionalities. JSAGA library (which incorporates the three main features such as security, file management and Job management) implements the SAGA specification by providing a lightweight, modular and pluggable set of java libraries for interacting with the underlying infrastructures and middleware. Consequently, the API aligns with all middleware standards within the OGF standard.

**The Science Gateway interface**

The Science Gateway interface simply consists of the different functions to interact with both the Catania Grid and Cloud Engine and the User Tracking DB.

***Accessing the CSGF Science Gateways using the AAI Module***

The CSGF consists of an AAI module that manages user authentication and authorisation. It authenticates users that make use of Identity Providers (IdPs) and who are members of one or more Identity Federations. It provides the user membership management for the portal by making use of identity federations and identity providers. In CSGF, user authentication and authorisation are managed by using these two different modules (Bruno et al., 2013b). Identity federations consist of one or more identity providers as single entities. They are made up of the agreements, standards, and technologies which enable the portability of identity and entitlements across autonomous domains. By setting up and supporting a common framework helps different organisations to manage access to online resources. Identity providers are the entities that are used to identify users and authorise user membership of a given community. The CSGF makes use of several supported IdPs and each may belong to one or more Identity Federations. As such, SSO service is enabled across SPs for each Identity Federations thereby enabling SSO across the supported identity providers in the federation. Federations are supported based on the SAML 2.0 standard specifications and its Shibboleth implementation and SimpleSAMLphp and for the support of different user identities.

One of the most important requirements for having a Science Gateway is to ease the access to distributed computing and storage resources. To fulfil this requirement and (at the same time) satisfy the security level required by the distributed infrastructures, authentication and authorisation mechanisms at various levels have been conceived to provide easy and secure access to applications.

***Authentication***

IDF, whose aim include setting up and supporting a common framework for different organisations to manage access to online resources, is made up of the agreements, standards, and technologies that make the identities and privileges portable across autonomous domains (Casarino et al., 2015). User authentication relies on Identity providers which in turn belong to one or more IDF.

Two technologies which are instrumental to the actualisation of these security mechanisms are the SAML (used to enable the federation of organisations that have different authentication policies) and X.509 robot certificates (which ensure that individuals who do not have a personal certificate can still access the required computing resources). One important mechanism (the Lightweight Directory Access Protocol) is also used to map authorised users to Grid resources. To comply with the rules of the EGI VO portal and EGI Security Traceability and Logging Policy, the Science Gateway includes user tracking database which stores each operation that is being performed via the Science Gateway. This takes care of one of the most important requirements of the Grid Security Infrastructure, i.e. the non-repudiability of Grid transactions.

There are different types of Identity Federations within the CSGF such as the "catch-all" Identity Federation, known as Grid Identity Pool (GrIDP), which consists of both the IdPs that do not already belong to a federation as well as the users of the Science Gateway that are not registered with any IdPs (Bruno et al., 2013b). This is considered to be particularly useful where the general public (who do not belong to any virtual organisation) may need to access the e-Infrastructure for self-learning purposes. Within the GrIDP

Federation, there are the “Social Networks” Bridge Identity Providers that allow user authentication by utilising the credentials which are used for authenticating to other well-known social networks such as Facebook, Google+, etc. When a user is granted access to the Science Gateway, they can then select the identity provider from the identity federation which they belong. A new page (the Identity Provider login page) is presented where they can enter their credentials which usually include the combination of username and password. If the user is successful at this point, the Science Gateway control system will check if the user is in the the LDAP registry which will then map their role with the registered user rights and thus grant access.

### *Authorisation*

While the user authentication is enabled by external services such as IdPs and IdFs (services which are independent on the Science Gateway), the user authorisation on the other-hand occurs at the Science Gateway level. When a user request is approved, they (together with their roles and privileges) are stored in a LDAP-based registry. When a user has been granted access to the Science Gateway, they can then sign on and execute jobs using the requested portal application.

#### *Robot Certificates*

Another mechanism that has been implemented to ease the access to distributed computing is the Robot certificates. This was introduced on the basis that the management of personal digital certificates for accessing DCIs has proven to be extremely difficult to use (especially for non-experts users) which may consequently hamper the adoption of the technology. The Robot certificate was therefore introduced to make a smooth and transparent access to Grid infrastructures (on behalf of users) by integrating them with traditional general purpose portals and Science Gateways. These Robot certificates are often stored in smart cards (temper-resistant devices) to improve security and avoid fraudulent use of the private keys (Ardizzone et al., 2012b). The Robot certificates are often managed by a multi-threaded server (otherwise known as e-Token server) which is created and configured to manage the list of robot certificates.

***LDAP-based Registry***

This has been created to store and manage user roles and privileges for Science Gateway users. This mechanism is used in mapping authorised users to Grid resources.

### 2.5.1.2 Web Services Parallel Grid Runtime and Developer Environment Framework (WS-PGRADE/GUSE) workflow system

The work done by (Balasko, Farkas and Kacsuk, 2013) outlines an approach to developing web based portals in which a generic purpose work flow management system otherwise known as WS-PGRADE was developed. Based on the concept and the lessons learnt from the first generation P-GRADE portal, this second generation P-GRADE portal introduced many advanced features both at the work flow and the architecture level. This framework utilises a multi-tier service architecture that can support various user communities/domains and different types of DCIs. This includes the Architectural tier, the Middle tier and the Presentation tier.

The Architecture tier (through the DCI Bridge job submission service) allows access to different types of DCIs. The Middle tier contains gUSE services such as gUSE repository which is used for the management, storage, sharing and execution of workflows and the presentation tier is the layer that presents the graphical WS-PGRADE user interface. The gUSE contains a set of services such as workflow storage, file storage, workflow interpreter, gUSE information system and an application repository for realising the workflow management backend of the portal. It adopts the DCI-Bridge (a standard BES interface) for executing jobs on the different infrastructures. The WS-PGRADE portal uses three different modes such as the End-User Mode (a generic yet simple interface), Application Specific Module (that generates a web-based interface from scratch) and the Remote API (for users who have their own interface but still wants to make use of the workflow management and execution capability of WS-PGRADE/gUSE) for web portal development. All these functionalities ensure that the WS-PGRADE has the tools, APIs and the interfaces that can aid in the customisation of an application specific gateway.

Kacsuk (2011) summarised the most advanced features of the P-GRAGE Grid portal and introduced the second generation P-GRADE portal known as WS-PGRADE. This Grid portal, as discussed above by (Balasko, Farkas and Kacsuk, 2013), advances the first generation P-GRADE portals by introducing a new workflow system and architecture concepts and (though P-GRADE is declared as a generic-purpose e-science portal) it is customised to create an application-specific gateway by using a module known as the Application Specific Module for specific application/user domain. Furthermore, the author compared the three major variants of the P-GRADE portal consisting of the original P-GRADE and the NGS P-GRADE (first generation), and the WS-PGRADE (second generation) which advances the first two portals by adding new and improved features such as the gUSE repository, parameter sweep application, and a more sophisticated workflow engine called Zen. The members of a user community can be classified into two categories of Grid application developers (who develops grid applications) and end users (who executes the available applications), and a Grid portal should support at least one or both of these communities. In addition, an application repository is built with the Grid portal to enable scientist to publish their templates for other members of the community thereby facilitating collaborations between e-scientists and application developers. It adopts a user interface/application/high-level services layer kind of architecture that overlays the high-level services above the Grid middleware. The application layer resides on top of these high-level services layer and finally, the graphical user interface layer is built on top of the other layers as shown in Figure 2.9 below.

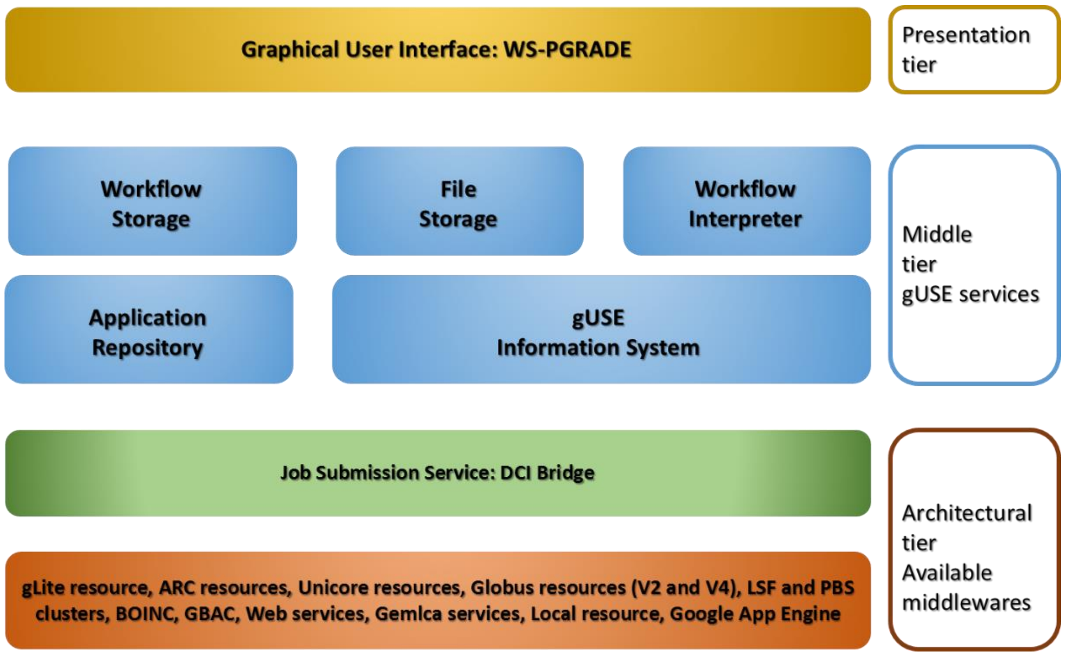

Graphical User Interface: WS-PGRADE
Presentation tier
Workflow Storage
File Storage
Workflow Interpreter
Application Repository
gUSE Information System
Middle tier gUSE services
Job Submission Service: DCI Bridge
gLite resource, ARC resources, Unicore resources, Globus resources (V2 and V4), LSF and PBS clusters, BOINC, GBAC, Web services, Gemlca services, Local resource, Google App Engine
Architectural tier Available middlewares

Figure 2. 9 WS-PGRADE/gUSE architecture by (Balasko, Farkas and Kacsuk, 2013)

WS-PGRADE is a generic purpose workflow-oriented graphical user interface. It is used to create and run workflows on various DCIs such as Grids, Clouds and Clusters. It makes use of two (2) API interfaces namely, the ASM API and the Remote API. This is used to meet the needs of different user communities by creating application-specific Science Gateways. According to Balasko et al. (2013) while the task of a Science Gateway is to hide the details of the underlying middlewares from the users via an easy-to-use user interface, workflow systems on the other-hand are used to enable the creation and management of workflows consisting of a number of scientific applications. There are two main categories of gateways namely: the generic DCI gateway frameworks which could be used by the different scientific domain and the application-specific science gateways which target a well-defined set of scientists working in the same field of science. Examples of other Generic DCI gateway framework also include GridPort, P-GRADE, Vine Toolkit, and the CSGF (See Table 2.2).

To create application specific gateways, there are two options available to the user. Either write the gateway from scratch using portal frameworks such as Liferay or customise an existing generic DCI gateway framework according to the needs of a particular scientific domain. The latter approach is better due to the reduction in the production time and cost of producing the gateway. WS-PGRADE was designed based on the lessons learnt from the 1st generation P-GRADE portal. The WS-PGRADE framework is made up of three (3) main tiers namely: the architecture tier, the middle tier and the presentation tier. The architecture tier of the WS-PGRADE portal enables access to many different kinds of DCI through the DCI Bridge job submission service. The middle tier, on the other hand, contains a high-level gUSE services that enable the management, store and execution of workflows. Lastly, the presentation tier provides the graphical WS-PGRADE user interface of the generic DCI gateway framework which can easily be customised and extended.

Different types of users can utilise the generic DCI gateway framework. These are the workflow developer who develops workflows for the end-user scientists, the end-user scientist who does not bother with the details of the underlying infrastructures, the scientists who only require some customisations to run their workflow applications (via the ASM API) on DCIs, and the final set of users who do not require a user interface but simply want to access the gUSE services and execute WS-PGRADE workflows directly via an API. The most distinguishable feature of WS-PGRADE/gUSE in comparison to other Science Gateway framework is that it is workflow-oriented. It has three (3) distinct features such as workflow support, facilitation of multi-DCI workflow execution and enabling the customising of the framework towards application specific Science Gateways.

### *Graphical User Interface – WS-PGRADE (Presentation Layer)*

This tier provides the graphical WS – PGRADE user interface of the generic DCI gateway framework. This layer can easily be customised according to the needs of different user communities. There are different ways in which potential users can make use of the generic DCI gateway framework.

The first categories include the workflow developer who simply develops workflows for the end-user scientists. The activity of the workflow developer includes edit, configure and run workflows and to support the work of the workflow developer, all these activities are supported. After a workflow has been developed, it is uploaded to a repository and scientists can access and make use of them.

The second category of users consists of the end-user scientists. These type of users are not familiar with the features of the underlying DCI or of the structure of the workflow that realises the applications they want to execute. Normally, the end-users only need to download the required workflow from the application repository and parameterise it before execution.

In addition, to support the development of application specific user interface or specific portlets which is used for application specific requirements, an ASM API is provided to aid the rapid customisation of portlets. For this third category of users, the WS-PGRADE end-users user interface is replaced by the customised Application Specific user interface which could then be used to access the gUSE services through the ASM API.

The fourth category is the set of users that already have their Application Specific UI and simply want to employ the existing UI to access various DCIs. Finally, the fifth category is the set of users that want to access the gUSE services without the use of a specific user interface. These sets of users simply want to run WS-PGRADE workflows via a direct API and the gUSE remote API was provided to facilitate it.

***The Middle Layer (gUSE services)***

The middle layer consists of all the gUSE services such as the workflow storage, file storage, workflow interpreter, application repository and the gUSE information system. It was originally developed to support the needs of application development for Grid computing on different DCIs. It allows developers to easily specify the sequences of tasks (i.e. the workflow) to be executed on different DCIs which are required by their application and parallel execution. Since gUSE treats the computing resources of a Cloud as just another DCI, it is therefore ideal for Cloud computing as well.

***DCI – BRIDGE (Architecture tier) – Job submission service***

The DCI Bridge is used to provide standard access to the aforementioned DCIs. The DCI Bridge is developed to eliminate the need to adopt and use different submitters for as many DCIs that need support. The major components of the DCI Bridge include the Resource Registry, the Application Management, the Runtime System and the Monitor component all of which could run within a tomcat based web container. To configure the accessible DCIs, the resource registry subsystem provides an online configuration interface and sends information about the configured resources to other external software components. The Application Management subsystem gives the implementation of the Basic Execution Service (BES) management port type thereby providing the possibility to supervise the software based access of the BES factory service. The runtime system is used to accept all standardised JSDL job description documents from the workflow interpreter service. DCI Bridge (via a well defined communication interface) helps in enabling workflow management systems to access various DCIs. The DCI Bridge utilises the OGSA Basic Execution Service (BES) interface to submit jobs transparently into various DCIs.

#### 2.5.1.3 The Vine Toolkit: A Java Framework for Developing Grid Applications

An earlier web portal project known as the GridPortlets is said to have several shortcomings. Among these is the support for only one Grid middleware (such as the Globus Toolkit), as the extension of support for other middleware is said to be overly complicated. Also, GridPortlets can only be used or tested within a web application server and is dependent on the elements of the GridSphere Portal Framework which has to be managed by a portlet 1.0 API compliant container at run time. It is this sort of problem that the Vine Toolkit seeks to address. Russell et al. (2008) set out to develop a java-based framework that provides developers with an easy-to-use, high-level API for Grid-enabling applications. Vine is said to include an API that advances the original GridPortlets API and it is an actual general library that is deployed for use on a number of platforms such as Java portlet 1.0 environments, Java Servlet, Java Web Start, etc. Unlike the original GridPortlets, the VineToolkit has been developed to support different middlewares and services such as gLite, globus Toolkit, GRIA, OGSA-DAI, SRB, UNICORE and VOMS amongst others. According to (Russell et al., 2008), the Vine Toolkit consists of a base API (the core project) and a programming model which serves as building blocks for each sub projects that addresses major problem areas. Interfaces and classes are contained within the Group Vine (where the user account management and registration concepts are defined) and the Grid Vine (where Grid application development concepts are described). At the time the Java SAGA 1.0 was still evolving the authors planned to work with the SAGA group by simply wrapping the Vine Toolkit.

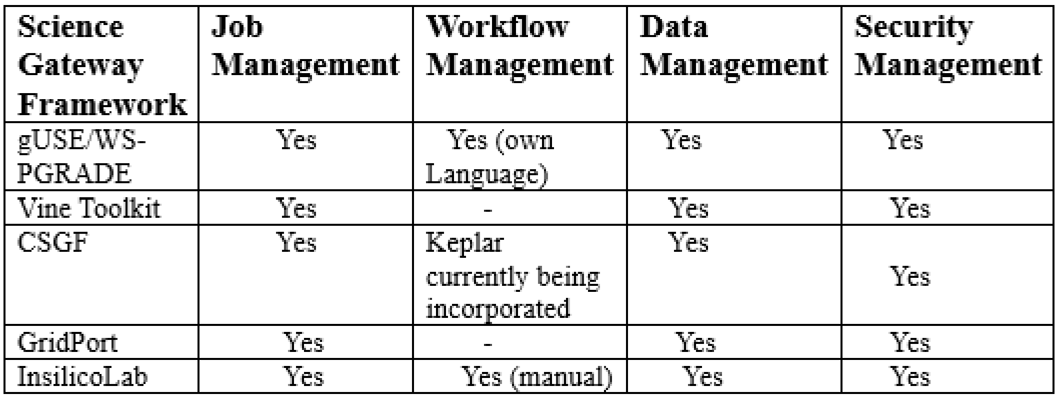
#### 2.5.1.4 A Framework for Domain-Specific Science Gateways (InSilicoLab)

In another framework that was developed for building domain specific Science Gateways, (Kocot et al., 2014) used several case studies/scientific domains such as computational chemistry, cherenkov telescope array, simulations in astrophysics and bioinformatics to gather what they considered as the requirements for the development of a gateway framework. The Requestor-Message Broker-Worker architecture is designed to specify the model of communication between the system components. While the Requestor is an entity that defines the computation to be performed (i.e. a portal), the Worker is the actual program that performs the computation. The Message Broker, on the other hand, is used to coordinate the requests of both the Worker and the Requestor. The architecture of the framework, otherwise known as the InSilicoLab, is composed of three layers namely: Domain Layer, Mediation Layer and the Resource Layer. Even though the framework is usually integrated with a standalone portal for a particular domain, it can however be composed of a larger integration platform such as Liferay portal. The authors identified three types of processing (such as batch operations, immediate operations and on-demand deployment of services) from the case studies that were implemented with a processing model which also provides a framework for creating domain/community specific gateways.

### 2.5.1.5 Science Gateway Framework (List and Comparisons)

Table 2.2 shows the features associated with the different Science Gateway Frameworks. This table includes the relevant information about the different frameworks such as the support for job management, workflow management, data management, and security management. One advantage of customising existing Science Gatway frameworks is that they already provide the required back-end with access to various DCIs which could help save development time. Science Gateway instances or application specific Science Gateways which are developed for different research communities are customised from existing Science Gateway frameworks. Table 2.2 therefore shows a list of the Science Gateway frameworks and their most widely adopted functionalities. These Science Gateway frameworks have been explained in detail in the above section.

Table 2.2 List of Science Gateway Frameworks and their major functionalities (Adapted from Balaz et al., 2013)



| Science Gateway Framework | Job Management | Workflow Management | Data Management | Security Management |
|---|---|---|---|---|
| gUSE/WS-PGRADE | Yes | Yes (own Language) | Yes | Yes |
| Vine Toolkit | Yes | - | Yes | Yes |
| CSGF | Yes | Keplar currently being incorporated | Yes | Yes |
| GridPort | Yes | - | Yes | Yes |
| InsilicoLab | Yes | Yes (manual) | Yes | Yes |

The above Table 2.2 shows the major Science Gateway frameworks which are used to customise instances of Science Gateways otherwise known as application specific Science Gateways. The table shows that, at a very high level, Science Gateways incorporates major functionalities such as job management, workflow management, data management and security management. Science Gateway Frameworks are therefore developed with a more generic interfaces, providing user interface for generic features which may be needed by different user communities and application specific Science Gateways. Each user community typically requires specific functionalities according to their specific needs compared to the original, more generic functionalities of the Science Gateway frameworks from which the application specific Science Gateway are customised. Hence, different communities may also opt to incorporate one or all of the functionalities described in Table 2.2.

### 2.5.2 Application Specific Science Gateways (instances)

An instance of a Science Gateway (otherwise known as application specific Science Gateway) provides a simplified user interface that is highly tailored to the needs of a particular scientific community. In order to develop an application specific Science Gateway, a developer can either create from scratch or customise an existing Science Gateway framework. However, many instances are created based on customising an existing Science Gateway framework because creating the back-end with the necessary DCI access can be time-consuming and there are existing Science Gateway frameworks and services which could be utilised. This section will briefly explain some Science Gateway instances with emphasis on the adopted Science Gateway framework.

#### 2.5.2.1 The GISELA Science Gateways

The Grid Initiatives for e-Science virtual communities in Europe and Latin America (GISELA) Science Gateway is an instance of the Catania Science Gateway Framework (CSGF) that was developed by (Ardizzone *et al.*, 2012). The reason for the low uptake of Grid infrastructures despite the huge investment that has been made is mainly due to the complexity of the technology used in accessing these environments especially for the non-ICT expert users. They identified requirements such as simplicity, use of standards, re-usability and easiness of use as the primary requirements that inspired their work. Based on the CSGF framework that was used, as seen in the work of (Barbera, Fargetta and Rotondo, 2011), GISELA Science Gateway made an extensive use of Certificate Authorities, Identity Federation, LDAP registry, Grid Engine (which comprises of the Job and Data Engine and based on JSAGA) in order to support a common framework for managing access to resources by the different participating organisations and thus establishing a standard way for user authentication and authorisation for the running of jobs and data management. Several implementation of the GISELA project based on the CSGF framework includes several scientific domains such as life sciences, industry sciences, cultural heritage and the field of mathematics.

#### 2.5.2.2 The Decide Science Gateways

The Diagnostic Enhancement of Confidence by an International Distributed Environment (DECIDE) project is another instance of a Science Gateway which was developed based on the CSGF framework. According to (Ardizzone *et al.*, 2011), the DECIDE Science Gateway is developed to enable e-Health based on the European research network and Grid infrastructures for European citizens by providing easy access to high-quality diagnosis and prognosis service for Dementia Disease such as Alzheimer. The DECIDE architecture takes the form of network connectivity, Grid computing resources and domain specific applications. The Science Gateway which is built within the Liferay framework is fully compliant with JSR 268 standard which makes it compatible with other portal frameworks. Also, similar to the GISELA Science Gateway, as seen in the work of (Ardizzone *et al.*, 2012), and due to the Science Gateway framework that was used, the combined technologies that led to the realisation of the DECIDE Science Gateway includes Certificate Authorities, Identity Federation, LDAP registry, Grid Engine (which comprises of the Job and Data Engine and based on JSAGA) to realise a common framework for managing access to different resources. Apart from providing extensive support for gLITE middleware, adaptor interfaces are designed through the use of JSAGA to make the extension of other middlewares such as Globus Toolkit and UNICORE easy.

#### 2.5.2.3 The agINFRA Science Gateway for Agricultural Sciences

agINFRA is another Science Gateway instance that has been developed on top of the CSGF. According to (Bruno *et al.*, 2013), the aim of the agINFRA Science Gateway is to introduce the concept of open and participatory data-intensive science to the agricultural communities by eliminating existing obstacles regarding data sharing and open access to scientific information as well as facilitating the management of relevant data available within these communities. The authors claimed that to overcome these obstacles/barriers, the widely accepted standards used within the CSGF (such as SAGA and SAML) and several other standards must be adopted. Several applications that were specifically developed and ported on the agINFRA Science Gateway to help with data management activities within the agricultural research communities include agrovoc tagging for indexing documents, WEKA for data mining problems, R statistical analysis tool for statistical computing and graphics, etc. Among the services integrated within the agINFRA Science Gateway portal are the Italian Soil Information System, Annotate and Browse Soil Maps and the CLEVER Cloud.

#### 2.5.2.4 “Social” Science Gateways to e-Infrastructures

Another effort that was made in the development of Science Gateways is seen in the work of Carrubba et al (2013) which was built on the Catania Science Gateway Framework (CSGF).

The authors argued that despite the high availability of high-performance computing infrastructures and the efforts being made to improve them, their uptake is still relatively low when compared with potential users. Their low uptake was compared and contrasted with social network sites which have seen a major surge in their usage. This surge was credited to their intuitive and easy to use interfaces. As a result of this revelation, the aim of the authors is to therefore develop a Science Gateway that is plugged directly into a social network so that tools provided by the Science Gateway are directly accessed via the social network. This is aimed at attracting a larger audience by utilising these user-friendly interfaces and authenticating users of Science Gateways with same credentials used for their Social Networks. To increase the number of potential users, facebook was the social network that was chosen due to its high popularity and usage and it is also estimated that many potential users of the DCIs are facebook users. The portal framework (Liferay) that was used in the CSGF provides a complete set of API to interact with facebook which could be done by either integrating the whole Science Gateway or individually porting the applications within facebook. Two of the Science Gateways developed which may be accessed from within facebook includes agINFRA Science Gateway, seen in the work of Bruno et al (2013a) and GARR Science Gateway.

#### 2.5.2.5 A Grid Portal with Robot Certificates for Bioinformatics Phylogenetic Analyses

Some major work was also done in the aspect of simplification and user friendliness of grid/grid portals for wider communities of users such as the work of Bogdanski et al (2007) and Gesing et al (2011). Another work that was done to simplify the usability of grids is seen in the work of Barbera et al (2011). The authors proposed an approach to grid security whereby the process of retrieving Public Key Infrastructure of X.509 certificate is made easy for users by adopting a technology called robot certificate. This approach to retrieve and manage certificates is a standard approach which has been adopted in the Catania Science Gateway Framework (CSGF) which was explained earlier. The new approach to managing certificates will ensure that users who do not own a personal certificate and do not necessarily belong to any virtual organisation can still access and make use of the Grid resources. These robot certificates are usually stored on USB form factor smart cards, called e-tokens and the e-tokens are plugged into e-token servers. These certificates are used to identify a person responsible for an unattended service or a process acting as client and server for a virtual research community. Robot certificates have been created and deployed on all regional infrastructures so developers do not need to worry about it. All they need to do is to access the VPN and the Science Gateway will connect to the robot certificate server, (otherwise known as the e-token server) on behalf of the users in an easy and transparent manner. The functionalities of the robot certificate are tested on the bioinformatics communities in order to perform large-scale Bayesian Phylogenetic analyses on the Grid.

#### 2.5.2.6 A Data-Centric Neuroscience Gateway: Design, Implementation, and Experiences

An instance of a Science Gateway which is based on the Grid and Cloud user support environment known as WS-PGRADE/gUSE framework (Balasko, Farkas and Kacsuk, 2013), was used in creating the e-BioInfra Science Gateway. According to (Shahand *et al.*, 2015), e-BioInfra Science Gateway was built for the analysis of large scale biomedical data on the Dutch e-science Grid. It is designed to help facilitate a simple interface where biomedical researchers can make use of services such as community Grid certificate and semi-automatic file transport. Due to the limitations of the e-BioInfra Science Gateway, such as the need to support richer data resources and operations, a new Science Gateway was designed and implemented specifically for the computational neuroscience research community of AMC. The lessons learnt from the e-BioInfra Science Gateway meant that new Science Gateways that is data-centric (in which everything revolves around data and meta-data) rather than application-centric should be developed. They claimed the old e-BioInfra Science Gateway has functionalities such as transparent authentication and authorisation, semi-automatic data transfer and an extensive set of applications which were mainly centred on applications and their underlying resources. This led to so many data-related errors where invalid input data and inadequate file processing capability are claimed to be the ultimate cause. This consequently resulted into the proposal of new Science Gateways which is data-centric and provide all the necessary tools and services needed to interact with their data for the purpose of data discovery, exploration, preparation and processing.

#### 2.5.2.7 VisIVO Workflow-Oriented Science Gateway for Astrophysical Visualization

Another Science Gateway instance based on the WS-PGRADE framework (a highly flexible interface for the Grid User Support Environment) is built for the Astrophysics communities. According to (Sciacca *et al.*, 2013), a web-based workflow-enabled framework known as the

VisIVO Science Gateways that integrates on different Distributed Computing Infrastructures, a large scale, multidimensional datasets and applications for visualisation and data filtering has been developed. The VisIVO Science Gateway is developed to help with the collaborative visualisation of sharing visualisation experiences as well as enabling the reproduction of specific visualisation results. To obtain a robust analysis of large-scale astrophysical datasets, complex workflows are created and executed on a variety of infrastructures which are easily accessible to both astrophysical researchers as well as the wider public. Several portlets running in a Liferay portal environment enables users to set up, submit, run, and evaluate simple or sophisticated VisIVO scenarios for large-scale datasets exploiting DCI resources. As a result of this technology, the shareable workflows and reusable portlets of the gateway are exploited by both the astrophysicists and the wider public.

#### 2.5.2.8 MoSGrid Science Gateway for US and European Infrastructures

Another instance of a Science Gateway which has been developed on top of the gUSE/WS-PGRADE framework and based on the Liferay portal framework is the Molecular Simulation Grid (MoSGrid) Science Gateway. According to (Gesing *et al.*, 2015), the original MoSGrid Science Gateway which serves the computational chemistry community with interfaces for jobs, workflow, data and metadata management has a few pre-configured workflows which are dependent on the service-oriented Grid middleware (UNICORE). It also utilises SAML assertions to provide a SSO mechanism for the different available resources. The authors argued that to ensure MoSGrid Science Gateway can utilise the resources of the Extreme Science and Engineering Discovery Environment (XSEDE) and Partnership for Advanced Computing in Europe (PRACE), some modifications must be made to the original MoSGrid Science Gateway. They highlighted the main challenges in areas such as security mechanisms, distributed data management and workflows and subsequently proffered solutions. Such solutions include using XSEDE to support user credentials as opposed to using a Science Gateway credential and integrating iRODS for data management since the XtreemFS used in the original distributed data management in MoSGrid is not available in XSEDE and PRACE infrastructures. To resolve the issues surrounding workflow execution, the authors installed Gromacs and NWChem which both serve as comprehensive and scalable open-source solutions for large scale molecular simulations on the MoSGrid Science Gateway.

Barbera et al (2007) set out to provide a problem-solving environment that will allow scientists to access, execute and monitor applications running on Grid resources by simply making use of a web browser. The Portal solution is based on Enabling Grids for E-science in Europe (EGEE) project's middleware Grid portal otherwise known as GENIUS Grid portal and allows users to access the Grid services from various devices such as desktops, laptops, cell phones, and Personal Digital Assistants (PDA), etc., and from anywhere in the world. The Grid portal was implemented on top of the gLite middleware services and it utilises a three tier model for its architecture. This architecture includes the client that runs on a web browser on the user's workstation, the server which comprises of a gLite user interface that oversees the gLite middleware services used for submitting jobs and managing data on the Grid and remote resources. The GENIUS Grid portal adopts the EngineFrame (a tool that provides interfaces to activities that were once command line oriented) to provide a complete separation between the user interface (front-end) and the grid-enabled business logic (back-end). The authors also claimed that (in addition to job submission and data management) other services that were integrated as part of the GENIUS portal include the interactive services and security services.

Kertesz, Otvos and Kacsuk (2014) reports on the collaboration between scientists from various disciplines to execute scientific workflows using web-based portals. More specifically, the authors presented a general way of executing legacy biochemical applications into parameterised scientific workflows, using web based portals to exploit the computational power of the different participating DCIs. In porting the biochemical legacy application, the WS-PGRADE portal framework was chosen. Consequently, the authors presented a way of converting a legacy biochemical application into a parallel scientific workflow application that can run on different DCIs. The functionalities of the WS-PGRADE framework enable users to create jobs which are responsible for the generation of input data, sub-workflows that contains jobs for executing different algorithms and a collector job that gathers the output files of the sub-workflows.

Matsunaga et al. (2007) described an approach to Science Gateways, known as the In-VIGO approach where virtualisation technologies are used to provision dynamic virtual resources on demand in a distributed environment. It is a process where applications which are accessible via user-friendly web interfaces can effectively be turned into Grid services. As a result, the use of virtual application as Grid services along with a Virtual Application Service (VAS) architecture for the generation, customisation, deployment, were described. This is therefore different from other approaches used for building Science Gateways in the sense that it utilises existing and novel technologies to virtualise resources (machine, networks, application and data) as it decouples user environment from the physical resources. The key features of the In-VIGO Science Gateway include facilitation of the dynamic creation and execution environments, secure access to resources, applications and data, virtual application and virtual application service.

Cobb et al. (2007) set out to build a Science Gateway for the TeraGrid resources to support collaboration with the Spallation Neutron Source. The purpose of the Neutron Science TeraGrid Gateway is for Neutron science users to run experiments, generate datasets, remotely perform collaboration activities, perform data reduction, analysis and visualise results and to achieve long-term data in repositories. As a result, they set out to develop a Science Gateway to include all of the aforementioned features for the use of the Neutron science community. In light of the above, they argued that a portal/gateway should make scientists work as easy as possible by requiring no software installations, have user favourite tools (and the ability for users to be able to add these tools), and the portal should also address latency issues. These features were therefore designed and implemented as part of their Gateway.

Wilkins-Diehr (2007) surveyed different community interfaces to Grid resources that were deployed globally, such as generic interfaces for job submission and interfaces for specific scientific domains. The primary purpose of their work was to gather researchers or resource providers who are trying to provide Grid capabilities to a particular science community via portals, web services on resources in operational Grid and get them to access the Grid resources via the desktop. They utilised TeraGrid and conducted thorough interviews with ten Science Gateways which were expected to make use of this Grid. Among the several questions covered include: the type of middleware used, the utilised software applications, the use of authentication, accounting plans and security issues and the different user capabilities. Majority of their work is seen in the summary of the common features within Science Gateways, the different development approaches and the technologies being used. The different Science Gateways which were considered falls into one of three categories: the

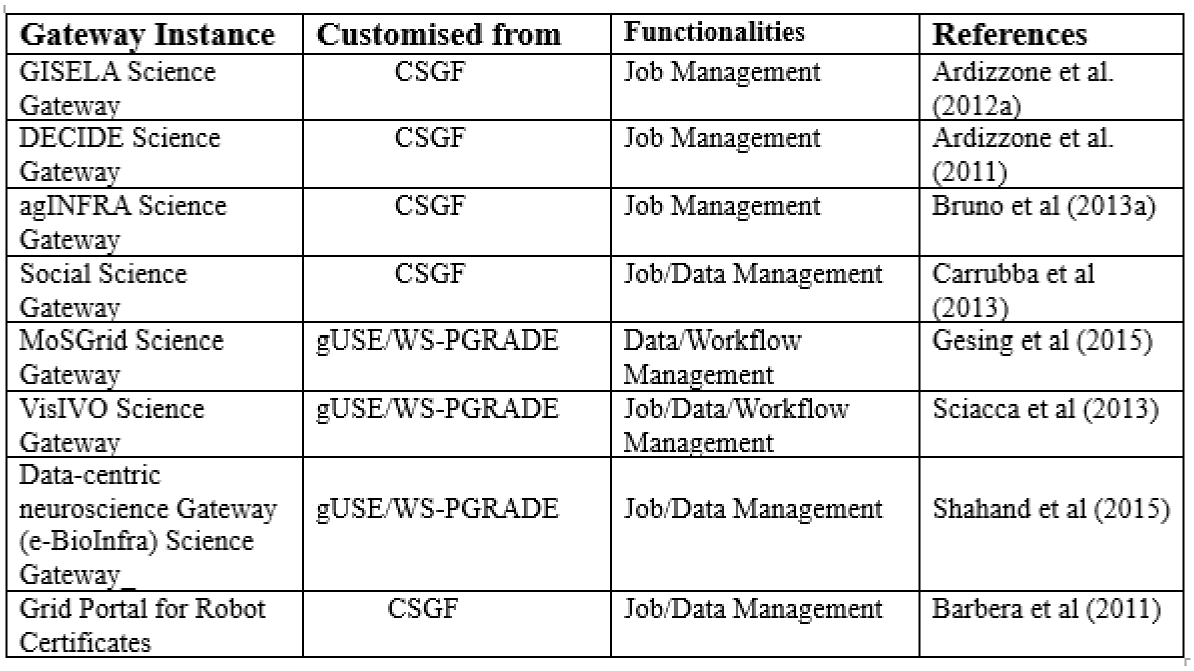
Science Gateways providing specific services for different scientific domains, the Science Gateways providing more generic mechanisms and the required technologies needed in the implementation of Science Gateways. Several science portals discussed were built using a variety of technologies such as OGCE portal toolkit, uPortal, GridSphere, and JetSpeed. In addition, some projects (such as the LEAD portal) require the support for workflow execution whereby access to weather prediction simulations and services running on remote resources is facilitated. Other projects such as the GridSAT portal which provides an interface to complex grid applications. Applications in this group require job management functionality for Science Gateways and provide interface where users can have access to complicated code that is executed on large number of DCI resources. For the more generic resource access portals, projects such as the GENIUS Grid portal (which provides security, job submission and data management functionalities) were discussed. Lastly, enabling technologies which are used to provide transparent access to DCI resources were also discussed. The projects in this category include Grid Resources for Industrial Applications (GRIA), Grid-enabled Application Service Providers (GridASP), etc.

Gesing et al. (2011) reported on the aspect of simplification and user interaction of portals. This research (similar to the work of Wilkins-Diehr, (2007)) is focused on the contributions made on portals in life science field that sees the collaboration of various scientists from different fields such as life science, bioinformatics and computer science. The sole purpose of this is to create a platform to exchange ideas and share their experiences and technological advances, in regards to portal development, in the field of molecular and system biology. To illustrate the solutions used in these portals, the authors opted to use five generic portals namely OGCE, VineToolkit, P-GRADE, Genius and GridPort. The authors claimed that all the aforementioned portals (except Genius) are built on top of Gridsphere portal framework due to its compliance with JSR 168 standards which makes it compatible with other portal frameworks such as Liferay, IBM's Websphere and Apache's Jetspeed. Comparisons were made for the different portals in the aspect of the functions they provide such as VO certificate management, Grid data management, Grid resource monitoring, information service and job execution monitoring. Other comparisons that were made include the ability to support parameter sweep applications, workflow editors and workflow execution.

Another research which aims at collecting experiences for the building of Science Gateways gathered from different scientific areas is the Distributed European Infrastructure for Supercomputer Applications (DEISA). In this paper, (Soddemann, 2007) described the development process from the point of view of both project engineer and software architect of the DEISA material sciences and plasma physics web portal application. This is a slightly different approach to the work done by Thomas (2007), Gesing et al. (2011) and Wilkins-Diehr (2007) in the sense that, instead of sharing experiences and exchanging ideas among scientists from different fields, user requirements were sort out in order to combine current efforts and advance the field. Potential users were involved in the specification of the requirements for the use of remote supercomputing. At the end of the analysis, user requirements include easy access to computing resources, easy way to submit jobs, the creation of input files for complex applications and focus on performing experiments rather than becoming Grid experts.

Welch et al. (2007) developed an approach to developing Science Gateways to manage a community of users whereby the power to authorise individual users is being handed to the community and thereby absolving the resource provider of the responsibilities. Different communities make use of different security models from a traditional single-user point of view, thus a security model for managing community accounts and organised by authentication, authorisation, auditing and accounting (AAAA) which constitutes the four A's of security was proposed. The authors’ motivation for having this kind of community accounts stems from the fact that low-level interfaces such as a command line shell (used in accessing resources) have a steep learning curve coupled with the burden placed on user communities to maintain state for each user that utilises the resources.

Table 2.3 Application Specific Science Gateways



| Gateway Instance | Customised from | Functionalities | References |
|---|---|---|---|
| GISELA Science Gateway | CSGF | Job Management | Ardizzone et al. (2012a) |
| DECIDE Science Gateway | CSGF | Job Management | Ardizzone et al. (2011) |
| agINFRA Science Gateway | CSGF | Job Management | Bruno et al (2013a) |
| Social Science Gateway | CSGF | Job/Data Management | Carrubba et al (2013) |
| MoSGrid Science Gateway | gUSE/WS-PGRADE | Data/Workflow Management | Gesing et al (2015) |
| VisIVO Science Gateway | gUSE/WS-PGRADE | Job/Data/Workflow Management | Sciacca et al (2013) |
| Data-centric neuroscience Gateway (e-BioInfra) Science Gateway_ | gUSE/WS-PGRADE | Job/Data Management | Shahand et al (2015) |
| Grid Portal for Robot Certificates | CSGF | Job/Data Management | Barbera et al (2011) |

Several other Science Gateways were also developed to support scientists of different disciplines with the sole aim of granting access to computational resources in the easiest possible ways. An example of this is seen in the work of Pierantoni *et al.*, (2017) where it is deemed that (since information can be missing or hard to isolate at the right layer) complex and unstructured flow of actions and information poses difficulties in the development and usage of Science Gateways. As such, the work concentrated on the design and implementation of more flexible and user friendly Science Gateways and workflow management systems. Zhao *et al.*, (2017) conceived that the task of bringing data and tools online into a Science Gateway environment is still daunting for domain science users. Therefore, they developed a reusable geospatial data analysis building blocks for Science Gateways which includes core components such as geospatial data management system named iData, libraries for easy creation of geospatial data analysis tools hosted in the gateway, GeoBuilder for creating GIS-enabled data exploration tools, and general purpose tools for geospatial data processing and visualization. Elia *et al.*, (2017) focuses on climate change and biodiversity research by providing a Science Gateway for end-users with a highly integrated environment, addressing mainly data analytics requirements. It is believed that to better understand the mutual interaction between climate change and biodiversity there is a strong need for multidisciplinary skills, tools and a large variety of heterogeneous, distributed data sources. Kee and Schrock, (2018) based their work on the fact that several Science

Gateways have focused on technological tools. They therefore concentrated on the teams behind the tools by creating a checklist for practitioners to access the functioning of an existing team, or strategically design new teams. They believed that these checklist can help domain scientists, center administrators, and computational technologists across tools, projects, and domains reflect on the needs of their teams.

Other researchers such as Smith and Abeysinghe, (2017) concentrated on the aspect of workflows by developing a Science Gateway for Parallel Hierarchic Adaptive Stabilized Transient Analysis (PHASTA) software that supports modelling compressible, steady or unsteady flows in 3D using unstructured Grids. This Science Gateway creates a searchable archive of past jobs to support reproducibility. Similarly, Sciacca *et al.*, (2017) developed a framework that allows astronomers to process new generation surveys of the Galactic Plane to build and deliver a model of Milky Wax Galaxy. As such, a Science Gateway (based on WS-PGRADE/gUSE framework) that operates as a central workbench for the community which helps to deal with the growing data size was developed. The design and implementation of a unified scientific Cloud framework called Science Gateway Cloud was implemented by Kim *et al.*, (2017). This Science Gateway provides a broker between users and providers and is able to process various scientific applications efficiently upon heterogeneous cloud resources. Kiss *et al.*, (2017) also developed a WS-PGRADE/gUSE based Science Gateway that supports various application scenarios on multiple heterogenous federated Clouds. They conceived that both scientific and commercial applications require automated scalability and orchestration on Cloud computing resources. Thsir work therefore centers on how an automated orchestration can be added to Cloud applications without major reconstruction of application code.

### 2.5.3 Early effort to develop Tools and Services that aid in enabling the Distributed Computing Infrastructures

This section discusses the early efforts that were made in order to enable access to distributed resources (most especially to the grid utility computing). Several methods/approaches to building Science Gateways have already been discussed in the previous sections. These methods are seen in the form of frameworks which are used to design and develop simple and intuitive user interfaces for accessing various DCIs. These interfaces are used to perform useful operations such as job submission, user management, data management, and resource monitoring and information discovery activities. Traditionally, an approach to accessing these environments such as the command line interface is very complicated and often very difficult to use most especially for the non-ICT expert users. This can effectively affect the uptake of the technology as their complexities ensure that potential users are not naturally inclined to make use of their services. An alternative to this complex interface therefore is the Science Gateway which ensures that scientists are presented with a more familiar, simple but very intuitive user interface where users can only focus on conducting experiments without having to worry about the details of the underlying middleware and infrastructures that does their computation. Before the emergence of Science Gateways however, several other projects were developed and utilised in enabling access to the Grid utility computing which will thus form the basis of discussion in this section. Some of these projects and their timeline are summarised in Figure 2.10.

An early effort to develop Grid portals is seen in the work of Fox et al (2002) which summarises the status of Grid Computing Environments (GCE) by analysing and categorising the first-generation portal efforts. The authors argued that globus toolkit which is the most widely used Grid middleware does not provide adequate support for building GCEs. As such, they discussed other such building blocks for developing GCEs (other than the globus toolkit) such as Java, CORBA, Python and Perl Community Grid interfaces. Other building blocks also include the Grid Portal Development Toolkit (GPDK) which is a JavaBean suite that is suitable for Java based GCE environments and was designed to support Java Server Pages (JSP). Together with the Commodity Grid (COG) Toolkits, it provides the most widely used framework for developing GCEs that make use of Globus middleware. GCEs were categorised into two main classes namely Problem Solving Environment (PSE) or the application portals and the shell-like system environments otherwise called the GCEShell portals. While PSEs provide custom interfaces for specific sets of applications, GCEShell, on the other hand, grants access to basic command for file manipulation activities. For all the efforts made in GCE projects, a major flaw of this achievement was the fact that there were no common portal programming interfaces which limited the amount of collaboration and code sharing that could be done.

A gradual change began in the early 2002 when two important concepts such as the reusable portal components known as portlets and web service architectures were introduced. This was the period when portlet components became a Java standard where java portlet components were standardized with the Java Specification Request JSR 168. As a consequence, reusable functional components are provided based on standard-based portlets which could then be shared between different portal installations. Alameda et al (2007) in their work reported on the efforts of the Open Grid Computing Environments (OGCE) collaboration where a ‘three-tiered architecture’ model of enterprise portal system was introduced. These layers include the user interface tier, the service tier and the resource tier. The OGCE collaboration is said to be built on this model and the general problem that needs to be solved is summarised as the need to support container-independent portlet development, simplifying development through abstraction layers of Grid clients and the need to support user communities by integrating collaboration tools and services. Other research and development efforts in the OGCE collaboration (in addition to the portlet development environment) include the Grid computing abstraction layers (The JAVA COG KIT), advanced grid services for information, science application and data management and services for group collaboration.

Another early effort at developing computational science portals is seen in the Grid Portal Development kit (GPDK) which was reported in the work of Novotny (2003). As already discussed by Fox et al (2002), the authors noted that GPDK leverages off commodity Web technology such as Java server Pages and servlets as well as existing globus/Grid middleware infrastructure. The GPDK is said to provide several key reusable components for accessing various Grid services in addition to facilitating the development of Grid portals. Similar to the OGCE portals GPDK is based on the standard three-tier architecture however, it adopts the one-three tier used by most Web application servers. The authors also emphasised GPDK’s use of Model-View-Controller (MVC) design pattern in separating control and user interface layer from the application logic. They also noted that the GPDK (much like OGCE) utilises Java CoG toolkit to provide the necessary functionalities needed to implement the various Grid services which was embedded within the core GPDK service beans. GPDK was developed with the sole purpose of providing services such as submission, cancellation and monitoring (job submission activities), storing and retrieving data to and from a variety of storage resources (file transfer), resource discovery mechanisms to discover hardware and software resources (information services) and the ability to perform operations by securely authenticating to remote resources. These services are embedded within the GPDK service beans.

Zhang, Kelley and Allen (2007) attempts to compare two Grid portal solutions namely GridPortlets and OGCE collaboration, both used by portal developers to interact with Grid middleware by providing many of the tools that portal developers need to build their application-specific Grid portals. They set out to investigate the strengths, weaknesses and the limitations of both solutions with a bid to providing an insight into the evolution of Grid portal solutions. While GridPortlets implements its API using Java Commodity Grid Toolkit (Java CoG) by providing a high-level interface for developers to access a range of Grid services OGCE, on the other hand, utilises JavaCoG as its main API for accessing the Grid resources. As already mentioned, the basis of the OGCE architecture is the pluggable components in the form of service and portlets. While GridPortlets consist of many reusable components for its user interface (which could be easily customised into other portlet-based applications) OGCE on the other hand does not comply with the JSR-168 standard which makes it a major disadvantage to portlet development as it is not interoperable with other portlet container.

A Grid portal that leverages a set of high-level Globus-Toolkit-based Grid libraries, otherwise known as the Grid Resource Broker (GRB) was discussed by Aloisio et al 2007. Its primary goal is to hide the complexity of the underlying middleware by mediating between the user request and Grid services. The GRB was considered necessary to bridge the gap between the scientists and the Grid. The GRB is made up of such components as profile manager, credential manager, resource finder, job assistant and job supervisor. The authors utilised Liferay portal framework as the GRB portlet container as it provides a set of JSR-168 compliant APIs to develop portlets using a Model View Controller (MVC) based design pattern.

Another research in the area of Grid web-based portal is seen in the work of Chrabakh and Wolski (2007) which provides a web portal for solving satisfiability problems using the national cyberinfrastructure. This was done with the sole purpose of providing a simple and public interface (called the GridSAT) to complex Grid applications, which run on different distributed computational resources in various national computing centres. The GridSAT is implemented in the form of a master-client model where individual clients share intermediate results by communicating directly with one another. They ascertained that GridSAT is made up of five main components such as the resource manager, the job manager, the client manager, the scheduler and the checkpoint server. In addition to these components, the GridSAT also consist of the monitoring and discovery system (MDS) and the Network and Weather Service (NWS) otherwise known as the external components which the master uses to interact.

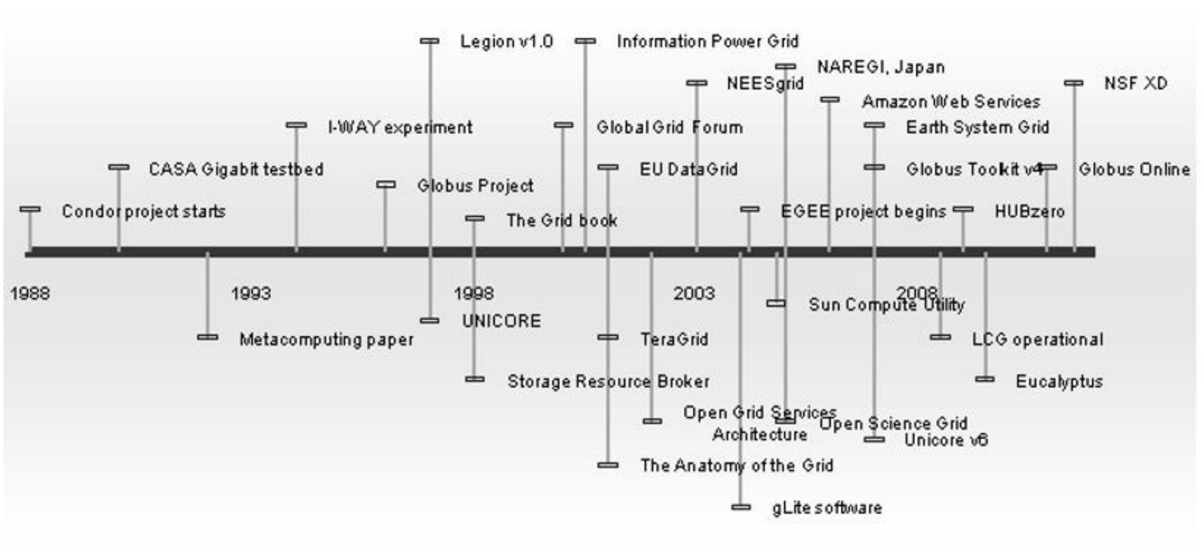


Figure 2.10 Grid Project Timeline between 1988-2011 (Foster and Kesselman, 2010)

Several application portals are created from a particular methodology for Grid portal development. This approach is seen in the work of Thomas et al (2002) where an approach to Grid portal development (otherwise known as GridPort) was used to create application portals that support different user applications and communities. The authors described their experiences in building and developing Grid portals which are all based on the GridPort toolkit. The authors used the globus toolkit as its Grid middleware. As such, resources that could run on globus is added to the portal system. Different aspects of the portal system which supports the general requirements of various portals are represented as layers. These include the clients, NPACI portals, portal services, Grid services and computer resources. The portal system supports two categories of services that are accessible via any application portal. These are information (machine status, load, and usage), and interactive services (secure transactions that provide user with direct access to HPC resources). They also summarised the functionalities of the GridPort based portals as: authentication, jobs and commands, accounts and files.

Another attempt at coordinating and managing distributed computing, data and resources, on distributed infrastructure is seen in the work of Youn et al (2007) on the GEONGrid portal. In their approach, they designed and implemented GEON specific services for building portlet components embedded in the Service Oriented Architecture (SOA). They based their emphasis on the development of reusable services and portlet components that are interchangeable between GEON member sites. Using the SOA, developers can create specific implementations of service components and make provision for services which different portals can share. The authors grouped the main portlet components into three different areas such as account management for the administrator, resource management and science applications and utility services for the portal user.

Akram et al (2007) also followed the approach of Youn et al (2007) of separating the business logic from the presentation logic by using the Service Oriented Architecture (SOA) approach. They ascertained that since portlets are compatible with J2EE, therefore they provide additional capabilities necessary in the SOA. In addition, the authors described ways in which the functionalities within a portal server (with the help of the Web service gateways) are provided to support Grid applications. To carry out their task, i.e. build Council for the Central Laboratory of Research Councils (CCLRC) portal infrastructure that will support research facilities, they leverage on the work of the National Grid Service (NGS) and e-HTPX portals.

Yang, Akram and Allan (2007) did a similar work to Youn et al (2007) and Akram et al (2007) by recognising the need for a clear separation between the user interface, business logic and data layers which could be defined using the J2EE component oriented architecture (COA) and thus help draw a clear high-level picture of the architecture. In their approach, the authors implemented both the business logic and data layers by defining session beans and entity beans in a technology known as Enterprise JavaBeans (EJB). EJB is one of the most important technologies in J2EE which provides the server side component architecture. Also, the authors described ways of converting a COA based system to a service oriented architecture (SOA) based system by exposing the EJB as web services.

A TeraGrid gateway for the Linked Environments for Atmospheric Discovery (LEAD) was described in the work of Christie and Marru (2007). This application portal is designed to make the most of grid resources for the sole purpose of exploring mesoscale meteorological phenomena. They also described the various technologies such as fine-grained capability based authorization, an application service factory toolkit and a web service based workflow execution engine which they utilised in bringing complex and powerful tools to educational and research networks. The LEAD portal is based on the Open Grid Computing Environment (OGCE), a portal toolkit/solution that was earlier discussed in the work of Alameda et al (2007) which provides utilities that are used to develop standards-compliant portlets which are deployed in a Grid environment. A customised version of GridSphere is used as the Portal container as it complies with the JSR-168 specifications for building portlets. Similar to the OGCE portlets, it makes use of the JavaCoG Kit by providing an API for contacting Grid services. The capabilities provided by the LEAD portal includes data management, query tools, workflow configuration, workflow composition and workflow monitoring.

Bogdanski et al (2007) in their work utilised what they termed the PROGRESS project to address the need for a flexible and user-friendly Grid portal environment. In addition to having a low-level grid management system such as GLOBUS or UNICORE Toolkit which has been implemented using a Grid resource broker, the Grid Service provider (GSP) was also integrated. This acts as an extra layer between the user interface (portals) and the Grid resources. They ascertained the two most important services of the GSP as: Job Submission Service and the Application Management service. The authors also attempted to draw parallels between the PROGRESS approach and GridPort portal solution as described in the work of Thomas et al (2002) by claiming that both systems attempt to coordinate low-level services into more sophisticated services and tried to make provision for an integrated high-level API that cuts across a number of low Grid level services.

Ogawa et al (2007) developed the GridASP framework to provide application execution services by binding the application providers, service providers and resource providers together. It also enables the Grid utility computing by realising the Grid-enabled application service providers. According to Ogawa et al (2007), its principal aim is to enable collaboration between diverse specialisations to improve return on investment and to assist new businesses in taking advantage of the available technical know-hows. In addition to using GridPort toolkit and GridSphere as the standard portal framework, the GridASP framework is built on top of several other standard technologies such as the Apache Tomcat Jetspeed-1 portal framework. The resultant GridASP realises a general purpose Grid portal by providing a collection of services, scripts and tools and thus provide features such as job management, data management, visualised workflow editor and resource brokering services.

Fabra et al (2015) proposed a management and execution framework that can integrate different heterogeneous computing resources. Their research stems from a lack of metadata to describe the resources in the Universia repository which complicates their classification, search and recovery. To resolve this issue, the authors identified ADEGA algorithm as a tool to semantically annotate the available educational materials across the different platforms, such as Cloud, Grid and Clusters. The time required to achieve a processing of up to 15 million educational resources was estimated to be more than 1600 years of CPU time.

In an attempt to provide users with a single point of access to HPC resource centres in Europe, Oleksiak et al (2007) observed that most of the HPC centres in the HPC-Europa project were already deployed with their individual site specific Grid infrastructures and HPC and (as a result) an HPC-Europa portal that will provide a flexible, intuitive and transparent user interface would be of utmost importance. Thus, the underlying infrastructures and the heterogeneity (even of Grid middleware) of the resources and services provided are hidden from the users. This framework is built on top of the Gridsphere portal framework and it follows a separated business logic (portlet services) which is based on the Spring framework. The fact that all the portlets included in the SPA are all JSR-168 standard compliant makes it portable between portal containers/frameworks as seen with both Gridsphere and Spring framework. The authors claimed that the support for the management of applications for building Science Gateways for specific domains or groups of applications was also integrated with the SPA portal. Functionalities provided by the SPA portal include job submission, job monitoring and control, accounting, data management, authorization, and resource information.

In order to integrate numerical response from thousands of large-scale earthquake engineering computational simulations, a simple web-based computational portal framework was developed by Youn et al (2014). The resultant framework enables access and use of high performance computing resources. It also facilitates an environment that is able to upload input model files, edit input model file, create job scripts, query running job status and specify HPC job parameters. An effective and efficient simulation environment that helps scientists in their daily work and speeds up scientific results/discoveries may be promoted using the web-based science portal.

According to Binns et al (2007) the lack of sufficient access to advanced visualisation resources prompted the development of visualisation gateway which provides simplified access to these resources. In this work, the authors described the existing implementation of a system known as Tgviz, which makes resources available to a wide range of users and implemented on the OGCE and uPortal. Also, the ParaView portlet (an application that supports models for processing and visualising large data sets) was discussed. They also raised some potential technology and policy issues surrounding the TeraGrid resources, such as security and data management and possible solutions to them.

Also similar to the work of Wilkins-Diehr (2007) and Gesing et al (2011), where scientists from different fields gather together to exchange ideas and share experiences in order to advance the field of Grid portals, is the work of Thomas (2007). The author summarised the outcome of a workshop that focused on Grid Computing Portals and its environments. The purpose of this workshop is to create an environment for portal developers to share and exchange ideas in order to advance the field of Grid portals which consequently resulted in a comprehensive survey of Grid portals. At this workshop, papers that were presented fall into two categories of technologies (i.e. portal frameworks and SOA/Web technologies, Grid usage, and sensors/instruments) and applications (i.e. water resource management, coastal observation, computational material science, collaborative informatics and geology). A component-based architecture based on portal frameworks such as Jetspeed, the OGCE, GridSphere, Sakai and Java/EJB's, were used for most of the portal architectures. This architecture includes the lowest layer that consists of the Grid and compute resources, the Web/Grid service API layer, the portlet API layer and the Portal login layer. Tasks performed by application portals include data access, job management, account and credential management, visualisation of results and workflow management.

While several authors examined the access to the Grid environment by developing methods and approaches for doing it, Jha et al (2013) on the other hand examined a wide range of scientific applications in order to provide a better understanding of the different abstractions in use and how they can be improved to support the future development, deployment and execution of high performance scientific applications on a broad range of infrastructures. In order to identify the common utilities, patterns and abstractions that exist within the different scientific applications that were being considered, the authors surveyed the existing models and methods in use. In their findings, it was discovered that there are no single coherent means by which distributed applications are enabled as well as the lack of generality in the different applications as a result of the various ad hoc solutions used for developing and executing many distributed applications.

## *2.6 Workflow Management System*

The previous sections discussed efforts that were made to facilitate the execution of scientific applications on distributed systems from projects such as Grid portlets, web portals, GPDK, OGCE, Science Gateway frameworks and their corresponding instances. In addition to the aforementioned approaches, another method used to systematically execute these applications on DCIs is the workflow management systems. Workflow is "the automation of business process, in whole or in parts, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules" (Jablonski and Bussler, 1996; Hollingsworth and Hampshire, 1995). They are usually done to model and execute scientific experiments. Workflows are usually of two types namely concrete and abstract workflows. The concrete workflow involves the listing of the tasks to be performed together with the instructions required to carry out the tasks. The abstract workflow, on the other hand, consists of the information about the task and how these tasks are interconnected.

According to Görlach et al. (2011) and Deelman et al. (2009) a scientific workflow comprises of four major phases in its lifecycle. These are the composition, deployment, execution and monitoring and the analysis step. A SWMS is therefore defined as means to process workflows and organise data sets for many scientific computing scenarios. Several SWMSs are now being used extensively in many scientific fields such as computational engineering, astronomy, modelling and simulation, etc., to perform scientific operations. Such SWMSs include Galaxy (Goecks, Nekrutenko and Taylor, 2010), Pegasus (Deelman et al., 2015), Kepler, Swift (Zhao et al., 2007), Askalon (Fahringer et al., 2007), Chiron

(Ogasawara et al., 2013), Triana (Taylor et al., 2007), Taverna (Oinn et al., 2004) and the SWMS Gateway framework such as the WS-PGRADE/gUSE (Kacsuk et al., 2012) which is discussed in detail in Section 2.5. A brief discussion about these different types of SWMSs will now be summarised as follows:

### 2.6.1 Types of workflow management system

***Galaxy*** was developed for data intensive scientific research communities such as the bio-informatics field. It is a web-based open source infrastructure project which is integrated on the Cloud via the cloudMan middleware and is easily deployed on the Amazon EC2 (Afgan et al., 2010). It is made up of different layers such as presentation, user services, Workflow Execution Plan (WEP) generation and the infrastructure layers which are used for the execution of scientific applications.

***Pegasus*** is used in a wide variety of disciplines such as bioinformatics, astronomy, climate modelling, etc. (Deelman et al., 2015). It is a WFMS that ensures that complex scientific workflows are mapped onto distributed resources. A series of configurable refinements are made to ensure that abstract workflows are transformed into executable workflows. It is portable to different distributed infrastructures such as grid and cloud resources. Its main components include mapper, local execution engine, remote execution engine, job scheduler, and a monitoring facility. Similar to the Galaxy SWMS, it consists of different layers such as presentation, user services, WEP generation and Infrastructure layer

***Kepler*** is a workflow orchestration, composition and construction SWMS. It utilises different workflow engine and workflow models to perform data modelling and analysis. It also consists of the different layers such as presentation layer, user services layer, WEP generation layer and the infrastructure layer. According to (Wang and Altintas, 2012) Kepler operates with the Cloud via Kepler/Amazon EC2 actors and thus creates EC2 virtual machines and Elastic block store volumes.

***Swift*** is a SWMS that was developed from the GriPhyN Virtual Data Systems (VDS). It is used for scheduling and optimising the execution of programs using a simple virtual data language. For the execution of scientific workflows, it performs five major functions namely program specification, scheduling, execution, provenance management and provisioning of resources. In addition, it also consists of the different execution layers such as presentation, user services, WEP generation and infrastructure layers. It can also provide dynamic provisioning for Clusters, Grid and Cloud computing.

***Askalon*** is a SWMS that was originally developed for the Grid environment. It supports the composition and execution of scientific workflow applications and it is primarily used to simplify the optimisation and development of Grid workflow applications that uses the Grid computing power. However, it is now deployable in the cloud environment via the dynamic creation of virtual machines (Fahringer et al., 2007). It utilises different layers for the execution of scientific applications. These layers include the presentation, user service, WEP generation and the infrastructure layers.

According to (Oinn et al., 2004) taverna is an SWMS open source myGrid project which was developed to provide a high-level middleware that exploits grid technologies. It can invoke services such as AXIs for web service applications and WSDL4J for data documentation. In addition, it consists of the different layers such as the presentation, user services, WEP generation and the infrastructure layers which are used for the execution of scientific applications.

***Chiron***, an algebraic-based parallel scientific workflow engine, is designed for the execution of parallel workflows in an HPC environment (Ogasawara et al., 2013). The algebraic workflow is used to aid better distribution of scientific workflow execution in parallel. It also consists of the presentation layer, user services layer, WEP generation layer and the infrastructure layer. Chiron uses a Message Passing Interface (MPI) to enable communication between processing nodes and could be deployed on the cloud through an extension known as Scicumulus, which ultimately enables dynamic provisioning.

***Triana*** is a Grid based problem-solving environment (PSE) and SWMS. It was first developed for data analysis for gravitational waves project GEO 600 (Taylor et al., 2007) and it integrates a variety of workflow construction and management components. It utilises a drag-and-drop graphical workspace for workflow composition. It also consists of the presentation layer, user service layer, WEP generation layer and the infrastructure layer (which aids the deployment of computing resources in the grid or cloud) for execution of scientific applications.

***The WS-PGRADE/gUSE*** workflow management system is different from the aforementioned SWMS in that it provides a framework for the rapid prototyping and development of Science Gateways community of practices. It is open source software that is used for research and commercial activities (Kiss et al., 2014). In addition to the presentation, user service, WEP generation and the infrastructure layers, it also supports different aspects of scientific workflows such as parameter sweep and meta-workflows. (For more on this SWMS, see above sub-section 2.5.1 on WS-PGRADE/gUSE). The different SWMS features and components are summarised in Table 2.4 below.

Table 2.4 Scientific Workflow Management System Features and Components

| SWfMS | Structures | Workflow sharing | UI type | Parallelism | Scheduling |
|---|---|---|---|---|---|
| Askalon | DCG | Supported | GUI | Activity | Static & Dynamic |
| Chiron | DCG | Not Supported | Textual | Data & Pipeline & Independent & Hybrid | Dynamic |
| Galaxy | DCG | Supported | GUI (Web Portal) | Independent | Dynamic |
| Kepler | DCG | Not Supported | GUI | Pipeline & Independent | Static & Dynamic |
| Pegasus | DAG | Not Supported | GUI & Textual | Data & Independent | Static & Dynamic |
| Swift | DCG | Not Supported | Textual | Pipeline & Independent | Dynamic |
| Taverna | DAG | Supported | GUI | Data & Pipeline & Independent | Dynamic |
| Triana | DCG | Not Supported | GUI | Data & Pipeline & Independent | Dynamic |
| WS-PGRADE/ gUSE | DAG | Supported | GUI (Web Portal) | Data & Independent & Hybrid | Static & Dynamic |

## *2.7 Major Science Gateway Functionalities*

In this Chapter several projects (such as DCI Portals, grid portlets, SWMS, and Science Gateway projects) which lend themselves to the transparent access of DCI resources are discussed. These technologies are characterised by different attributes and functionalities which are required by the communities that need them. They aid in the easy access to Cloud and Grid resources and enable applications from within web browsers. They also help to hide the details of the underlying infrastructure by exposing only the science-specific parts of the scientific application to be executed in the various DCIs. These solutions which are often presented in the form of web portals, Grid portlets, SWMS and Science Gateways, provide an interface for running and executing scientific jobs on distributed resources. Research communities are often aided by facilitating and delivering DCI services over the web and these functionalities are embedded in web portals, which is one of the possible ways to access remote computing resources being offered by DCI environments.

This section will give a summary of the high level functionalities of the technologies that aid in the transparent access of the services provided by the DCIs as discussed throughout the literature. Table 2.2 captures the commonly used Science Gateway frameworks and their widely adopted functionalities. Table 2.5 will therefore integrate and build on Table 2.2 by highlighting the most important and commomly used functionalities across the different projects which have been discussed throughout the literature. Based on the review of literature, the most important and commonly used functionalities which are required for consuming the services provided by the DCI resources include security management, job management, workflow management and data management. These functionalities which are taken at a high level are summarised across the different projects discussed earlier in this Chapter. These functionalities are presented in Table 2.5 below as follows.

Table 2.5 Summary of the Attributes and Functionalities supported by the different DCI projects discussed in the literature

| DCI Project | Low level Functionalities, User and DCI Requirements | Security Manageme nt | Job Submission Manageme nt | Workflow Manageme nt | Data Manageme nt | Section Number/R ef |
|---|---|---|---|---|---|---|
| PROGRESS Approach Grid portal | Job submission service, data management system | - | Yes | - | Yes | (2.5.3) Bogdanski et al (2007) |
| LEAD Portal | Authorisation framework, Application service factory, Web service based workflow system, data management, query tools, workflow configuration, workflow composition, workflow monitoring. | Yes | - | Yes | Yes | (2.5.3) Christie and Marru (2007) |
| GEONGrid Portal | Data Registration services, data mediation services, data integration services, workflow services, job submission service, job monitoring service, certificate management services. | Yes | Yes | Yes | Yes | (2.5.3) Youn et al (2007) |
| Web-based Science Portal for Earthquake Engineering (OpenSees) | Upload input model file, create job script, edit input model file, specify HPC job parameters, submit large number of jobs, query running job status, store output data | - | Yes | - | Yes | |
| CCLRC (NGS) Portal | X-ray data collection, access to HPC resources, GRAM job submission, MDS resource discovery, GridFTP, batch job monitor, Storage | - | Yes | - | Yes | (2.5.3) Akram et al (2007) |

| | | | | | | |
|---|---|---|---|---|---|---|
| | resource broker (SRB). | | | | | |
| OGSA-DAI portal | Data access, data discovery, data integration, uniform interface to access data. | - | - | - | Yes | |
| Genius Grid Portal | Security service, job submission service, data management service, interactive service. | Yes | Yes | - | Yes | (2.5.2) Barbera et al (2007) |
| NPACI portals | Information services, interactive services-access to HPC resources, submit, monitor, delete jobs, manipulate directories and files, and manage accounts. | Yes | Yes | - | Yes | (2.5.3) Thomas et al (2002) |
| NBCR Portal | Advanced computation, data services, visualisation capabilities | - | Yes | - | Yes | |
| Bayes, Basins and Estauries Portal | Authentication service, job submission, file migration. | Yes | - | - | Yes | |
| Grid Resource Broker Portal (GRB) | Profile manager, credential manager, resource finder, job assistant, job supervisor. | Yes | Yes | - | - | |
| GridSAT | Resource manager, Job manager, client manager, scheduler, checkpoint server. | - | Yes | - | - | (2.5.3) Chrabakh and Wolski (2007) |
| Grid Computing Environments (GCE) portals | File transfer management, job submission and monitoring, data management, user management. | - | Yes | - | Yes | (2.5.3)<br><br>Fox et al (2002) |
| Grid Portal Development | Security, job submission/monitori | | | | | (2.4.1) |

| | | | | | | |
|---|---|---|---|---|---|---|
| kit (GPDK) | ng, file transfer, information services. | Yes | Yes | - | Yes | Novotny, Russell and Wehrens (2004) |
| Open Grid Computing Environments (OGCE) portal | Information services, job and file management services, authentication services. | Yes | Yes | - | Yes | (2.5.3) Alameda et al (2007) |
| Grid Portlets | Single sign-on, resource information provider, job submission, job tracking, workflow model, physical file/replica management. | Yes | Yes | - | Yes | (2.5.3) Zhang, Kelley and Allen (2007) |
| DEISA SGs | Easy access to computing resources, easy way to submit jobs, check job results, and perform post-processing and data analysis, non-sophisticated and intuitively usable GUI, and the creation of input files for complex applications. | - | Yes | - | Yes | (2.5.2.8) Soddemann, 2007 |
| Portals for life sciences | Support for job submission, support for data management. | - | Yes | - | Yes | |
| VisIvo TGviz Gateway | Data management repository, repository for fundamental workflows, job submission, data management, visualisation (to be added) | - | Yes | Yes | Yes | (2.5.2.7) Sciacca *et al.*, 2013 |
| In-VIGO approach | Secure/easy access to resources/applications and data, dynamic provision of execution environments. | - | - | - | Yes | (2.5.2.8) Matsunaga et al. (2007) |

| | | | | | | |
|---|---|---|---|---|---|---|
| TeraGrid Gateway | Easy access to run experiments, generate datasets, perform data reduction and analysis, visualise results, collaborate with remote users and archive long-term data repositories. | - | - | - | Yes | (2.5.2.8) Cobb et al. (2007) |
| HPC – Europa Single Point of Access (SPA) SG | Ease of access to the resources of HPC centres in Europe, provide transparent, uniform, flexible and intuitive user access to HPC, uniform job submission, and security and accounting. | Yes | Yes | - | - | (2.5.3) Oleksiak et al (2007) |
| A Data-Centric Neuroscience Gateway | Unified, secure and easy access to data and meta-data (data discovery, exploration, preparation and processing), automatic and interoperable file transport, automatic provenance information collection and single sign on. | Yes | - | - | Yes | (2.5.2.6) Shahand *et al.*, 2015 |
| Molecular Simulation Grid (MoSGrid SG) | Easy and intuitive user interfaces for jobs, workflow services, data and meta data management. | - | - | Yes | Yes | (2.5.2.8) Gesing *et al.*, 2015 |
| XSEDE/PRACE project | Security mechanisms, distributed data management, workflow services. | Yes | - | Yes | Yes | (2.5.2.8) Gesing *et al.*, 2015 |
| InSilicoLab Framework | - | Yes | Yes | Yes | Yes | Kocot et al., 2014 (2.5.1.4) |
| Catania Science | - | | | | | (2.5.1.1) Barbera, |

| | | | | | | |
|---|---|---|---|---|---|---|
| Gateway Framework (CSGF) | | Yes | Yes | Still being developed to include workflow | Yes | Fargetta and Rotondo, 2011 |
| WS-PGRADE/gUSE | - | Yes | Yes | Yes | Yes | (2.5.1.2) Balasko, Farkas and Kacsuk, 2013 |
| Vine Toolkit | - | Yes | Yes | - | Yes | (2.5.1.3) Russell et al. (2008) |

The above Table 2.5 presents a summary of the most important and commonly used functionalities across the different projects discussed earlier in the literature. These projects aid in the easy access to the services offered by the distributed resources. While several projects such as web portals and Grid portlets have earlier been used to provide access to DCI resources (most especially to Grid utilities), Science Gateway framework on the other-hand is a fairly recent concept used to provide a transparent access across different DCI resources.

According to Table 2.5, each of the aforementioned projects require at least one or more of the major functionalities (as identified throughout the literature) for consuming the services provided by DCI resources. Out of the 29 projects listed in the table, 16 require some form of security management services such as security and accounting mechanisms, single sign-on/authentication services and credential/certificate services. This accounts for more than 50% of the entire project in Table 2.5 and therefore signifies the need for security management functionalities in the framework of services required by DCI projects. Similarly, 22 projects require job management services such as job scheduler, job assistant and supervisor, job tracking and uniform job submission and monitoring. More than 70% of the projects in Table 2.5 require these services which therefore signify the need for job management functionalities in the framework of services required by DCI projects. In addition, 9 projects require workflow management services such as workflow configuration, workflow composition and workflow monitoring. This accounts for about 30% of the entire projects and hence, the need for workflow management functionalities. Finally, 27 projects require data management services such as data and meta-data management, long term data archiving, data registration and mediation, data analysis and file transfer which accounts for

90% of the entire project. Consequently, the need for data management functionalities is therefore paramount in the framework of services required by DCI projects.

Table 2.5 therefore presents the high level functionalities which are utilised for developing Science Gateways for the purpose of consuming the services offered by DCI resources. Consequently, a Science Gateway should support these different functionalities such that, since different DCIs support different types of security mechanisms, Science Gateways should provide support for all the security mechanisms required by the different DCIs. Similarly, different DCIs implement different types of job submission protocol therefore a generic Science Gateway framework should be equipped to handle the different submission protocols. In addition, Science Gateways should provide workflow editing and execution services in order to support more advanced application types. Finally, different DCIs apply different data access and management protocols and the more generic Science Gateway frameworks should be equipped to handle all the protocols. Instances of Science Gateways also known as application-specific Science Gateways are developed from scratch to include one or more of the aforementioned high-level functionalities which may largely depend on the services required by a given community of practice.

The entire Chapter 2 has discussed the technologies which can be used to aid the transparent access of DCIs as well as identify the major Science Gateway functionalities required by different communities of practice. More so, it discusses the different portal technologies and Science Gateway frameworks that may be used for the implementation of these functionalities within Science Gateways. However, throughout the literature, there are no methodologies (which can provide the sequence of steps) to aid developers in the actual implementation of their scientific applications in Science Gateways. The different functionalities discussed in this section will serve as the basis for both the framework of services required by Science Gateways and the proposed methodology which can be utilised for developing these services in Science Gateways which is discussed in Chapter 4.

## *2.8 Open Science Research*

This Chapter discusses the technologies that can aid the easy accessibility and availability of scientific software applications to communities of practice worldwide. However, making use of Science Gateways to execute scientific jobs on DCIs produces results or some form of scientific data which can be further stored for purposes such as analysis, verification, collaboration, reproduction, redistribution and reuse. Ideally, the software, data and results presented in a scientific article should be available for other scientists to use, validate and build upon for their own research (Taylor *et al.*, 2016). Usually these research outputs do not always make it to the public domain. Therefore, the reuse, replication and/or reproduction of a scientific experiment may prove difficult.

Reproducibility refers to the fact that scientific findings are not singular events or historical facts. It is the extent to which consistent results are observed when scientific studies are repeated (Open Science Collaboration, 2012). In definite terms, reproducibility (and relative terms such as repeatability and replicability) refers to whether research finding recur. As such, scientific claims should not gain credence due to the status or authority of their originator but by the replicability of their supporting evidence. According to (Baker, 2016), more than 70% of researchers have tried and failed to reproduce another scientist's experiments, and more than half have failed to reproduce their own experiments.While some of the scientists conceived that there is a significant crises of reproducibility, others believed that the failure to reproduce published results means that the result is probably wrong. However, most insist they still trust the published literature. According to (Pordes *et al.*, 2007), the Open Science Grid (OSG) provides a distributed facility where the consortium members provide guaranteed and opportunistic access to shared computing and storage resources. The innovative aspects of the project are the maintenance and performance of collaborative (shared and common) petascale national facility for several users transferring terabytes of data a day and providing robust and usable resources for different scientific groups.

There are several initiatives promoting Open Science. One of these is the Energising Scientific Endeavour through Science Gateways and e-Infrastructures in Africa (SciGaIA) project. To support Open Science, and to promote Open Science in Africa, Sci-GaIA has created the Open Science Platform (Taylor *et al.*, 2016). Other initiatives include the Center for Open Science (COS) that aims to promote Open Science research by making use of a variety of software tools, workflows and data storage solutions to help researchers manage, archive, discover and share research more openly. The Open Knowledge Foundation (OKF) (a non-profit organisation) is yet another Open Science initiative which advocates open knowledge, open data, transparency and civil participation. These initiatives help to promote

Open Science and ensure that research validation, reproduction, redistribution and reuse can take place.

## *2.9 Summary*

In this Chapter, a review of the literature of different technologies and projects which were developed to aid the transparent access of DCIs was conducted. This ranges from API’s, Web portals, Portal framework, Gridports, Science Gateway framework to the commonly used SWMSs. This Chaper starts by discussing the various distributed resources and their role in the support and execution of scientific applications. The different application programming interfaces which are used to abstract the different middleware implementation and the underlying infrastructure resources are also discussed. Furthermore, the different approaches for developing Science Gateways (such as building portlets from scratch or customising an existing Science Gateway framework) are discussed. Consequently, some of the most widely used portal frameworks are presented. Developers may choose to build their gateways from scratch (using portal frameworks) especially if the services are limited and the gateway is trying to support the usage of a particular DCI. More so, Science Gateway frameworks which are used to customise instances of Science Gateways for specific communities of practice are discussed. These Science Gateway frameworks which include GridPort, P-Grade, VinetoolKit, WS-PGRADE/gUSE and CSGF enable developers to build application specific Science Gateways according to the different needs of user communities. In doing so, some or the full power of an underlying Science Gateway framework can be exploited and hidden behind an intuitive application-specific user interface. This process ensures that developers who want to build such application specific gateway can achieve that in a faster and easier way and thus help to save cost, time and manpower effort.

Furthermore, more than just simple job submissions, applications that solve complex problems such as scientific simulations require the creation and execution of scientific workflows. To this end, the different SWMSs are briefly analysed. This Chapter summarises with the high-level services (such as security management, job management, workflow management and data management) which generic Science Gateways should possess based on the functionalities and attributes of the various technologies that are discussed. This Chapter concludes with a brief discussion about open research and some major initiatives that support Open Science.

# Chapter 3: RESEARCH METHODOLOGY

## *3.1 Overview*

Chapter 3 discusses the philosophy behind the research and also justifies the research methodology that was adopted throughout the study. The rest of this Chapter is organised as follows. Section 3.2 discusses the concept of research in general - the different research perspectives and the different philosophical assumptions that underpin the research perspectives. In Section 3.3 the nature of the research was established and (by matching the research questions with the research objectives) three fundamentally different research types (Exploratory, Design and Development, and Reflective Evaluation) were utilised. Section 3.4 presents the adopted research methodology, the guidelines and outputs of the DSR as well as justify the use of DSR approach throughout the section. The distinct nature of the research types which is also captured in the DSR suggests the adoption and use of different methodologies/approaches for achieving each research type. Section 3.5 therefore presents the adopted methodology which was captured by the DSR and subsequently used to achieve the three identified research types.

## *3.2 Research Approaches*

A research is defined as an activity that helps in contributing to the understanding of a phenomenon (Kuhn, 1996). This phenomenon is often conceived to be a collection of behaviours which are specific to a given set of entities and which a researcher or a research community has found to be interesting. Consequently, a research methodology then presents the best applicable practices to specific issues and outlines the best ways that research could be undertaken (Howell, 2012). Due to the considerable number of diverse communities that contribute to Information System (IS) knowledge base such as engineering, psychology and natural sciences, the area of Information System is dominated by multidisciplinary field (Purao, 2002; Gregor, 2002). (Mingers, 2001; Robey, 1996) therefore investigated the issues concerning diversity in IS such as the merits and demerits of adopting many different research traditions. Consequently, the adoption of a suitable research approach then becomes paramount to a successful research of this nature.

### 3.2.1 Research Perspectives

According to (Stolterman, 2008; Fallman, 2003), there are four different types of research paradigms, namely positivist, interpretive, critical and design science. However, DSR according to (Vaishnavi, V. and Kuechler, W., 2015) is yet another set of synthetic and analytical techniques and perspectives which compliments the positivist and interpretive perspectives for performing research. In order to have a thorough understanding of the different research perspectives, it is important to outline and analyse these perspectives from different point of view. This is briefly discussed as follows:

**Positivist**: This is the prediction of human behaviour by making use of the principles of natural sciences. Science, according to this view, is an objective, logical and systematic method of analysis of phenomena in order to enable the accumulation of reliable knowledge (Nuryatno, 2003). Positivist studies are therefore used primarily to test theories in a bid to increase the productive understanding of the phenomena being studied. (Orlikowski and Baroudi, 1991)

**Interpretive**: This is the principle of understanding the human behaviour based on interpreting the lived experience. In contrast to the dualist/objectivist of the positivist approach, Interpretive evolves transactional/subjective relations. These kinds of studies assume that people create their subjective meanings while they interact with their world.

**Critical**: This approach is based on the transformation of the condition of the humanity of people. It assumes that reality is socially constructed. These studies aim to critique the status quo through structural contradictions with social systems and in the process, restrictive social conditions are said to be alienated.

**Design science**: This approach (also known as the science of the artificial) is a body of knowledge about the design of artificial (man-made) objects and phenomena/artefacts designed to meet certain desired goals. DSR has been contrasted with the natural science research. This was done by (Simon, 1996) in his book “The Sciences of the Artificial” where he defined natural science as the body of knowledge about some class of things and objects of phenomena in the world which describes how they behave and interact with each other. Design science, on the other hand, is a body of knowledge about the design of artificial or man-made objects and phenomena/artefacts which is designed to meet certain desired goals.

These types of research are said to be the two predominant types. This is consistent with the ontological beliefs (the different sets of philosophical assumptions are discussed in the next section) where the phenomena under investigation is either assumed to be objective and therefore independent of humans (natural), or they are subjective and therefore depends on the actions of humans in creating and recreating them (artificial/man-made). It is a widely held view that IS research is largely based on the natural science approach however, (Carlsson, 2006) argued that (in order to advance the field) such research should be complemented with the knowledge that is based on the design science approach.

### 3.2.2 Philosophical assumptions for the different research perspectives

To fully understand the basic assumptions and postulations of the different research perspectives, three fundamental questions must be addressed by the paradigm of science. These are ontological, epistemological and methodological. A fourth assumption, the axiological philosophical assumption, was discussed by (Vaishnavi, V. and Kuechler, W., 2015). The classification of these assumptions constitutes the philosophical stances that researchers may adopt when doing their work. These philosophical assumptions are defined as follows:

**Ontology** (beliefs about physical and social reality): These beliefs are associated with the phenomena under investigation by ascertaining whether the empirical world is assumed to be objective (i.e. independent of humans) or subjective (by depending on the action of humans in creating it). As such, it is the study that simply describes the nature of reality.

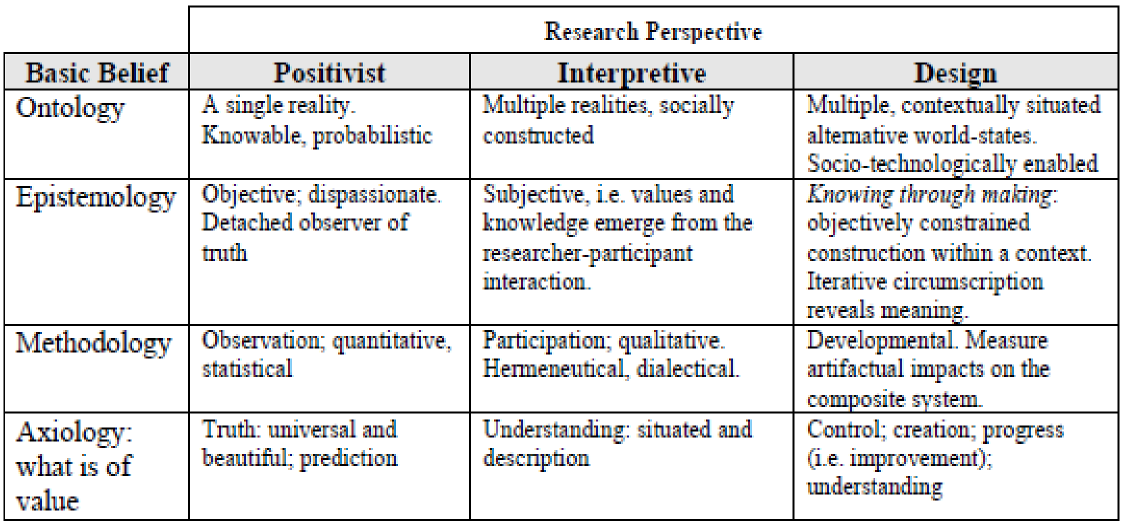
**Epistemology** (beliefs about knowledge): This study explores the nature of knowledge by simply ascertaining what knowledge depends on, and how researchers can be sure of what they know. It therefore entails the criteria for the construction and evaluation of knowledge.

**Methodologies**: This indicates which research methods are appropriate for the generation of valid empirical evidence. It simply investigates the possible ways or approaches that researchers may choose to obtain knowledge.

**Axiology** (study of values): This study defines the value that is associated with a given subject and their corresponding effects on the conduct of research. These values could be held by an individual or groups of individuals and the reason behind such values.

The different types of philosophical assumptions as they relate to all four research perspectives (positivist, interpretive, critical and design) are therefore illustrated in Table 3.1 below.

Table 3. 1 Philosophical Assumption of Research Perspectives. (Vaishnavi, V. and Kuechler, W., 2015)



| | Research Perspective | | |
|---|---|---|---|
| **Basic Belief** | **Positivist** | **Interpretive** | **Design** |
| Ontology | A single reality. Knowable, probabilistic | Multiple realities, socially constructed | Multiple, contextually situated alternative world-states. Socio-technologically enabled |
| Epistemology | Objective; dispassionate. Detached observer of truth | Subjective, i.e. values and knowledge emerge from the researcher-participant interaction. | *Knowing through making*: objectively constrained construction within a context. Iterative circumscription reveals meaning. |
| Methodology | Observation; quantitative, statistical | Participation; qualitative. Hermeneutical, dialectical. | Developmental. Measure artifactual impacts on the composite system. |
| Axiology: what is of value | Truth: universal and beautiful; prediction | Understanding: situated and description | Control; creation; progress (i.e. improvement); understanding |

The authors compared and contrasted the DSR and the positivist ontology where a single, given composite system forms the unit of analysis. For DSR, even the problem statement is subject to revision as the research effort proceeds. However, even though design science researchers are comfortable with alternative world states, most of them believe in a single, stable underlying physical reality that constrains the multiplicity of the world-states. This is therefore consistent with the positivist research paradigm and the ontology philosophical assumption.

In light of the discussion thus far and in the context of our research, the DSR which is underpinned by a shift from positivist research is the most suitable approach that is chosen for this research. Initially, from the positivist-ontology point of view, this research aim is to explore a new approach to developing scientific software applications in Science Gateways. This is considered as a single reality in this world and is therefore consistent with the positivist philosophy. Furthermore, new approach to developing scientific software applications in Science Gateways will be explored based on observing existing scientific facts which, using subsequent experiments, could then be converted into knowledge. This is also consistent with the positivist-epistemology. In addition, a comparative study is performed on the effectiveness of the proposed MESSAGE methodology which is, once more, compatible with the positivist-methodology. Finally, a thorough evaluation of the system is performed to predict the truth about its findings which are captured in the positivist-axiology.

## *3.3 Nature of the Research*

To determine the nature of this research and subsequently the research type(s) that are appropriate for addressing it, the research objectives as well as the research questions that were identified in Chapter 1 are re-visited. Consequently, attempt is made to match the research questions with the reseach objectives. It is believed that this approach will give more clarity and will help to identify the adopted research type(s) for this study. The aim of this research is to create a methodology for developing scientific software applications in Science Gateways. This (as anticipated) is beneficial to different communities of practice to develop their scientific applications and to ensure that potential users can adopt the DCI technology without being deterred by the complexities of the underlying infrastructures.

As mentioned earlier, the research questions are matched to correspond to the different research objectives that were identified in Chapter 1. As such, correlations made between research questions and research objectives are highlighted below as follows:

1) RQ1 corresponds to the first identified research objective.

2) RQ2 corresponds to the second and third identified research objectives.

3) RQ3 corresponds to the fourth and fifth identified research objectives.

The above correspondence between research questions and research objectives shows that all three research questions are fundamentally different in nature. As such, it is evident that to address each of the research questions, different research types must be employed.

Exploratory research is used to investigate a problem that has not been clearly defined (Shields and Rangarajan, 2013). Since the first question seeks the best approach for developing scientific software applications in Science Gateways, this type of research is ideal for RQ1 in order to explore alternative approach for developing scientific software applications in Science Gateways. This exploration is done based on the related literatures on the different technologies and approaches that can enable the easy access to distributed resources and thus help to achieve the first research objective.

Similarly, a relative research type is adopted to answer the second research question (RQ2) which will consequently help to achieve our second and third research objectives. Since both objectives involve processes which will lead to the development/actual implementation of the artefact (i.e. the Infection Model portlet), there should be a different methodology for this phase. There are a number of ways (in the form of software development approaches) through which the design and development of software artefacts can be achieved. The different approaches to software development and the adopted approach for this research are briefly discussed and justified in Section 3.5.

Lastly, reflective assessment which is used for verification purposes will be used to answer the last research question (RQ3). This question deals with the effectiveness of the proposed MESSAGE methodology as well as the developed software artefacts. As a consequence, this research type is suitable for conducting this research question and thus help to achieve the last research objectives. In addition, empirical research methods which can also be used for hypothesis generally starts with a given theory where the researcher continuously tests the theory or develops it by practice in the real life scenario (Davis, Eisenhardt and Bingham, 2007). According to Yin (2009), a case study is "an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident". A case study will therefore be used in this study to verify how effective the proposed MESSAGE methodology is.

## *3.4 Research Methodology*

In the previous section, three different research types were identified based on matching the research questions and the research objectives in Chapter 1. The consequence of having three distinctly different research types is that a research methodology that is capable of incorporating all the identified research types must be adopted. DSR methodology which gives a detailed description of how a typical DSR effort should proceed and consists of different stages (such as awareness of the problem, suggestion, development, evaluation, and conclusion) seems to be the most suitable approach for this type of research. As a result, a description of how DSR (which captures the three distinctly different research types) is employed for this research is discussed in the following section.

### 3.4.1 Design Science Research Methodology

DSR was earlier defined as a research that is performed simply by using design as a research method or technique. However, (Hevner and Chatterjee, 2010) gave a more detailed definition of DSR which they described as a technology-oriented, problem-solving approach that seeks to design artefacts which can help in defining ideas, practices, and products and which can serve as a basis for analysing, designing, implementing, and using information systems in an effective and efficient manner. The primary objective of DSR is to design innovations (artefacts) for information systems research and these artefacts usually form the basis of the outputs/outcomes of DSR. Furthermore, the outputs of design science is classified into seven (7) distinct forms of constructs, models, framework, architectures, design principles, methods and instantiations. These different outputs are explained in the next section.

#### 3.4.1.1 Outputs of Design Science Research

As already mentioned in the previous section, the output of DSR is the development of software innovations (artefact). These artefacts can take several forms of outputs such as constructs, models, framework, architectures, design principles, methods, and instantiations each of which represents design science knowledge (Vaishnavi, V. and Kuechler, W., 2015). The classification of DSR output as given by (Purao, 2002) and built upon by (March and Smith, 1995) are given below as:

**Constructs**: They are the conceptual vocabulary of problem/solution domain. Constructs are refined throughout the DSR cycle and are used during the conceptualisation of the problem.

**Model**: This is a set of statements which are used to express the relationships between constructs. It uses the construct to represent a real world simulation (such as a defined problem and its solution).

**Method**: This is made up of a set of steps or guidelines for performing a task. In order for a solution statement model to be realized, methods serve as a goal directed plans for manipulating constructs.

**Instantiations**: This is the implementation of an artefact with the use of constructs, model and methods. While other types of artefacts are referred to as abstract artefacts, Instantiation, on the other hand, is referred to as material artefacts. However, a design theory usually consists of both the abstract artefacts as well as instantiations (material) artefacts. The research output is therefore expected to take this very form. Table 3.2 summarises the different DSR outputs based on the expected outputs of this research.

Table 3.2 Design Science Research Outputs

| | **Output** | **Description** |
|---|---|---|
| **1** | Constructs | The conceptual vocabulary of a domain |
| **2** | Models | A set of propositions or statements expressing relationships between constructs |
| **3** | Methods | A set of steps used to perform a task – how-to knowledge |
| **4** | Instantiations | The operationalization of constructs, models and methods and other abstract artefacts. |

March and Smith (1995) built on the above by adding a fifth research output. This research output is otherwise known as better theories-design theories. Even though the final output of this research effort is in the form of an instantiation, it is imperative for the nature of this research to also indicate several other DSR outputs produced in the course of developing the final artefact. These outputs are briefly discussed in the following section.

### 3.4.1.2 Relating the DSR outputs to the final research output

***Instantiations*:** The most suitable output which may be used to describe our DSR output is seen in the form of an instantiation of the software artefact. As already discussed, Instantiations make use of constructs, model and methods in the implementation and the realisation of the artefacts in an environment. As a result, the nature of our research can also indicate the existence of several other DSR outputs as follows:

**Constructs:** This research attempts to explore new approaches to executing scientific software applications on worldwide infrastructures and to ultimately create a methodology for developing scientific software application in Science Gateways. It is intended to be of immense benefit to different communities of practice. The third research question (RQ3) deals with measuring how effective the proposed methodology is. This will project some metrics that will help measure whether the proposed methodology has fulfilled the intended purpose and therefore establish the constructs of the artefact.

**Model:** Based on the construct that was defined above, another output of this artefact research is seen in the conceptual model that can be used to capture the relationships within the components of the artefact for future research efforts. This may include the use of the proposed methodology in developing scientific software applications for the different scientific domain by adopting other Science Gateway frameworks.

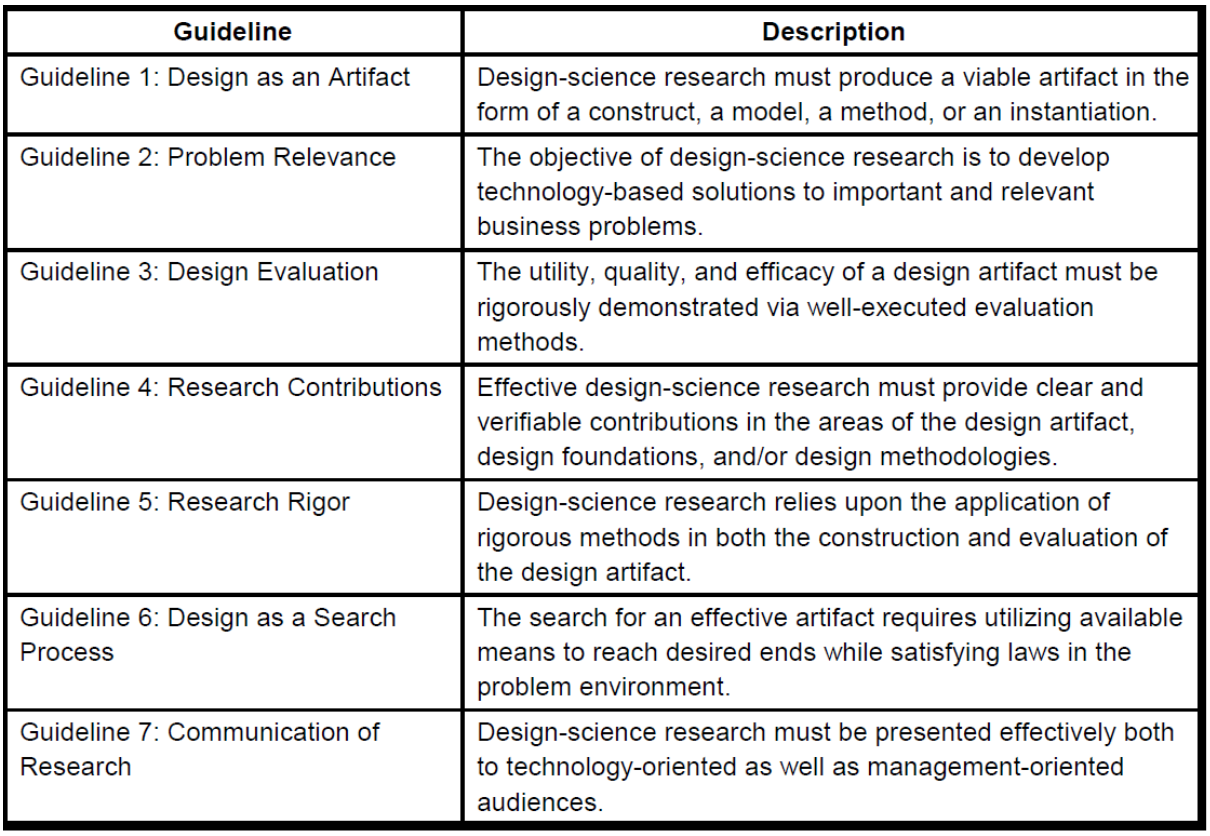
**Methods:** This is made up of a set of steps or guidelines employed in answering the second research question (RQ2). These steps and guidelines are explained in great detail in Section 3.5.

Furthermore, in order to understand the different forms that knowledge could take, it is good to start by understanding the possible types of knowledge contribution of DSR. These include Invention (inventing new solutions for new problems), Improvement (developing new solutions for known problems), Adaptation (innovative adaptation of known solutions to new problems) and Routine design (applying known solutions to known problems). Routine design is said to be hardly considered as a significant research contribution and for a knowledge contribution to be considered to be significant, it must be significant within the current state of the knowledge in the research area.

### 3.4.2 Design Science Research Cycle

In order to achieve the desired research outputs and, consequently, the required artefacts, adherence to well-specified processes is paramount. These set of processes serve as the basis for a successful design and construction of an artefact and may involve the actual sequence of events used in constructing an artefact (building) and ways of determining how well the artefact performs (evaluation) (March and Smith, 1995). Nonetheless, the building and evaluating processes are not very clearly stated. As a consequence, (Von Alan *et al.*, 2004) further highlights the seven guidelines that are crucial to the successful building and evaluation of artefacts. These guidelines are summarised in table 3.3.

Table 3. 3 Guidelines for Research Science Design



| Guideline | Description |
|---|---|
| Guideline 1: Design as an Artifact | Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation. |
| Guideline 2: Problem Relevance | The objective of design-science research is to develop technology-based solutions to important and relevant business problems. |
| Guideline 3: Design Evaluation | The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods. |
| Guideline 4: Research Contributions | Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies. |
| Guideline 5: Research Rigor | Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact. |
| Guideline 6: Design as a Search Process | The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment. |
| Guideline 7: Communication of Research | Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences. |

Furthermore, (Vaishnavi, V. and Kuechler, W., 2015) described a computable design process model that should be followed which further builds on the processes that were suggested by (March and Smith, 1995) and (Von Alan *et al.*, 2004). This process is otherwise known as the DSR cycle which is seen in Figure 3.1. This constitutes the different stages that a design science researcher should go through in order to contribute new and true knowledge as this needs to be the key focus of DSR.

According to Figure 3.1, the process steps for a DSR consist of five different stages of Awareness of problem, Suggestion, Development, Evaluation, and Conclusion. Each of the stages has a corresponding output in the form of proposal, tentative design, artefact, performance measures, and results. The Awareness of the problem stage is where an interesting research problem is identified either from multiple sources or new developments in the industries. The corresponding output of this phase, as evident from the diagram, is a proposal. The Suggestion phase, on the other hand, is the stage where new system functionalities/requirements are proposed and defined. The output here, along with a proposal, is the Tentative design of the artefact. In addition, further development and implementation of the Tentative design is done at the Development phase, and the output here is the desired artefact. The Developmental phase is followed by a thorough evaluation of the developed artefact which is based on the criteria that were made explicit in the Awareness of the problem/ Proposal phase. The output of this phase is in the form of the different set of measures that should be carried out on the developed artefact. The final phase of the DSR cycle/process is the Conclusion phase where the results of the efforts are consolidated based on the behaviour of the artefact and the new knowledge is categorised as either facts or used as the basis for further research. The output here is the Results obtained from evaluating the artefact.

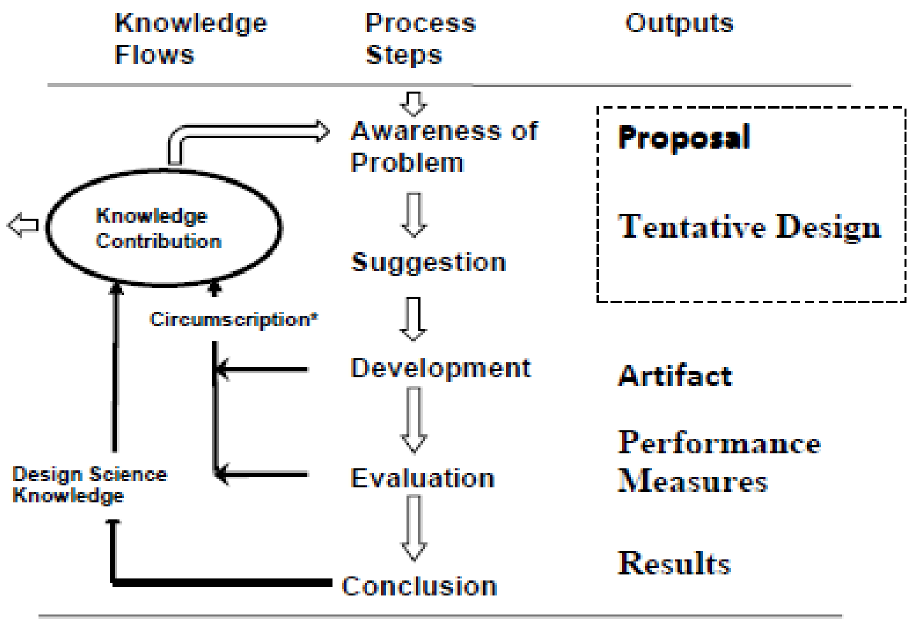


Figure 3. 1 Stages in DSR methodology by (Vaishnavi, V. and Kuechler, W., 2015)

In order to determine how relative our research approach is, attempt has been made to match the identified research types (Exploratory, Design and Development, and Reflective Assessment) with the seven guidelines for constructing an artefact, by (Von Alan *et al.*, 2004), and the DSR cycle/process, by (Vaishnavi, V. and Kuechler, W., 2015). Table 3.4 below therefore shows how our research types correspond with the DSR cycle, DSR outputs and the seven guidelines for constructing artefacts which have been discussed above. This will therefore justify (and culminate into) the adoption of the DSR methodology.

Table 3. 4 The Correspondence between the Research Guidelines, DSR cycle, DSR outputs and the identified research types

| **Research Guidelines** | **DSR Cycle** | **DSR Outputs** | **Identified Research Types** |
|---|---|---|---|
| 1.Problem Relevance | Awareness of problem | Constructs | Exploratory |
| 2. Design as a search process | Suggestion | Models<br><br>Methods | Exploratory |
| 3 -Research Rigor<br><br>-Design as an Artifact | Development | Instantiation | Design and Development |
| 4 – Research Rigor<br><br>-Design Evaluation | Evaluation | Design theories/better theories | Reflective Evaluation |
| 5–Research Contributions<br><br>-Communication of Research | Conclusion | Design theories/Better theories | - |

### 3.4.3 Rationale for choosing Design Science Research Methodology

DSR is defined as the learning through building (creation) of an artefact. A careful study reveals that the DSR cycle maps well with the three different research types (Exploratory, Design and Development, and Reflective Evaluation) as well as the seven guidelines for constructing an artefact which was suggested by (Von Alan, March et al. 2004) (see Table 3.4). As such, DSR appears to be the most suitable research methodology that may be used to address all three different research types. Furthermore, according to (Purao 2002), and in addition to the artefacts, design research offers two more types of research contributions such as reproducible knowledge (a novel artefact which consist of the different DSR outputs that is used to further improve the existing knowledge base) and methodologies and theories (which present ways to support the phenomena of interest based on the development and use of the novel artefact). As a result, DSR is the implementation of artefacts which could well be used to improve theories and thus serve as a significant research contribution. The different stages of the DSR and how it maps with our three research types as well as the seven guidelines for constructing an artefact is seen as follows:

In this research, the awareness of the problem phase (Proposal) involves the need to create methodologies for developing scientific software applications and execute jobs in a distributed environment, particularly, via a Science Gateway interface. As a consequence, the major problem that is identified includes the non-availability of appropriate methods and procedures for developing the major Science Gateway functionalities discussed in Section 2.7 of Chapter 2. In identifying this problem, this research made use of the Exploratory research study (which is our first research type) and also employed the second guideline (the research relevance) of the seven guidelines for constructing an artefact.

The suggestion phase (Proposal and Tentative design) relates to finding ways in which scientific software applications are currently being developed and suggests improvements based on the findings. Similar to the Awareness of the problem phase, our Exploratory research study still applies in this phase and the sixth guideline (Design as a search process) for constructing an artefact was employed to tackle this problem.

The development phase (Artefact) relates to the best possible ways for constructing the actual artefact and, as such, the first guideline (Design as an Artefact) and the fifth guideline (Research Rigor) were utilised. This also corresponds to the second research type (design and development) which was earlier identified.

The evaluation phase (Performance measures) relates to the different measures for evaluating our designed artefact based on the criteria which were set out in the proposal phase. According to (Von Alan, March et al. 2004), evaluation is carried out based on metrics such as accuracy, reliability, performance, completeness, functionality, usability, and consistency. In addition, based on the guidelines for constructing an artefact, the third guideline (Design evaluation) and the fifth (Research rigor) have been employed. This also corresponds to the third research type (Reflective Assessment) which was earlier identified.

The final phase of our research, the conclusion phase (Results), is the stage where the results of the first four processes of realising the artefact are discussed and generalisations are made. Based on the seven guidelines for constructing artefacts, the fourth (Research contributions) and the seventh (Communication of research) guidelines was employed, which also aligns with the third research type.

## *3.5 Methodology for the different aspects of the Design Science Research*

We have already established three distinctly different research types based on the research questions and the research objectives which are used in the course of this study. These are Exploratory, Design and Development, and Reflective Evaluation studies. We have also established a research method, known as the DSR methodology, which is employed to address all the different types of research types. This methodology is also supported by the seven guidelines that were given by (Von Alan *et al.*, 2004) to be utilised for the successful building and evaluation of artefacts and therefore provides a strong rationale for its adoption. As a consequence, the methodology that is adopted for the different stages of DSR and thus the different research types is described in the following subsections.

### 3.5.1 Awareness of Problem and Suggestion (Exploration)

As earlier discussed the output of the awareness of the problem phase which will form the basis of a new research effort, is a formal or informal proposal. For the course of this study, a proposal for the different attributes and functionalities (in the form of services) that Science Gateways should possess was produced after a thorough review of the literature was carried out in Chapter 2. This produced an awareness of the problem by presenting the essential Science Gateway services and the need to have a systematic approach of developing these services in Science Gateways. Suggestions based on the identified problems come in the form of a tentative design. The design and implementation of the different case studies will therefore be detailed in subsequent chapters of this thesis.

### 3.5.2 Software Development (artefact design and development)

The process of software development constitutes one of the most important aspects of the entire research since it involves the sequence of steps that will lead to the realisation of the actual artefact. As such, it is important to choose a suitable software development methodology, otherwise known as software development lifecycle (SDLC) models or software process models. According to (Boehm, 1988), SDLC models are important primarily because they provide the necessary guidance on the order in which a software project should perform the required tasks so as not to pursue the development and evaluation phases in the wrong order. This guidance is usually seen in the form of order, phases, increments, prototypes and validation tasks which an artefact should follow to achieve the desired output. Consequently, the choice of software methodology for developing specific software artefact is of utmost importance.

In this section, a brief overview of the different SDLC models which may be used for the realisation of a software artefact is presented. Furthermore, the choice of software methodology which has been adopted for this research and the reason behind its selection is justified. An overview of the SDLC models which was presented in the work of (Ruparelia, 2010) is seen in Table 3.5 as follows:

Table 3. 5 Software Development Methodologies

| **Methodology** | **Merits** | **De-merits** |
|---|---|---|
| 1. Spiral | An improvement on the Waterfall model | Subjective risk assessment |
| 2. Rapid Prototyping | Geared towards user satisfaction | Limited testing is required |
| 3. Waterfall | It is very well documented | It lacks flexibility |
| 4. Object-Oriented Programming | Can easily adapt to change | Absence of object organization |
| 5. Code and Fix | Mostly adapted for small software | Code is really expensive to maintain |
| 6. Incremental Delivery | Critical requirements are satisfied for prompt use | Little or no user feedback |

SDLC, which is generally used to describe the steps that must be followed within the lifecycle process, is a conceptual framework that helps in the structure of the stages involved in the development of an application from its initial feasibility study through to its deployment. As shown in Table 3.5, there are different types of SDLC models currently in use. However, the SDLC models which can be used for the development of the artefact such as waterfall approach, and to a lesser extent the Rapid Application Development (RAD) and Spiral approach are discussed in a bit more detail. (Check appendix A for the discussion of the different software development methodologies).

The very nature of this research makes the waterfall approach the ideal SDLC model to adopt for the development of the artefact. Firstly, end-users are not required to test the developed artefacts, therefore the much needed flexibility in adapting to user requirements, which is often seen in other SDLC models such as RAD, is not of utmost importance. Secondly, the waterfall approach is well suited for this research because the user requirements which were defined at the early stages of the research remains consistent throughout the research process and, since the aim of the research is to explore new approaches to developing scientific software applications in Science Gateways, the ultimate aim, therefore, is not the artefact but the proposed methodology. However, the artefacts are used to implement different aspects of the proposed methodology.

### 3.5.3 Evaluation and Conclusion (Reflective Evaluation)

In order to address the evaluation and conclusion stage and thus evaluate the proposed methodology for developing Science Gateways for communities of practice, a Reflecive Assessment as well as an empirical enquiry (in the form of a case-study) which investigates a contemporary phenomenon within its real life context is employed (Demeyer, 2011). This is usually employed when the boundaries between the phenomenon and the context are not clearly evident. Also, since we have employed the design science approach which is the learning through building of an artefact, our hypothesis (it is feasible to create methodology for developing scientific software applications in science Gateways) has to be empirically studied. According to (Runeson and Höst, 2009), an empirical research tool is the case study therefore, the proposed methodology is implemented by using real life scenarios to test and evaluate ideas and theories.

## *3.6 Summary*

This Chapter discusses and justifies the research methodology that was adopted throughout this research. It starts by discussing the different research perspectives and philosophical assumptions and how they relate to the study at hand. Furthermore, the nature of the research was established and, by matching the research questions with the research objectives, three fundamentally different research types (Exploratory, Design and Development, and Reflective Assessment) were identified. In addition, it justifies the adoption of the DSR and proposed the approach that is used to achieve the research types and, consequently, the different aspects of the DSR.

# Chapter 4: MESSAGE METHODOLOGY

## *4.1 Overview*

While the previous Chapter discusses the philosophy behind the research and the adopted research methodology, this Chapter presents and analyses the essential services that a Science Gateway should possess as well as propose a MESSAGE methodology that may be used to develop the services in Science Gateways. The following Section 4.2 presents the different Science Gateway services which are required for developing an application in a Science Gateway for particular communities of practices (as identified in the review of literature). Furthermore, these Science Gateway services are investigated using an exemplar Science Gateway framework known as CSGF, which is discussed in Section 4.3. Section 4.4 presents the proposed MESSAGE methodology for developing scientific software applications in Science Gateways and the summary of the entire chapter is discussed in Section 4.5.

## *4.2 Generic Framework for Science Gateway Services*

Chapter 2 identified four high-level functionalities/attributes, the approaches used for developing and deploying these functionalities and established that the easy access to distributed resources, specifically for the execution of scientific applications on distributed resources, is the most important Science Gateway requirements.

This chapter proposes a MESSAGE methodology for developing scientific software applications in Science Gateways. The proposed MESSAGE methodology will help to create services in Science Gateway to be utilised by different user communities for performing activities such as job submission, job monitoring, job retrieval, workflow execution, data management, etc. To execute any kind of job on DCIs, users can simply utilise the Science Gateway interface to perform the aforementioned activities which will help in ensuring that users can focus solely on performing experiments without having to worry about the details and complexities of the underlying infrastructures and services.

Science Gateways can help to deliver all the high-level functionalities and attributes which are required by Science Gateways to execute and manage jobs for different scientific communities within a distributed environment. Figure 4.1 consists of three main parts which shows the components necessary to execute jobs in DCIs. These include:

- Graphical User Interface
- SG Service Layer
- DCI Resources Layer (such as Cloud, Grid, HPC, etc.)

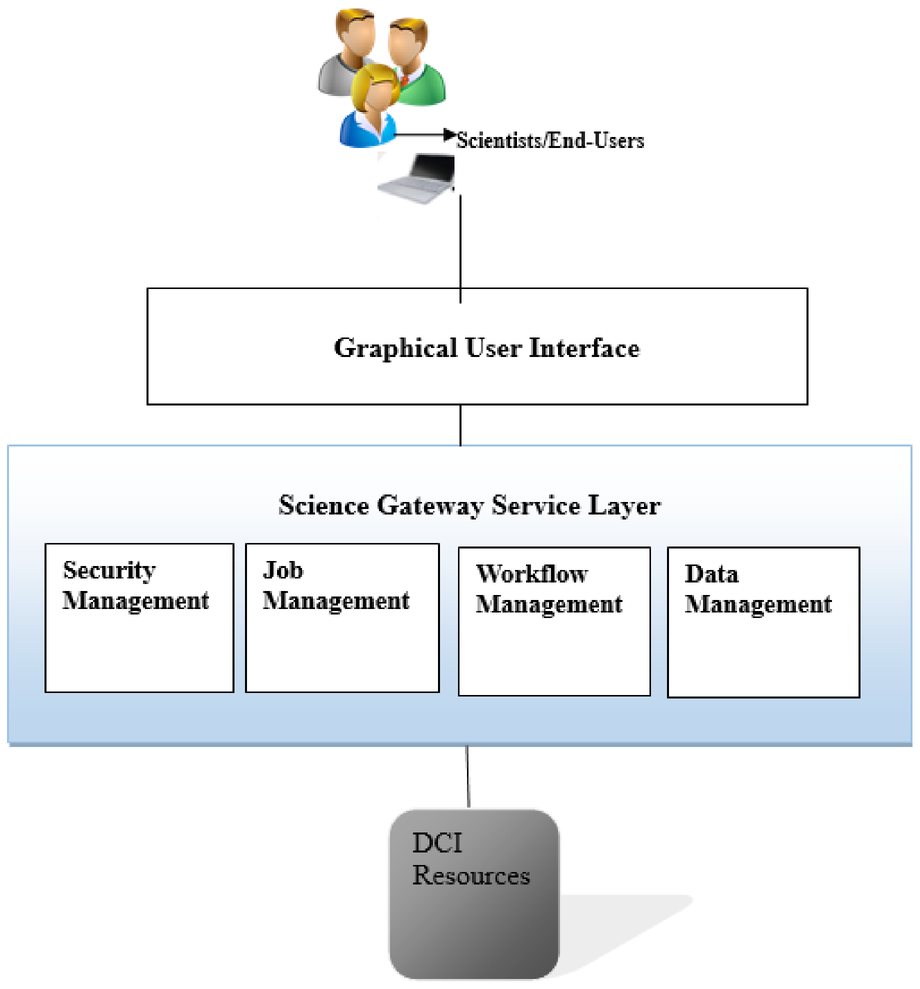


Figure 4. 1 Generic Framework for Science Gateway Services

In order to execute a scientific application in a distributed environment, end-users use a front-end application with a graphical user interface. Consequently, users do not require to understand the underlying infrastructure and framework of services but can focus solely on the specifics of their own scientific experiment.

***The Graphical user interface Layer*** presents a thin client which scientists or end-users could use to interact with the required Science Gateway service in order to carry out the desired scientific tasks. This is usually in the form of web interfaces on a workstation (such as

Desktop/laptops/mobile apps) of a potential Science Gateway user.

***The Science Gateway Service Layer*** (which consists of the different services which may be implemented by the proposed MESSAGE methodology) is made up of the different high-level functionalities and attributes of Science Gateways which are obtained from the review of the literature. These include the security management, job management, workflow management and data management.

***The DCI Resources Layer*** consists of distributed resources such as Grid and Cloud and any other resources in which a particular scientific application has been configured to run. This type of configuration is normally done at the Science Gateway level where a developer can specify the resources on which any given applications could be deployed.

The following section will discuss in detail the different Science Gateway services as mentioned in the Science Gateway service layer. The Science Gateway service layer which consists of the major functionalities that Science Gateways should possess include:

### 4.2.1 Security Management

One of the basic requirements of Science Gateways which was described throughout the literature and incorporated as one of the major services of Science Gateways is security management. Security service is of major importance as they have emerged as mediums through which large numbers of user communities can easily and securely access resources at a number of organisations. More so, as some of these resources may be utilised via a Science Gateway interface to process data for communities of practice, especially where the data is of sensitive nature, securing access to the gateway services, the infrastructures and the data accessed by them becomes paramount. These security activities are often carried out to prevent resource usage and leakage of vital information to unauthorised users and are usually implemented in four different forms of authentication, authorisation, auditing and accounting (AAAA). All these aforementioned security mechanisms are jointly used to define how gateways are accessed, how access to each service is minimised, how the use of each service is accounted for, and how each of the Science Gateway components are made secure.

Authentication mechanism is a service that is used to verify that users are actually who they claim to be (i.e. a process of verifying the identity of a user or process). This service usually precedes the other aforementioned security mechanisms as it serves as the basis for establishing user authorisation privileges as well as the subsequent auditing and accounting that ensues. Usually, users first register via a site which then creates their credentials/accounts (using the combination of user-name or email and a password) and subsequently collects their vital information. A user is then able to perform subsequent log-on to the Science Gateway using the credentials provided during registration. A user may log-on to the Science Gateway after a successful registration process has taken place. There are other means of providing user authentication such as single sign-on where external identity providers may be used to identify users.

Authorisation mechanism is a service that gives permission to specific resources and services. Once a user has successfully log-on, they still need to acquire permission to access specific applications and services which are required to perform experiments in distributed environments. User defined roles and privileges are usually stored in credential services such as Lightweight Directory Access Protocol (LDAP), Virtual Organisation Membership Service (VOMS), Central Authentication Service (CAS). Once the identity of the user has successfully been verified, user roles are then mapped to those performing DCI transactions using the aforementioned credential services. Consequently, access to resources (including data and metadata) are controlled based on user access rights and the groups/communities which they belong.

Auditing mechanism is a service that records and keeps all user activities for future reference. In the event where the resource owner observes the use of the Science Gateway for malicious activities, investigation must take place thereafter, to determine its cause and the appropriate action that must be taken to prevent future occurrences. A prior record of all the user activities will therefore enable such investigations to take place. The auditing mechanisms will also ensure compliant with the European Infrastructure VO Portal and DCI Security Traceability and Logging policies.

Accounting mechanism (which is less important than the other aforementioned security mechanisms is frequently neglected and not usually supported) is a service that enforces resource usage quotas, cycles, bandwidth, and other use policies. Communities usually have agreements in place with the resource provider which normally grants community allocations for the required services. Communities may desire to divide their allocation among its members and, to achieve this, the community needs to know how much of the allocation each of the members has used. Using a Science Gateway in this way for accounting can ensure that resources communicate the total consumption of each request on its completion back to the Science Gateway which then makes it a natural place to keep track of each user's total resource usage.

### 4.2.2 Job Management

Job management is one of the most important services that should be provided by a Science Gateway. It may involve several aspects of job management such as submission, processing, monitoring and the collection of final results/outputs. Science Gateways are used to facilitate access to DCI resources to handle large-scale computations (usually performed in jobs) for scientific applications.

Science Gateways provide facilities for the submission, monitoring and the output retrieval (for both sequential and parallel jobs) to different DCIs. Such facilities can be equipped to provide support for different DCIs as opposed to just having gateways which are tightly bound to specific or small number of DCIs. Furthermore, different DCI's implement different types of job submission protocols and Science Gateways should be equipped to handle them all. Consequently, Science Gateways are now equipped to provide access to several DCI types with different middlewares operated by different providers with different policies and procedures. Science Gateways make use of several solutions for accessing and submitting jobs on different DCI types. Two of the commonly used solutions include SAGA (for accessing Globus, UNICORE, gLite, etc.) and DCI Bridge (for accessing GT2, UNICORE, ARC, OpenStack, Eucalyptus, etc). These two solutions have already been explained in detail in Chapter 2.

### 4.2.3 Workflow Management

Scientific workflow Management is used to execute applications that solve really complex problems such as scientific simulations. In order to support these advanced types of applications within Science Gateways, workflow composition, deployment, editing and execution services should be provisioned. The different SWMS (such as Kepler, Swift, Triana, etc.) which may be used to support such advanced applications is briefly discussed and summarised in Chapter 2. More so, WS-PGRADE/gUSE Science Gateway (in particular) which is designed to facilitate workflow management was described in great detail.

### 4.2.4 Data Management

When jobs are executed, they usually generate a huge amount of data and therefore require access to data storage. Also, the most important entity in scientific research activity is data. Therefore, Science Gateways should be equipped with very effective data management facilities. Furthermore, different DCI's may have different access protocols as different scientific applications might be stored on different storage resources. As such, Science Gateways should enable access to the most important storage types. This is mostly achieved by hiding the technical details of accessing the different storage resources and providing an easy to use interface via Science Gateways for managing, uploading, downloading and transferring data between different types of storage resources. Science Gateways should also enable easy data discovery by providing tools to query on metadata. Data operation activities can be summarised to include storage, access control, metadata management, data sharing, etc.

## *4.3 Investigation of the different services of the Science Gateway*

This section investigates the different services of Science Gateways discussed in Figure 4.1 using one of the Science Gateway framework that was discussed in Chapter 2. The adopted Science Gateway framework that was used in this research is the Catania Science Gateway Framework (CSGF). The CSGF approach is used to investigate all the different Science Gateway services that were identified such as security management, job management, workflow management and data management as shown in Figure 4.1. According to the literature (and based on the number of application-specific gateways or instances of the gateway in use) CSGF along with WS-PGRADE/gUSE is one of the most commonly used Science Gateway Frameworks. CSGF was chosen because it represents an exemplar gateway which happens to cater for most of the services that were identified, and thus gives the medium through which these services could be further investigated. In addition, it hides the details and complexity of the underlying middleware and services from users by providing an abstraction over the different middleware implementtions and underlying infrastructures.

### 4.3.1 Security Management Service

CSGF adopts three (3) levels of security mechanisms that allow users to access Science Gateways and their associated DCI resources. In this approach of developing Science Gateways, authentication and authorisation has been decoupled. For the authentication, the

CSGF rely on the identity federation, the SAML standards and its Shibboleth implementation.

The second level of security is provided by the LDAP server which performs the authorisation of users who request access to the different resources of the DCIs. Unlike authentication, authorisation is carried out at the Science Gateway level. For this process, user roles and privileges are stored in the LDAP registry.

The third level of security is executed by the users tracking and monitoring database (also See Chapter 2). This is the auditing process which is done to store all the user information performed via a Science Gateway on every DCI transactions. It interacts directly with the job engine to ensure that all user activities conducted on DCI resources are recorded. The non-repudiability of Grid transactions which is one of the most important Grid Security Infrastructure requirements is also achieved in this way. The user Tracking and Monitoring database also realise the accounting mechanism of the Science Gateway by implementing flow controls over the rate of interactions which are being executed and the number of job submissions per Science Gateway user.

### 4.3.2 Job Management Service

The CSGF uses a module known as job engine of the Catania Grid and Cloud engine for job management and the JSAGA API (also See Chapter 2) to submit jobs towards different DCIs. The job engine manages the whole life cycle of the job execution right from the submission stage until the final outputs are retrieved. The job engine maps all job operations to JSAGA functionalities thereby allowing users the full use of the job management services. More than just simple job submissions, job operations such as monitoring/status check and retrieval of outputs are also mapped as part of JSAGA functionalities.

### 4.3.3 Workflow Management Service

The CSGF utilises Kepler for the execution of workflows in a distributed environment. Kepler utilises different workflow engine and workflow models to perform data modelling and analysis. It consists of the different layers such as presentation layer, user services layer, WEP generation layer and the infrastructure layer. It operates with the cloud via Kepler/Amazon EC2 actors and thus creates EC2 virtual machines and Elastic block store volumes. (The CSGF is currently working on using Kepler but is yet to be fully implemented within the framework. However, this could serve as a platform for conducting future work in this research (See future work discussion in Section 8.6 of Chapter 8).

### 4.3.4 Data Management Service

In the CSGF, all data operations are managed by a module known as data engine. Data operations may include: upload from local storage to remote storage, download from remote storage to local storage, 3rd party file transfer between two remote storages, update and edit operations, etc. The data engine is a module within the Catania Grid and Cloud engine which is used for data management and it uses the JSAGA API to execute the aforementioned data operations. In addition, the data engine also enables the direct transfer between the Science Gateway and all storage elements within the DCI environment. It ensures the possibility of arranging files in folders in ways similar to the structure of file systems on the physical disk. The Data Engine utilises an Object-Relational Mapping (ORM) library which provides a framework that enables the mapping between the actual database instance (MySQL) tables and each Java objects.

## *4.4 Proposed Methodology for Developing Scientific Software Applications in Science Gateways*

Some of the essential services which a Science Gateway should possess (as seen in Figure 4.1) have been discussed in Section 4.2. These functionalities include security management, job management, workflow management and data management. As discussed earlier, the investigation of these different services is done using a well-known Science Gateway approach otherwise known as CSGF.

Several applications may need to adapt one or more of the aforementioned Science Gateway services therefore a methodology for developing the applications and the services they require becomes really paramount. Consequently, this section will outline a proposed MESSAGE methodology that can be used to develop a scientific software application in Science Gateways. A case study is developed to test and implement the proposed MESSAGE methodology while a second case study will be used to examine its feasibility.

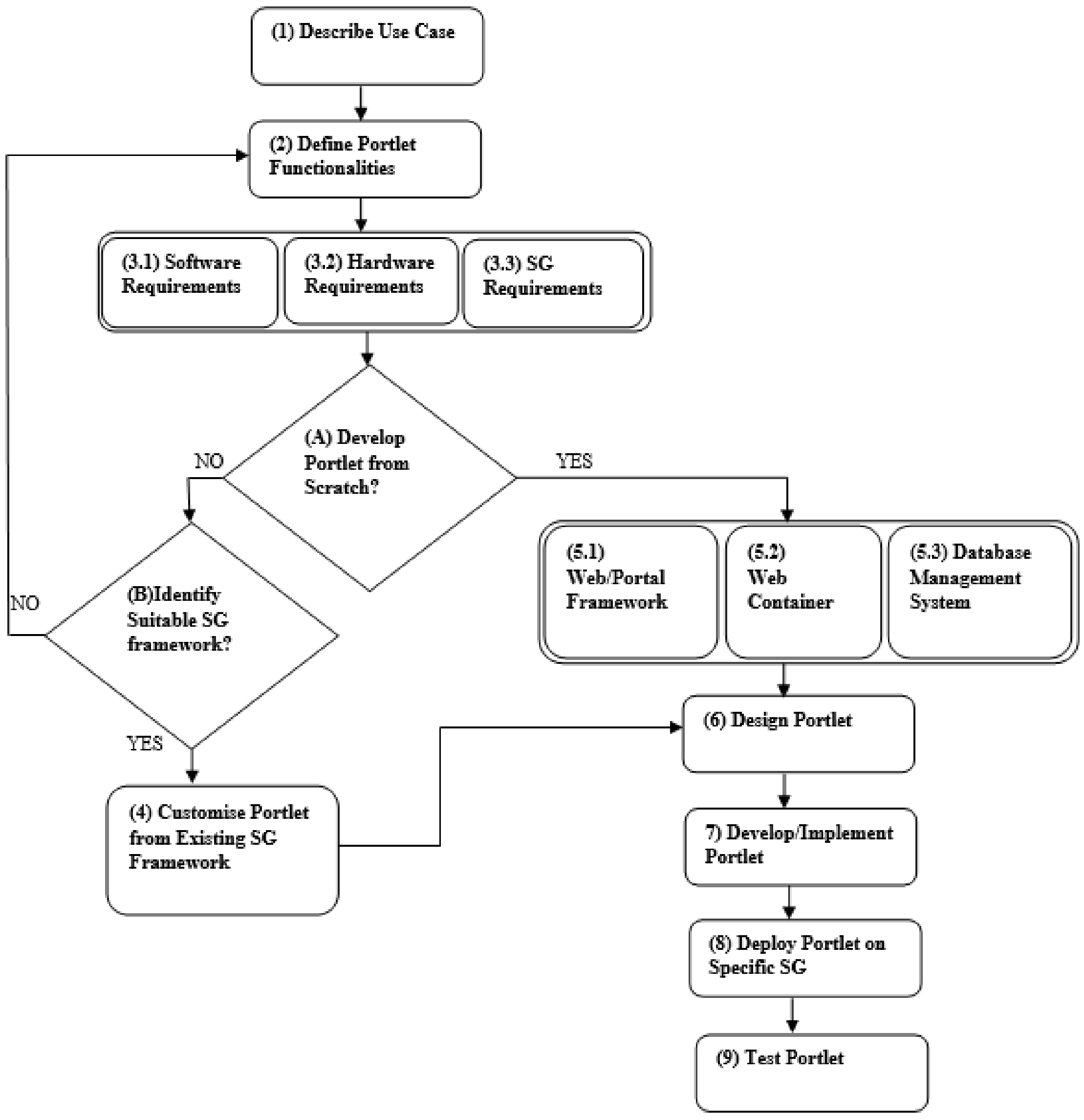

(1) Describe Use Case
(2) Define Portlet Functionalities
(3.1) Software Requirements
(3.2) Hardware Requirements
(3.3) SG Requirements
(A) Develop Portlet from Scratch?
NO
YES
(B)Identify Suitable SG framework?
NO
YES
(5.1) Web/Portal Framework
(5.2) Web Container
(5.3) Database Management System
(4) Customise Portlet from Existing SG Framework
(6) Design Portlet
7) Develop/Implement Portlet
(8) Deploy Portlet on Specific SG
(9) Test Portlet

Figure 4. 2 A Proposed Methodology for Developing Scientific Software Applications in Science Gateways (MESSAGE)

The MESSAGE methodology in Figure 4.2 starts by analysing a simple case study in (1) that is suitable for implementing the proposed MESSAGE methodology for developing scientific software applications in Science Gateways. Different aspects of the case study such as the scientific domain where it is applicable, the problem areas, the software use and how different actors/users will interact and benefit from its use are discussed in the case study description stage.

The portlet functionalities in (2) define the major components of the portlet. Figure 4.1 captures both the back-end oriented Science Gateway services as well as the front-end (graphical user interface) layer. The backend provides the necessary DCI access mechanisms which are necessary to realise the typical gateway functionalities. Therefore, the major focus in (2) is to define specialised user interfaces that will provide the required front-end for the target user communities. This includes user interfaces for job submission, certificate management, file and data management, etc., for the execution of jobs in DCIs. The choice of user interface usually depends on the different communities of practice or the kind of application that needs to be supported on the Science Gateway. In essence, each user community typically requires tailored user interface functionalities based on their specific needs or domain of science. Therefore, the portlet functionalities defined at this stage can either be specific to the needs of a particular community or generic to different communities of practice (which are already embedded within Science Gateway frameworks).

The general requirements of the portlets is categorised into software requirements, hardware requirements, Science Gateway requirements and user requirements. The software requirements in (3.1) consist largely of both the operating systems (such as MS Windows or Linux OS) and also partly consist of the software stacks required for enabling the Science Gateway technologies. The hardware requirements in (3.2) on the other hand simply define the required system resources (such as Grid and Cloud services) where scientific jobs will ultimately be executed in a distributed environment. The Science Gateway requirements in (3.3) define a suitable Science Gateway approach required for developing a portlet (i.e whether the portlet is developed from scratch or customised from existing Science Gateway framework and the adopted Science Gateway framework if the latter option is chosen). However, the different user requirements that are incorporated within the user interface layer and which usually defines the portlet functionalities are not defined at the requirements section. The user interface (otherwise known as the Graphical User Interface) consists of the front-end layer which may be utilised by a Science Gateway user in order to interact with the back-end services of DCIs. Therefore, the user requirements are captured in the portlet functionalities stage in (2).

Two different approaches may be adopted in order to develop and deploy fully functional scientific software applications in Science Gateways. The first approach is the decision point in (A) where developers can decide whether to develop portlet from scratch. At this point, if a decision has been taken to create the portlet from scratch, appropriate portal or web application framework, web container and database management system are configured for use with the required backend services. However, this approach may be time consuming as portal frameworks do not come with back-ends that support access to DCIs. The second approach is the decision point in (B) where developers can identify suitable Science Gateway framework (if a decision has been taken not to develop the portlet from scratch). There are different Science Gateway Frameworks (discussed in Chapter 2) that developers can utilise in order to develop their application specific Science Gateways. By identifying a suitable Science Gateway framework, developers already have the required configuration for specific portal or web application framework, web container and database management system which are tightly coupled with the adopted Science Gateway framework. This process initiates the customisation of the portlet in (4) by having the necessary configurations for the development environment and the actual implementation can thus begin. The option to develop application specific Science Gateways from existing Science Gateway frameworks is better (in terms of time and effort) than developing scientific applications from scratch as it is already provisioned with the necessary back-ends that support access to different DCIs and realise the typical gateway functionalities such as job submission management, security management, file and data management and workflow management. In addition, this approach has the added advantage discussed in (A) where portlets are built using portal frameworks such as Liferay portal, since Science Gateway frameworks are already built on top of such portal frameworks. Developers can therefore select any one of these approaches (i.e A or B) based on the type of application to be developed as well as other factors such as time constraints.

The Science Gateway software stacks are defined by web/portal framework (in 5.1), web container (in 5.2) and database management system (in 5.3) which provide the various enabling technologies essential for developing both the Science Gateway frameworks and the application specific Science Gateways. This includes a portal or web application framework for executing applications (such as Spring and Liferay), a web application container for hosting the applications (such as glassfish and tomcat server) and a database management system (such as MySQL) that keeps track of users, service actions and states. Furthermore, the configuration and use of the aforementioned software stacks is very important especially for developers who have opted to develop their portlets from scratch. However, most of the Science Gateway frameworks already come with the required configurations for specific portal or web application framework, web container and database management system (in addition to them being written on specific portal technologies). It is important to clarify here that since the researcher has opted to customise from existing Science Gateway framework, only one part of the MESSAGE methodology was tested. As part of the future work of this research, the other part of of the MESSAGE methodology in 5.1, 5.2 and 5.3 where developers need to develop Science Gateways from scratch, specify and set-up the different configurations (for the web/portal framework, web container and database management system) will be tested.

There are two main advantages when developing portlets from existing Science Gateway frameworks. The first is the provision of access to existing DCIs which is made possible via modules that enable portlets to interact with different DCI resources. Examples include the Grid and Cloud engine (which adopts the OGF standard SAGA and its JSAGA implementation of the CSGF) and the DCI-Bridge which are both generic job submission service that can submit jobs to all the major DCI types. Both of these modules have been explained in great detail in Chapter 2. They are used to provide the DCI access mechanisms which are required for the realisation of the Science Gateway services in Figure 4.1. They are typically generic in nature and can thus be used for the development of different Science Gateway instances. In a similar manner to the software stacks discussed above, these types of modules are tightly coupled with specific Science Gateway frameworks which can ultimately reduce the time needed to develop fully fletched application specific Science Gateways. The second advantage of using already existing Science Gateway frameworks involve the identification and configuration of the required DCI resources which are necessary to execute scientific jobs. These DCI resources, which are usually tightly coupled with the associated Science Gateway framework, are used to address the needs of researchers for digital services in terms of networking, computing and data management. A variety of working methods based on the shared use of ICT tools and resources across different domains are used for scientific endeavours. These tools include high-speed research communication networks, powerful computational resources (dedicated high performance computers, clusters, large numbers of commodity PCs), Grid and Cloud technologies, data infrastructures (data sources, scientific literature), sensors, web based portals, scientific gateways and mobile devices. All the aforementioned ICT tools are collectively known as e-Infrastructures.

The actual process of developing the portlet captures the steps involved in creating the software (within a portal context), from the design of the portlet in (6), develop/implement the portlet in (7), deploy the portlet on specific Science Gateway in (8), till it finally gets tested in (9). These different stages are captured in the second research type (Design and Development) which was identified earlier in this chapter, where the use of waterfall approach was justified as the ideal SDLC required to achieve this research type. Therefore, some relevant stages of the waterfall approach such as the portlet design, development/implementation, deployment and test are incorporated in the proposed MESSAGE methodology. The underlying importance of the MESSAGE methodology is that by having a generic methodology in place for developing scientific software applications in Science Gateways, it will help in simplifying the development of scientific applications across multiple scientific domains using different Science Gateway frameworks and technologies.

## *4.5 Summary*

This Chapter presented the services which are considered to be essential in Science Gateways. In addition, it investigates how the identified services are being utilised by using a specific Science Gateway framework. More importantly, it proposes a MESSAGE methodology for developing scientific software applications (which make use of these services) in Science Gateways. The identified services are based on high-level attributes and functionalities that were identified from the various technologies and projects discussed in Chapter 2.

# Chapter 5: DESIGN AND IMPLEMENTATION OF MESSAGE FOR THE EXECUTION OF SINGLE EXPERIMENT

## *5.1 Overview*

In this Chapter, the Design and Implementation of three different case studies (an Infection Model, WEKA - J48 and the Visualiser) are discussed. In particular, portlets that could execute jobs (sequentially) in a distributed environment are analysed. This Chapter begins with a general description of the first use-case (the Infection Model), its core components and the portlet functionalities and also outlines the rationale for having and implementing a Science Gateway for an Agent Based Modelling Simulation (ABMS) application.

To investigate some of the different identified Science Gateway services, Chapter 5 is organised as follows. Section 5.2 presents an overview of the first case study (an Infection Model). Section 5.3 introduces the first version of the portlet and Section 5.4 outlines the portlet functionalities. Section 5.5 presents the portlet requirements such as software, hardware, and Science Gateway requirements. Section 5.6 and Section 5.7 discusses the design and implementation of the Infection Model portlet, respectively. A general discussion of the use of the portlet on the Science Gateway is presented in Section 5.8. Furthermore, a second case study (WEKA - J48) is introduced and analysed in Section 5.9. Section 5.10 outlines the portlet functionalities and Section 5.11 presents the general requirements which further builds on the requirements defined for the first portlet. Section 5.12 discusses the design of this portlet. The implementation of the portlet is presented in Section 5.13 and the use and test of the WEKA - J48 portlet is discussed in Section 5.14. Finally, the processes involved in porting a visualiser portlet (that is used for the analysis of the simulation output results of the Infection Model portlet) are also discussed. The Case study description is presented in Section 5.15 and Section 5.16 analyses the portlet functionalities. The design and implementation of the portlets are presented in Section 5.17 and Section 5.18, respectively. Finally, the use of the portlet is described in Section 5.19 and Section 5.20 discusses the summary of the entire chapter.

## *5.2 Case Study Overview – An Infection Model (1)*

The SIR model is used to model the flow of people between three states: Susceptible (S), Infected (I), and Removed (R) (Boccara and Cheong, 1992). Based on diseases status, the individuals are divided into the three different disjoint groups. The Susceptible groups are those individuals who are not infected but are capable of contacting the disease and become infected. The Infected group are the individuals who are capable of transmitting the disease to susceptibles and the Removed are the individuals who have the disease and are dead, or isolated, or have recovered and are permanently immune. According to (Hethcote, 1976), the SIR model without vital dynamics might be appropriate for describing an endermic outbreak during a short time period, whereas the vital dynamics would be appropriate over a longer time period. Vital agent diseases such as small pox may have occasional large outbreaks in certain communities and yet be endemic at a low level in large population groups. The SIR model is used where individuals infect each other directly (rather than through a disease vector). An individual who recovers from the illness is also modeled to have perfect immunity to the disease thereafter and contact between people is also modeled to be random. Resistance against an infectious disease is the protection that reduces an individual’s risk of contracting the disease (Reluga and Medlock, 2007). Usually, the rate that people become infected is proportional to the number of people who are infected and the number of people who are susceptible. If there are lots of people infected, the chances of a susceptible coming into contact with someone who is infected is high. Likewise, if there are very few people who are susceptible, the chances of a susceptible coming into contact with an infected is lower (since most of the contact would be between the non-susceptible people, either infected or resistant).

ABMS is an approach to modelling systems that comprise of autonomous, interacting agents (Macal, North 2005). Advances in computational resources enable (across a variety of application domains) a growing number of agent-based models. An agent is considered as any independent component whose behaviour could range from primitive reactive decision rules to complex adaptive intelligence. In a more practical sense, an agent may be autonomous and self-directed, modular or self-contained, social and interacting with other agents (Macal, North 2009). This constitutes properties and attributes that an agent can possess. The different applications which the principles of ABMS can apply may therefore vary from modelling agent behaviour in consumer markets, supply chains, stock markets, to mitigating the threat of bio-warfare and predicting the spread of epidemics.

More specifically, the effective control of the transmission of infections requires a thorough understanding of the determinants and patterns which resulted in the spread of such infections. Over the years, scientists have used simulation techniques to develop computer models to model the interactions between individuals and their social networks. These simulation techniques could range from deterministic to stochastic models. ABMS is used by different scientific domains to study the behaviour of adaptive systems and usually complex social networks. However, once a simulation has been developed, there may be challenges in how a community of practice can quickly execute it and get results promptly especially with Agent-based models consisting of large populations of agents. Therefore, efforts could be hampered by performance considerations since an important criterion for ABMS is that, to be amenable to comprehensive and systematic analysis they must be able to run quickly (Collier, Ozik et al. 2015).

To demonstrate how Science Gateways may be used to execute jobs on DCIs, a demonstration Infection Model is developed. The Infection Model which is implemented using the well-known agent-based modelling simulation platform (otherwise known as the Recursive Porous Agent Simulation Toolkit (REPAST) Simphony) is an example of an Agent-based simulation infection model. This case study was chosen due to its features and also its tendency to help investigate some of the Science Gateway services that were presented in Chapter 2. The aim of the demonstration (using the Infection Model) is to show how scientists can access an ABMS simulation that is used to study the behaviour of infections with an annual outbreak. Three (3) different types of agents represent an entire population namely the infected, susceptible and the recovered population and agents are randomly located in a simulation environment. Susceptible agents try to avoid contact with infected agents. When an infected agent approaches a cell with susceptible agents, it infects one (randomly) selected agent. This susceptible agent then becomes infected and can infect other susceptible agents. The size of the population of infected, susceptible and recovered agents are the input parameters that the user can modify to experiment with different initial conditions. The user can also specify the time (in years) that the simulation will run. Consequently, the infection model portlet has four (4) input parameters namely:

**Simulation period**: This specifies the number of years the simulation will run. By default, the infection model has been pre-set to run for twenty years.

**Recovered Size**: This field specifies the initial recovered population. This is the initial healthy population which have immunity and cannot be infected immediately. However, after establishing contact with the infected population, they lose their immunity and become susceptible to infection.

**Infected Size**: This field is used to specify the initial infected population. Infected population can infect susceptible population upon contacting them. They (however) recover after a period of time and become healthy.

**Susceptible Size**: This specifies the initial susceptible population. Susceptible counts can be infected when contacted by infected population. If more than one susceptible agent is in the proximity of an infected agent, only one is infected.

In the context of this research, an experiment can be defined as the process whereby a potential Science Gateway user can specif the different input parameters (defined above) via the Science Gateway interface in order to study the behaviour of infections with an annual outbreak. Therefore, starting with an initial population of 1500 susceptible agents, 20 infected and 0 recovered, when an infected agent approaches a susceptible agent, it becomes infected and if there are more than one susceptible count in the cell, only one (randomly selected count) is infected. Infected agents recover after a period of time (uniform distribution) and become healthy with a level of immunity. Recovered counts immunity decreases every time they are approached by an infected count and when immunity becomes 0, the recovered agent becomes susceptible and can be infected.

In the implementation of the Infection Model portlet, potential users can select a specific set of input parameters and execute the simulation on DCIs from an interface that hides the complexity of the underlying middlewares and infrastructures. This is achieved (in parts) by developing a simple but intuitive user interface (otherwise known as portlet which is deployed on the Science Gateway) for running and analysing simulation experiments of an ABMS model on different DCIs.

The evolution of distributed systems have ensured that scientists have a large set of resources to run jobs and obtain results in good time. Simulation experiments with a considerable number of agents can now be run on distributed systems by distributing the agents among computer nodes which also reduces any potential memory pressure as the number of agents per computer node becomes lower. In this way, the power of clusters and other High-Performance Computing resources are leveraged to distribute a model across processes. In particular, advances in networking and distributed computing techniques where different organisations can combine researches and resources across multiple administrative and organisational domains (which is realised in Grid Computing) have changed the way scientists perform their experiments. This has evolved into complex international Information and Communications Technology (ICT) systems referred to as e-Infrastructures. Authorised scientists can therefore access their simulation and increase computational power by using distributed computing resources in this way to effectively perform experiments. On the other hand, the deployment and use of these resources are extremely complex and could be quite a daunting experience which could, in turn, prevent non-ICT expert users from adopting the technology. Historically, users often have to access resources by maintaining their own software or make use of complex programming languages via a command-line interface (Lawrence, Zentner et al. 2015), and may have to deal with other complex technical issues such as: job service description languages, execution scripts and the management of personal digital certificates across different administrative domains.

Two main issues therefore emanate from the adoption of distributed systems. The first issue is the easiness of access to these environments, while the second issue relates to the interoperability of the underlying infrastructures and middlewares. Consequently, the first iteration focused on developing a simple but intuitive user interface otherwise known as Science Gateway for running and analysing simulation experiments of an ABMS model on different DCIs. This is the view that by making use of this Science Gateway, users (especially non-ICT expert users) can easily access and execute their jobs on the distributed systems without prior knowledge/details of the underlying infrastructures and middlewares.

## *5.3 The Infection Model Portlet (For Single Experiments)*

The first version of the Science Gateway (a portlet that can execute an ABMS jobs sequentially on an e-Infrastructure) is designed by considering two major inputs. These are:

- The points noted in the first two steps of the DSR cycle (Awareness of the problem and suggestion), and

- The user requirements of the ABMS portlet (as detailed in the user requirements section) of both users and the communities that could potentially benefit from its use. Consequently, the implementation was then carried out based on the design that ensued.

Accordingly, findings obtained from the review of literature show that fully fledged Science Gateways can be developed either by building from scratch (by making use of standard portal building technologies) or customising an existing Science Gateway framework to suit the needs of specific communities of practice. This usually starts with the collection of the necessary information relating to the needs of specific communities. Such needs could vary from identifying the various ways in which users/scientists carry out experiments and the use of the underlying infrastructures required to execute their jobs (usually the requirements and functionalities of the user interface), to what e-Infrastructure resources are necessary to support the community of practice. The need for different e-Infrastructure resources could vary with different communities of practice who may require access to sensors, instrumentation, high-speed communication networks, powerful computational networks, storage resources and high performance computing facilities. Usually (and for different communities of practice that were identified for an African project known as ei4africa) a questionnaire-based approach is used to identify and collect necessary information regarding the different needs of each community.

For the Infection Model however, such information is collected by examining and identifying the most important features of the model (which consists of the different input parameters) and several consultations with a simulation expert. Other requirements were also identified due to the researcher's knowledge of the subject area which made it easier to project the necessary simulation needs.

## *5.4 Define Portlet Functionalities (2)*

This section constitutes the system functionalities from the users' point of view as well as defines the different user requirements that justify the need to have an infection simulation portlet. As mentioned earlier, the researchers' knowledge in this area, coupled with the direct interaction with the developer of the Infection Model presented the opportunity to collect detailed information that captures the necessary user requirements of the Infection Model.

Generally, in order for Science Gateways to support a particular community of practice, the various needs must be specified and analysed. These needs, in large parts, constitute the user requirements on the one-hand as well and the system requirements on the other. For the Infection Model (for instance), to specify ways in which a Science Gateway can be used requires that different simulation needs must be identified. Usually, a simulation would often require the upload of data or model, execution management, the collection of output/result and, in some cases, perform visualisation activities on the collected output/result. Based on the general system requirements and the components which make up the Infection Model the following deductions are made for the user interface requirements. A potential user of the portlet will:

- Be able to easily access the Infection Model portlet page on the Science Gateway.
- Be able to specify different input parameters of the Infection Model.
- Be able to perform and execute jobs of the Infection Model on DCIs.
- Be able to monitor jobs for statuses and updates.
- Be able to download outputs/results of simulation experiments.
- Be able to analyse the result of simulation experiments via a visualiser portlet page.

It is important to note that the user requirements that were identified here are consistent with the first two stages of the design research cycle (Awareness of the problem and suggestion stage) which emphasised on the identification of interesting research problem and the ensuing new system functionalities that should be defined. All the aforementioned requirements are incorporated at the graphical user interface of the portlet.

## *5.5 Requirements*

The following section will discuss the requirements specification stage of a fully functional ABMS Science Gateway by defining the requirements of the portlets at different levels. This will include the software requirements, hardware requirements and the generic Science Gateway framework which binds all the requirements together.

### 5.5.1 Software Requirements (3.1)

To develop a fully-fledged Science Gateway, i.e. integrate scientific case studies through the pervasive adoption of web technologies and standards and make them available to their end users through Science Gateways, the required stacks of software framework at different levels must be considered. These Science Gateway stack may include an application server which hosts the individual components, a runtime environment that executes the web applications, a database management system that keeps track of users, service actions and states, and a specialised library which provides an interface to various kinds of computing and data endpoints.

There are several options available to developers. The application server/runtime environment consists of different implementations such as the Python/Django, Ruby/Ruby on Rails, Java, NodeJs/JavaScript etc. The various database types may include the use of MySQL, PostGresSQL, and Oracle, while the specialised library necessary to provide the interface to the different kinds of endpoints could require the developer to either write his/her native interface or make use of the available standard that governs different interfaces such as SAGA. There are different implementations (such as Java and Python) of the SAGA standard. However, the Java stack was chosen as the language of choice for the implementation of all the essential Science Gateway stacks. For the specific selection of components that have been chosen for the development of the ABMS Science Gateway, the following implementations have been adopted for the different Science Gateway stacks:

- In terms of the website/portal builder, the Liferay portal system (an open source application container) and its software development kit were chosen (Liferay portal bundle and SDK). The Java stack was chosen due to the existence of well-known and documented standards, particularly, the JSR 168 and JSR 286. These standards make portlets portable between application server portal frameworks.
- Just beneath the portal stack, there is the specialised library which translates the actions of the portal to the actions to be conducted on the computing infrastructures. There are standard APIs to this functionality and since the adopted implementation is a Java stack, the JSAGA implementation was chosen. Ideally, it is desirable to exploit as many computational resources as possible but since each of these resources have unique interfaces for authentication and authorisation, resource discovery, job submission and management, data movement/storage, and accounting, it is therefore necessary to have an interface that is common to the available resources. JSAGA therefore provides the Java implementation of the SAGA standard that allows a unique interface to most of these functionalities and is independent of the actual

remote type. This is the library that enables the transmission of calls from the Science Gateway and submits those calls, after having translated them appropriately into lots of different kind of target infrastructures. These may include DCI such as production Grids, single or distributed Cloud sites, in the case of the federated Cloud of the EGI, or a single HPC centre with direct access via SSH.

- In terms of the execution environment/application server, the Java stack (Java JRE and JDK) which translates into a runtime environment and a Java development kit is utilised. Several Java application servers (also known as servlet or web container) are available for both commercial and open source. Several open source options are available such as Jetty, Geronimo, Tomcat, JBoss and Glassfish. However, the choice application server that was chosen is the Glassfish application server.
- For the database management system, MYSQL database was chosen due to its ubiquity and for the simple reason that the right configuration of MYSQL is already present in CSGF.

### 5.5.2 Hardware Requirements (3.2)

Two different methods that is used to execute the ABMS jobs on the distributed systems are:

- n Virtual Machines (VMs) with 1 core each at different cloud sites used for sequential job execution and;
- n VMs with M cores each at different cloud sites used for parallel job execution.

The sequential execution sees the mapping of ABMS jobs to VMs with single core across multiple Cloud sites, while the parallel execution maps ABMS jobs to VMs with multiple cores across different Cloud sites. The execution of jobs is performed primarily by using the EGI federated Cloud (a federation of institutional private Clouds that offers Cloud services to researchers in Europe and worldwide). Job execution can also be done with a single system that can scale up to user needs, integrate with multiple providers to give resilience, prevent vendor lock-in and enable resource provision that is targeted towards the research community. The EGI Federated Cloud enables standards-based federation of IaaS Cloud which exposes a set of independent Cloud services which are accessible to users that utilise a common standard profile and allows deployment of services across multiple providers and capacity bursting. The EGI federated Cloud have three different types of members namely user communities, technology vendors, and resource providers. The resource providers make use of different Cloud management framework such as OpenStack, OpenNebula, Synnefo, Cloudstack, etc., to build and manage Cloud computing platforms for public and private Clouds.

### 5.5.3 Science Gateway Requirements (3.3)

It was earlier mentioned that fully-fledged Science Gateways are either built from scratch (which could take a lot of time and effort) or customised from existing Science Gateway frameworks. These Science Gateway frameworks consist of Science Gateway stacks and make use of standards as already discussed in the case of the CSGF above. To develop the Infection Model portlet the CSGF approach, whose core components comprise of the Catania Grid and Cloud Engine, is adopted. The CSGF has an interface for interacting with the embedded portlets (known as the Science Gateway interface) as well as the interface for exploiting the computational resources of the different DCI's (JSAGA interface). It helps to submit jobs, retrieve data, and interact with the different DCIs. It therefore provides functionalities such as enabling secure access to the gateway, submitting simple jobs, monitoring job statuses, managing data storage, transfers and retrieval activities and making sure that submitted jobs are executed on the targeted infrastructures. Much of the functionalities which are required by either the user agreement of the remote computing federation or particular user communities have to be provided by the portal stack. CSGF provides extra Java libraries that ensure compliance with EGI Grid and FedCloud infrastructures, amongst others.

## *5.6 Infection Model Portlet Design (6)*

The Infection Model design is done based on the portlet functionalities which were presented in Section 5.4 above. Other requirements such as the software requirements, system requirements and the CSGF all combine to constitute the required building blocks necessary for the implementation of a fully-fledged ABMS Science Gateway. Consequently, the prototype design starts with the user interface based on the user requirements necessary for executing ABMS jobs. This design represents an approach which constitutes all the requirements needed to facilitate and execute a simulation job on distributed systems starting with the access requirements, the specification of the input parameters, job execution (data/model upload) and the subsequent collection of outputs and the visualisation of results. The design process starts by considerig the standard approach to authenticating and authorising a user to the Science Gateway irrespective of the application being accessed, which is facilitated and constitutes a major part of the CSGF. In addition, it provides the capability of supporting the specification of different input parameters (Simulation Period, Recovered Count, Susceptible Count, and Infected Count) that constitutes the ABMS Infection Model portlet. All the information which are collected via the Infection Model portlet are submitted to the distributed systems and executed using a standard interface (JSAGA API/Libraries) of the CSGF. Furthermore, it makes provision for a dedicated portlet where the statuses of all submitted jobs (running and done jobs) are displayed and monitored. As such, a user can retrieve the output of their experiments for subsequent analysis. Finally, the retrieved outputs can be analysed by the user's local visualiser or uploaded to the Infection Model visualiser which is provisioned on the Science Gateway. Figure 5.1 shows an overview of the system processes.

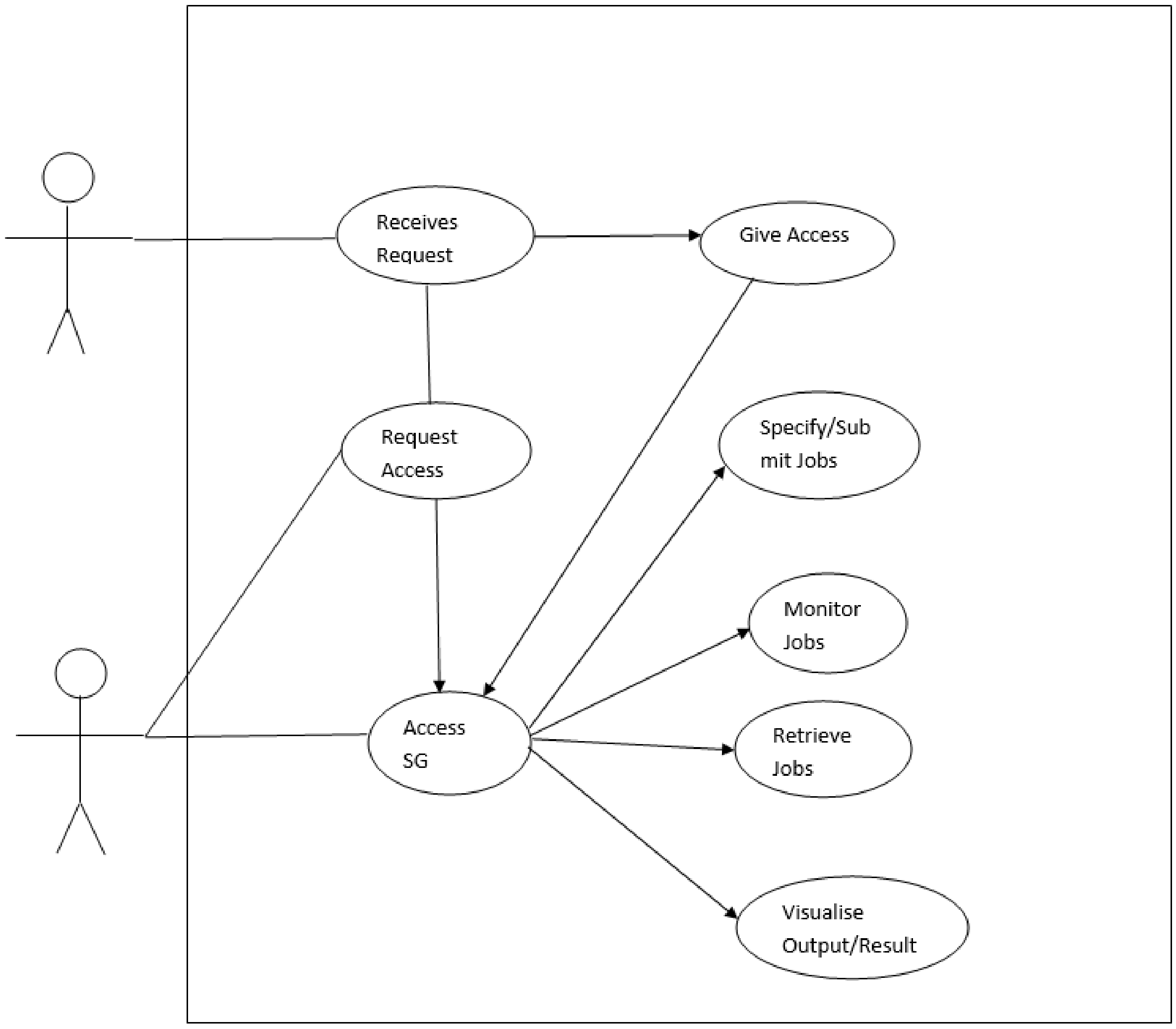


Figure 5.1 Infection Model Use Case Diagram

It is important to note from the use case diagram and consequently the system design process that both the sequential and parallel approach has been taken into consideration. However, the parallel execution of ABMS jobs which largely constitute a part of the entire process is examined and documented in Chapter 6. For the remainder of this chapter, the complete design and implementation of the sequential approach to developing the ABMS portlet in Science Gateways will therefore be described. By examining the above Figure 5.1, the different stages that will realise the above functionalities is described as follows:

1) The user seeks to access the Science Gateway.
2) Science Gateway takes the user through an authentication and authorisation procedures.
3) A user submits his/her credentials to the Science Gateway for access.
4) Based on the user roles and privileges, Science Gateway presents the desired portlet/application to the user.
5) The user can view portlet, specify different input parameters and submit jobs.
6) Science Gateway, using a pilot script, can collect input parameter being specified through the user interface and submit jobs to e-Infrastructure.
7) Science Gateway gives job statuses, and a user can monitor and download the outputs of jobs.
8) The user can visualise the simulation output results.
9) Procedure ends.

From the above sequence of steps, six main building blocks could be derived to represent and implement the initial prototype architecture. These include user access, user interface, pilot-script.sh, pre-configured virtual machine, myjobs portlet and the ABMS Visualiser. In addition, steps 1-4 above which captures the procedures for user access is built as part of the CSGF and represents the standard authentication and authorisation process for all embedded applications/portlets. Steps 5-8 captures the execution of jobs (i.e. the specification of Infection Model jobs via the portlet's interface, job monitoring and the collection and visualisation of simulation output results).

### 5.6.1 User Access

This is the standard access procedure (steps 1-4 above) within the CSGF which handles both user authentication and authorisation. Users that want to access the Science Gateway for the first time must register by filling out a dedicated web form. This is done by specifying the identity federation and the identity provider which the user belongs and, once this request is confirmed by the user, it is forwarded to the portal administrators. The administrators will determine whether to accept or reject this request and, if the request is granted the user information is stored on a module known as the LDAP registry. The user is either notified that they are successful or that request has been denied. When a user attempts to log-on to the Science Gateway, they redirected to a web page where they are prompted to select their respective identity federations and identity providers. A log-in page is presented afterwards and the user can submit their credentials (consisting of a combination of username and password). After a successful verification, the Science Gateway then checks to see if the user is registered on the LDAP registry (by checking what roles and privileges they have on the Science Gateway). Users who belong to different organisations may have different roles and privileges for each application on the Science Gateway and for the community it was developed. Therefore, a user is presented with the web page of an application and thus able to run jobs on the DCIs based on the roles and privileges.

### 5.6.2 User Interface

This module gives a general description of the ABMS Infection Model which includes the aim of the model and the tool that was used in its actual implementation. It gives the user a brief description of the different input parameters which are specified, coupled with how the ABMS Infection Model portlet may be utilised. Finally, it facilitates a way to specify the different input parameters of the ABMS Infection Model portlet (such as Simulation Period, Recovered Count, Susceptible Count, and Infected Count). The user interface is designed to simply enable authorised users to select from a pre-defined set of experiments which are specified by the developer.

## *5.7 Develop/Implement the Infection Model Portlet (7)*

After defining the general requirements such as software requirements, hardware requirements, the Science Gateway Framework and the design process of the ABMS portlet, the following section will describe a detail implementation of the Infection Model application using the CSGF. To implement the user interface of the Infection Model portlet the CSGF, which utilises the Liferay portal framework, is adopted. Liferay consists of many inbuilt applications called portlets. The Infection Model portlet was deployed in a portlet named

Infection Model. This portlet utilises several classes such as the AppInfrastructureInfo class (which stores the required information necessary to submit a job to a give infrastructure), the AppPreferences class (for storing all the values of portlet preferences), the AppLogger class (for displaying information about the console outputs) and most importantly the myRepast-infection-portlet class (main portlet class which extends and overrides the GenericPortlet class methods). In addition to the different classes used by the Infection Model portlet, several Java Server Pages (JSP) are used to present the desired views/pages to the Science Gateway users. These include the View page (which presents the user interface of the Infection Model portlet), the Edit page (for the Infection Model customisation), the Help page (for displaying instructions on how to use the Infection Model portlet) and the Submit page (for informing users about each successful job submission). For the implementation of the portlet, the following section will discuss the different classes which make up the portlet. Furthermore, the components of the main class (myRepast-infection-portlet) and the various JSP pages are discussed. Finally, the relationship between the main class (myRepast-infection-portlet) and how it interacts and make use of the different JSP pages are presented.

### 5.7.1 Agent-Based Simulation Portlet Mode

The portlet specification defines three portlet mode: VIEW, EDIT and HELP (Fabiyi *et al.*, 2016)

A. View Mode: Generates a mark-up and presents the normal user interface of a portlet.
B. Edit Mode: Allows for the customisation of an application and the setting of preferences.
C. Help Mode: Outlines the ABMS functionalities by specifying the portlet usage instructions.

Consequently, the different pages of the ABMS portlet have been developed using the view page and the submit page. These pages have been developed primarily by utilizing such programming languages as HTML5 and JavaScript. This is discussed as follows:

#### 5.7.1.1 The View Page

The view page consists of the normal user interface of the ABMS portlet. This page incorporates some or most of the requirements that were specified at the user requirement stage. It starts with a general description of the Agent-Based Simulation Infection Model, its implementation, how to utilize the portlet and the communities that could potentially benefit from its use. It also facilitates and enables potential users to specify the different input parameters. However, for this version of the portlet, a set of experiments have been pre-defined and hard-coded. Therefore potential users can only select and run experiments from amongst the available experiments that are specified within the portlet. This is due to the limited resources which was available at the time of developing this portlet and to prevent users from making excessive or unrealistic use of the computing infrastructures. The Infection Model portlet consists of a total of five experiments and a user may specify input parameters by simply selecting from pre-defined sets, after which the job may be submitted and executed on the DCIs by clicking the submit button provisioned on the user interface. The view page is mandatory both at the Science Gateway level (production) and the development stage of the portlet.

#### 5.7.1.2 The Submit Page

The submit page informs the user of any successful job submission. It presents the job identifier and informs the user where the output of jobs are to be retrieved. This component is a dedicated portlet known as MyJobs portlet. It also includes a link to run new applications which ultimately redirects users back to the View page. The submit page is mandatory both at the Science Gateway level and the development stage of the portlet.

### 5.7.2 The Infection Model portlet Classes

These include: The AppInfrastructureInfo class, the AppPreferences class, the AppLogger class and myRepast-infection-portlet class.

#### 5.7.2.1 The AppInfrastructureInfo Class

This class stores the required information for the portlet to submit a job to a given infrastructure. This include the use of different variables to store information such as enableInfrastructure, nameInfrastructure, acronymInfrastructure, bdiiHost, wmsHosts, pxServerHost, pxServerPort, pxServerSecure, pxRobotId, pxRobotVO, pxRobotRole, pxRobotRenewalFlag, pxUserProxy, and softwareTags. A brief description of each of the variables (of the AppInfrastructureInfo) and the given default values are given as follows:

enableInfrastructure is used to enable or disable an infrastructure. The assigned default value is set to a boolean string.

nameInfrastructure stores the infrastructure name. The assigned default value is set to “Infrastructure name”.

acronymInfrastructure stores the Infrastructure acronym. The assigned default value is “Infrastructure acronym”.

bdiiHost stores the topBDII host name. The assigned default value is set to “BDII host”.

wmsHosts is used for storing a separated list of the enabled WMSs. The assigned default value is “WMS host”.

pxServerHost stores the eTokenServer hostname. The assigned default value is set to “Robot proxy server host”.

pxServerPort stores the etoken server port number. The assigned default value is set to “Robot proxy server port”.

pxServerSecure stores the eTokenServer secure connection flag. The assigned default value is set to “Robot proxy server secure flag”.

pxRobotID stores the Robot proxy identifier. The assigned default value is set to “Robot proxy id”.

pxRobotVO stores the Robot proxy VO. The assigned default value is set to “Robot proxy VO”.

pxRobotRole stores the Robot proxy role. The assigned default value is set to “Robot proxy Role”.

pxRobotRenewalFlag stores the Robot proxy renewal flag. The assigned default value is set to “Robot proxy renewal flag”.

pxUserProxy holds a path to a user Proxy for test job submissions. The assigned default value is set to “User proxy”

SoftwareTags stores a separated list of software tags. The assigned default value is set to “Software tags”.

In addition, there are several methods that are defined by the AppInfrastructureInfo class such as the InfrastructureInfoToAppInfrastructureInfo (which aligns GridEngine infrastructureInfo with the AppInfrastructureInfo), updateInfrastructureInfo (updates the GridEngine infrastructure object by making use of the AppInfrastructureInfo instance), copy (to make a full copy of a given InfrastructureInfo object), updateInfrastructureValue (evaluates a given AppInfrastructure item name with the given item value) and getInfrastructureInfo (which returns the GridEngine InfrastructureInfo object initialised with the data that is contained in the AppInfrastructureInfo). Furthermore, it also defines a set() and get() method for each of the AppinfrastructureInfo fields which are defined above and required for setting particular infrastructures within the myRepast-infection-portlet class.

#### 5.7.2.2 AppPreferences Class

The AppPreferences class is used to store the values of each portlet preferences. The first time the portlet is started, the init() method of the myRepast-infection-portlet class will initialise the portlet preferences values with the corresponding init variables. The use of variables to store different portlet preferences include: gridOperationDesc, portletVersion, logLevel, gridOperationId, numInfrastructures, appInfrastructuresInfo, sciGwyUserTrackingDB_Hostname, sciGwyUserTrackingDB_Username, sciGwyUserTrackingDB_Password, sciGwyUserTrackingDB_Database, jobRequirements and the pilotScript.

In addition to having the corresponding get() and set() methods of the AppPreferences class field mentioned above, which is used by the myRepast-infection-portlet for specifying the portlet preferences, there are also equally important methods such as specify the current infrastructure number (setNumInfrastructures/getNumInfrastructures()), switch to the previous infrastructure (switchPreviousInfrastructure()), switch to the next infrastructure (switchNextInfrastructure()), delete the current infrastructure (delCurrInfrastructure()), add a new infrastructure to the infrastructure list (addNewInfrastructure()) and update appPreferences value of a given preference item (updateValue()).

#### 5.7.2.3 The AppLogger Class

The AppLogger class wraps the apache.common log object which allows the user to enable/disable logs according to a given log level. The higher the level the more verbose the produced output is. The different log level of the AppLogger Class includes Trace, Debug, Info, Warn, Error, and Fatal. Although developers can use system.out.println to print out their console outputs, the use of Java logs is highly desirable. Java Log object offers different output levels to show information. Each AppLogger fields define a method and the AppLogger class also defines another method (the setLogLevel) which allows the portlet to print out all logs types equal to or below the given log level according to the priority. Each AppLogger field accepts a string as parameter consisting of the proper message to show. AppLogger class uses the LogLevel enumerated types to express the log level verbosity. The AppLogger field and their associated values are illustrated in Appendix C1.

#### 5.7.2.4 The myRepast-Infection-portlet Class

The Infection Model is deployed in a portlet named Infection Model portlet. myRepast-infection-portlet represents the main class of the ABMS portlet which extends and overrides the GenericPortlet class methods (i.e. myRepast-infection-portlet extends GenericPortlet). The most important methods within this class include init(), processAction() and render() methods as seen in Figure 5.2. This class provides ways of managing user interaction with the portlet using the combination of Actions and Views. In addition, it provides means of managing different portlet preferences that are stored in the AppPreferences class as well as displaying application information simply by using the log objects. Lastly, it facilitates the method that is used to execute jobs within a distributed infrastructure by utilising the GridEngine methods.

In order to manage the different portlet modes and the corresponding views (i.e. different portlet pages) to display, myRepast-infection-portlet utilises Actions enumeration which constitutes the action status mode (ACTION_ACTIVATE, ACTION_INPUT, ACTION_SUBMIT and ACTION_PILOT) and its corresponding Views enumeration (VIEW_ACTIVATE, VIEW_INPUT, VIEW_SUBMIT and VIEW_PILOT) represents the view modes necessary to manage the application. It defines the AppInit class which stores all the data that is required to submit a job in a distributed infrastructure. Furthermore, the AppInput class that stores all the application input values are also defined. This contains all the input parameters that are specified by the user via the Infection Model portlet main page, such as the simulation period, recovered size, infected size and the susceptible size as defined in Section 5.2. The constructor of this class is created and the input parameters are set to empty strings. In order to retrieve the given application values or manage the user input fields (via the Infection Model portlet main page), a method, getInputForm(), is used to manage all the enumerated types containing all the JSP input parameters of the Infection Model.

Another important method within myRepast-infection-portlet class is the init() method (See Figure 5.2). This method is called when the portlet is installed for the first time or when restarting the portal server. The default values from within the WEB-INF/portlet.xml file is assigned to the application preferences and if the preference values already exist, the default settings are overwritten. To load the default values from the WEB-INF/portlet.xml, the Generic class init() method is overwritten and the application default value is set. The AppPreferences object that stores the Application preferences is instantiated as:

```
AppPreferences appInitPreferences = new AppPreferences(_log);
```

This object is used within the init() method to initialise the Application preferences as shown in Appendix C2.
Furthermore, the SciGwyUserTrackingDB_Hostname, SciGwyUserTrackingDB_Username, SciGwyUserTrackingDB_Password, SciGwyUserTrackingDB_Database, JobRequirements and the PilotScript are set and obtained using a similar approach. To load infrastructure settings the information stored in the AppInfrastructure class which are necessary to submit a job to a given infrastructure are obtained.

### 5.7.2.5 The processAction() and the render() methods

Two really important methods in myRepast-infection-portlet class are the processAction() and the render() methods. The render() methods consist of the doView(), doEdit() and doHelp() method. These methods, along with the init() method, constitute the most important methods within myRepast-infection-portlet class. The java code of the Agent-Based Simulation application showing all the aforementioned methods is therefore illustrated in Appendix C3. Figure 5.2 depicts the entire life-cycle of the Infection Model portlet components. The most important exchange occurs between the processAction() and the render() methods. The processAction() method is responsible for the actions which take place when the user utilises the input form, while the render() method simply present the user with the appropriate interface as a consequence of the user actions.

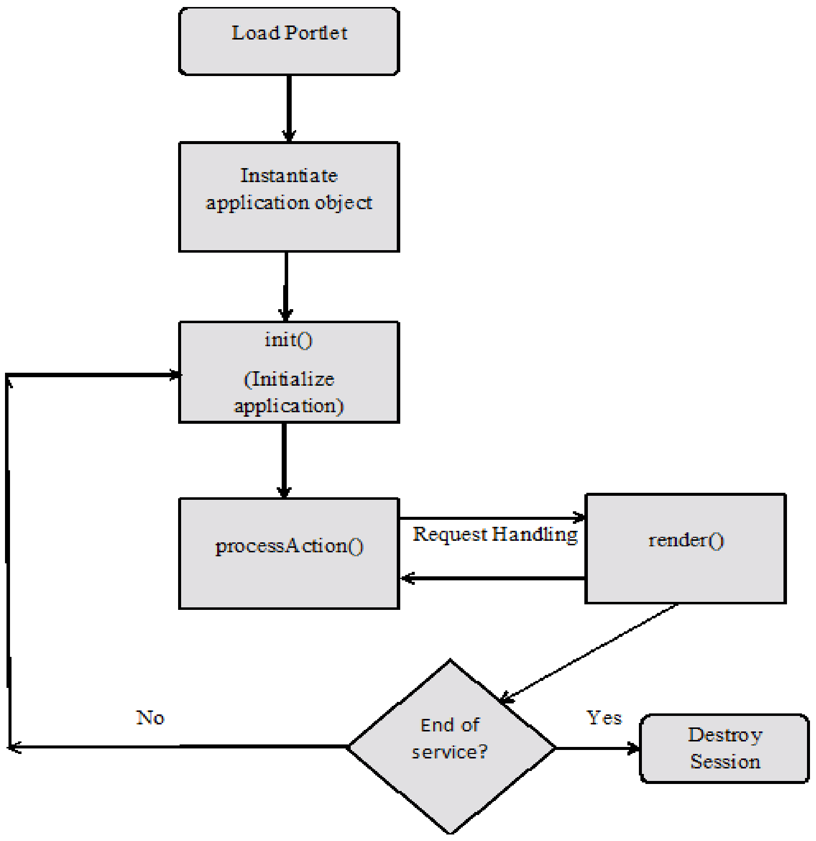


Figure 5. 2 The Portlet Lifecycle

### 5.7.2.6 The processAction() method

The processAction() method enables the Infection Model portlet to process an action request. It is normally called upon each user interaction (i.e. could be a submit button within a JSP page). It determines the current application mode via the actionRequest value, which is a PortletStatus variable and determines the appropriate view mode to assign by using the ActionResponse, which is another PortletStatus variable that is read by the doView(), doEdit() and the doHelp() method, accordingly. This method is of the form:

processAction (ActionRequest request, ActionResponse response)

Within this method, the username is specified and the application path name is determined. It also determines the current portlet mode and forwards the state to the response, according to JSR168/286 standard portlet modes (VIEW, EDIT and HELP) which were discussed in sub-section 5.7.1. To switch the different portlet modes, the actionStatus value is taken from the calling JSP file, through the PortletStatus parameter. The corresponding VIEW mode is stored and the portlet status will thus be registered as render parameter. This is represented as:

String actionStatus = request.getParameter(“PortletStatus”).

If the actionStatus parameter is null or empty, the default action is the ACTION_INPUT (input form). This happens the first time the portlet is viewed.

***The VIEW Mode*** is the normal portlet mode where portlet content is shown to the user. It shows the different actions which may be performed according to the different possible statuses. The actionStatus is used to switch between the different actions (ACTION_ACTIVATE, ACTION_INPUT, ACTION_PILOT, and ACTION_SUBMIT) that were mentioned earlier. This will consequently lead to the different VIEWS (VIEW_INPUT, and VIEW_SUBMIT) to be set using the setRenderParameter() method. When the portlet status is view mode, (i.e. a user is currently on the view.jsp page, and they perform an action), it initiates the processAction() method and all input parameters being passed by the user are retrieved via the ActionRequest. The processAction() then performs an action and calls the RenderRequest which in turn calls the doView() method. Consequently, the doView() method then performs an action and calls the view JSP page.

***The EDIT Mode*** is used to view and set-up the portlet preferences which are stored within the AppPreferences and the AppInfrastructureInfo classes. The edit mode is called after the user has sent the actionURL which was generated by the doEdit() method. It includes methods to store new preferences or to simply effect the changes that were made for the preference settings. The AppPreferences object that stores the Application preferences (See the AppPreferences class) is instantiated as follows:

AppPreferences appPreferences = new AppPreferences(_log);

This object is used to set all the values in the AppPreferences and the AppInfrastructureInfo classes which are necessary to store the required information that the portlet can use to submit a job to a given infrastructure. If the portlet is operating in the Edit mode, the render Request calls the doEdit() method which then sets the configuration variables and calls the necessary edit JSP page.

***The HELP Mode*** is used to display the portlet usage instructions and is simply called by using the doHelp() method. If the portlet is operating in the Help mode, the doHelp() method is initiated, and the help JSP page is displayed accordingly. However, It is also possible to call a JSP page from another JSP page without using the processAction() method. When the user is on the view.jsp page, the RenderRequest is called which bypasses the processAction() as the doView() method calls the necessary view JSP page.

#### 5.7.2.7 The render() methods

The render() methods consists of the doView(), the doHelp(), and the doEdit() methods.

*The doView() method*

This method is responsible for assigning the appropriate application view. The view mode is taken from the renderRequest instance by the PortletStatus parameter or is automatically assigned according to the application status or default view mode. This is of the form:

doView(RenderRequest request, RenderResponse response). The current view status which comes from the processAction is of the form:

String current View = request.getParameter(“PortletStatus”).

Different actions may be performed according to the different possible view modes. The currentView is used to switch between the different views (VIEW_ACTIVATE, VIEW_INPUT, VIEW_SUBMIT, and VIEW_PILOT) of the Infection Model portlet. When the currentView of the portlet is VIEW_INPUT, the PortletRequestDispatcher object, (dispatcher) is used to display the input.jsp page (the view page described earlier) using the getRequestDispatcher() method. Similarly, when the currentView of the portlet is VIEW_SUBMIT, the PortletRequestDispatcher object (dispatcher) is used to display the submit.jsp page.

*The doEdit() method*

This method will display the current preference values which are stored in the AppPreferences class and the AppInfrastructureInfo class. It will obtain the current preference values, the current infrastructure and the total number of infrastructures. The actionURL and the current preference may then be passed to the edit.jsp page (described in sub-section 5.7.1) and the preference values and the infrastructure data will then be sent. The PortletRequestDispatcher object (dispatcher) is used to display the edit.jsp page using the getRequestDispatcher() method.

*The doHelp() method*

This method is used to call the help.jsp page which is responsible for displaying the portlet information. The PortletRequestDispatcher object (dispatcher) is used to display the help.jsp page simply by using the getRequestDispatcher() method.

#### 5.7.2.8 The interaction between the methods of the myRepast-infection-portlet class

The Infection Model Java code (myRepast-infection-portlet) extends the GenericPortlets class and overrides methods such as init(), processAction() and the render() methods (doView(), doEdit(), doHelp()). It makes use of two main methods to exchange data to and from the JSP pages. This exchange occurs between the processAction() and the render() methods as seen in Figure 5.2. These methods enable the exchange of parameters between the user and a portlet. The processAction() method is responsible for the action being performed by the user in the input forms and the render() method determines the interface that is being shown to the user as a consequence of the actions being performed on the input page. Elements of the processAction() and the render() methods include the ActionRequest and the RenderRequest, respectively. The processAction() receives an input parameter using the ActionRequest and prepares the render object for the view methods. The RenderRequest on the other hand is the inputs for the view methods.

#### 5.7.2.9 The interaction between the myRepast-Infection portlet class and the ABMS application JSP pages

To enable an interaction between the JSP pages of the Infection Model portlet and the Java code, form statements are used to send parameters between the Java code (myRepast-infection-portlet) and the JSP pages of the Infection Model portlet. This interaction sees a continuous data exchange between the myRepast-infection-portlet class and the JSP pages (which present the necessary user interface of the ABMS application back to the user) and this interaction occur when users make use of the Infection Model portlet. All the java input fields are placed in the JSP code web form to implement the flow of data from the JSP pages of the application to the java code. This is illustrated as <form action=<portlet: actionURL portlet Mode=view> In addition, within the java code (myRepast-infection-portletclass) the input interface values are obtained with the methods: doView/doHelp/doEdit (RenderRequest request...)

In order to obtain the parameters, the string param i= request.getParameter(param name i) is set, where param name i is the portlet status and param i is the current view. For the exchange of data between the Java code and the JSP pages, the input interface values inside the Java code are obtained using doView()/doHelp()/doEdit() (RenderRequest request) and to obtain the parameters, we just set string param i=request.setAttribute (param name i, param value I) Within the JSP page, the parameter values are loaded with <jsp:useBean id=param name k class=<variable type k>scope=request>

### 5.7.2.10 Method that submits the Infection Model Portlet jobs to different DCIs

This section discusses how myRepast-infection-portlet of the Infection Model submits jobs to different DCIs using the Catania Grid and Cloud Engine. One advantage of the CSGF is that jobs are submitted to different DCIs through the Grid and Cloud Engine simply by using the Java implementation of the Simple API for Grid Access (JSAGA) which abstracts the different middleware layers. This ensures that users can simply submit and execute jobs without having to know the details of the underlying infrastructures and middlewares. To submit a job, the submitJob() method of the GridEngine is defined and the Multi-Infrastructure Job Submission object is initialised as shown in Appendix C4. The GridEngine uses two different types of constructors. The constructor that takes no database arguments is used for production environments, while the constructor which takes the SciGwyUserTrackingDB parameters (below) is normally used for development purposes. In order to switch-on the constructor which is used by the production environment, the portlet parameters in Appendix C5 are set to an empty string. The GridEngine Multi-Infrastructure Job Submission object for the production environment is therefore initialised as:

```
miJobSubmission = new MultiInfrastructureJobSubmission();
```

Having set the portlet parameters to an empty string, the constructor is simply set to void. However, for the development environment, the different portlet parameters which constitutes the database requirements are described in Appendix C6. The variables described in Appendix C6 are defined for the required database parameters and passed as arguments to the constructor and initialised as follows:

```
miJobSubmission = new MultiInfrastructureJobSubmission(arg1,arg2,arg3);
```

Furthermore, the GridEngine job description object (jobDesc) is defined and used in specifying the executable, application' arguments, the output directory, the std-output file, the std-error file, output files and the input files as illustrated in Appendix C7. The GEJobDescription object (jobDesc) for the Infection Model portlet is therefore specified as arguments within the GridEngine Multi-Infrastructure Job Submission object for both production and development environments, respectively as:

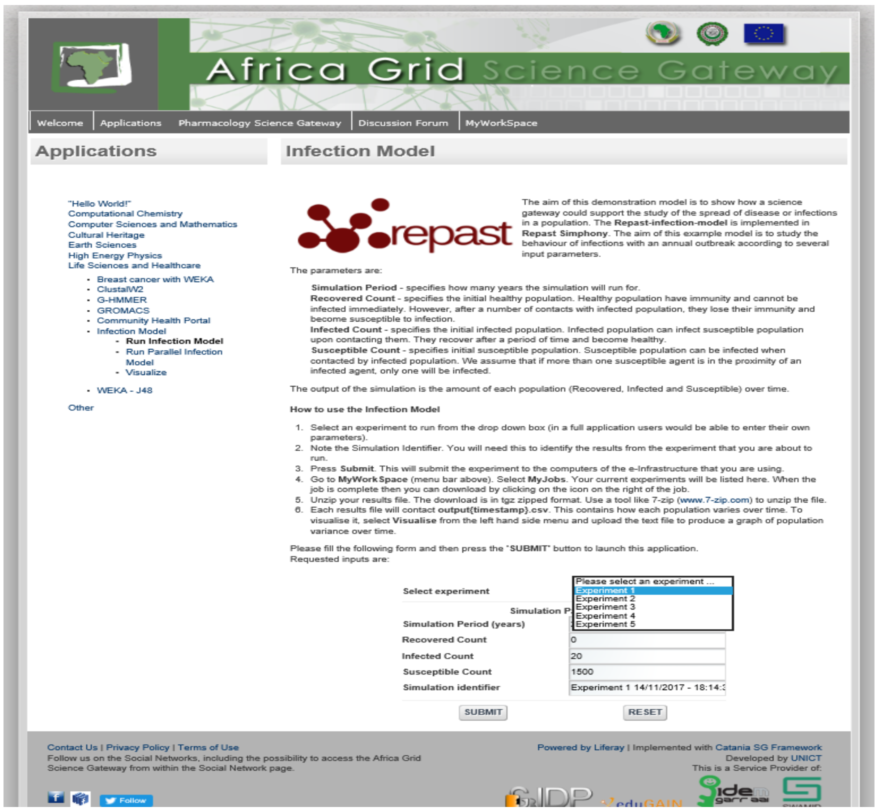
```
miJobSubmission = new
MultiInfrastructureJobSubmission(jobDesc);
miJobSubmission = new
MultiInfrastructureJobSubmission(arg1,arg2,arg3,jobDesc);
```

In addition, the portal IP address required by the UserTrackingDB is obtained. All enabled infrastructures are assigned to the Infection Model portlet and the details required to submit jobs (such as the application executable, executable arguments, etc) are defined. Furthermore, the GridEngine Multi-Infrastructure Job Submission object (miJobSubmission) is used to specify the Infection Model portlet job initialisation settings. Once the requirements necessary for the submission of the Infection Model portlet has been defined and its settings specified, the job is then ready for submission. This submission is done simply by using the submit obAsync() method of the GridEngine Multi-Infrastructure Job Submission where the necessary details are passed as arguments as illustrated in Appendix C8.

### 5.7.2.11 Deploy the Portlet (8)

In order to deploy the portlet, a pre-configured Virtual Machine (VM) image is provisioned with the Infection Model together with all its dependencies (such as the REPAST libraries and the Java runtime installation). These dependencies are deployed within the pre-configured VM. The Pilot_script.sh (See appendix A2) consisting of the instructions to start a simulation resides in the portlet context and is uploaded to the remote VM as soon as it becomes available. This image is used to start a VM on a remote cloud site from the several infrastructures available in the portlet configuration whenever the user submits a simulation job via the Science Gateway. The VM image is started whenever a simulation job is submitted and, according to the parameters being passed to the job, the execution is performed.

The final stage of the deployment process involves the creation of war files from the main Java class of the Infection Model and the different JSP pages described above. This war file will run on a glassfish domain under the Liferay portal. From the control panel (on the Liferay portal) the plugins installation page is used to install a new portlet (which in this case is the Infection Model portlet).



Applications

"Hello World!"
Computational Chemistry
Computer Sciences and Mathematics
Cultural Heritage
Earth Sciences
High Energy Physics
Life Sciences and Healthcare

- Breast cancer with WEKA
- ClustalW2
- G-HMMER
- GROMACS
- Community Health Portal
- Infection Model
    - **Run Infection Model**
    - Run Parallel Infection Model
    - Visualize
- WEKA - J48

Other

Infection Model

The aim of this demonstration model is to show how a science gateway could support the study of the spread of disease or infections in a population. The **Repast-infection-model** is implemented in **Repast Simphony**. The aim of this example model is to study the behaviour of infections with an annual outbreak according to several input parameters.

The parameters are:

**Simulation Period** - specifies how many years the simulation will run for.
**Recovered Count** - specifies the initial healthy population. Healthy population have immunity and cannot be infected immediately. However, after a number of contacts with infected population, they lose their immunity and become susceptible to infection.
**Infected Count** - specifies the initial infected population. Infected population can infect susceptible population upon contacting them. They recover after a period of time and become healthy.
**Susceptible Count** - specifies initial susceptible population. Susceptible population can be infected when contacted by infected population. We assume that if more than one susceptible agent is in the proximity of an infected agent, only one will be infected.

The output of the simulation is the amount of each population (Recovered, Infected and Susceptible) over time.

**How to use the Infection Model**

1. Select an experiment to run from the drop down box (in a full application users would be able to enter their own parameters).
2. Note the Simulation Identifier. You will need this to identify the results from the experiment that you are about to run.
3. Press **Submit**. This will submit the experiment to the computers of the e-Infrastructure that you are using.
4. Go to **MyWorkSpace** (menu bar above). Select **MyJobs**. Your current experiments will be listed here. When the job is complete then you can download by clicking on the icon on the right of the job.
5. Unzip your results file. The download is in tgz zipped format. Use a tool like 7-zip (www.7-zip.com) to unzip the file.
6. Each results file will contact **output{timestamp}.csv**. This contains how each population varies over time. To visualise it, select **Visualise** from the left hand side menu and upload the text file to produce a graph of population variance over time.

Please fill the following form and then press the "**SUBMIT**" button to launch this application.
Requested inputs are:

| | |
|---|---|
| **Select experiment** | Please select an experiment ... / [✓ Experiment 1] / Experiment 2 / Experiment 3 / Experiment 4 / Experiment 5 |
| **Simulation P** | |
| **Simulation Period (years)** | |
| **Recovered Count** | 0 |
| **Infected Count** | 20 |
| **Susceptible Count** | 1500 |
| **Simulation identifier** | Experiment 1 14/11/2017 - 18:14: |

Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid Science Gateway from within the Social Network page.

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT
This is a Service Provider of:

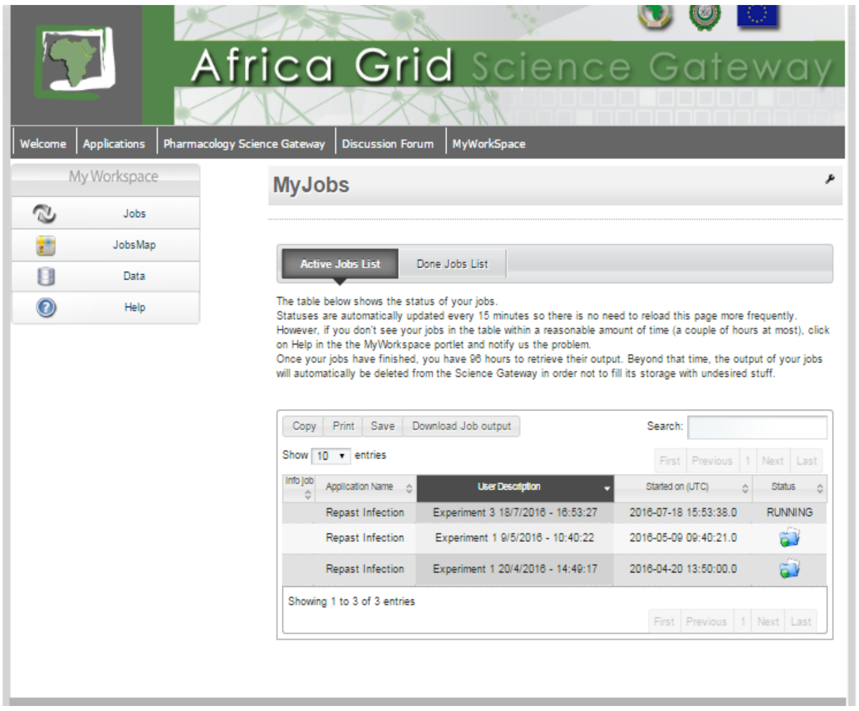
Figure 5. 3 Infection Model Application Portlet Main Page

## *5.8* Test the Portlet (9)

The Infection Model has been deployed in a portlet known as Infection Model and, along with other portlets belonging to different communities of practice, ported on the Africa Grid Science Gateway (See Figure 5.3). The Africa Grid Science Gateway is a standard based web

2.0 demonstrative platform that hosts the portlets developed the different communities of practice to be executed on a worldwide e-Infrastructure. The Infection Model portlet will aid potential users in conducting experiments by specifying input parameters (such as the ones described in Section 5.2), running experiments, monitoring and obtaining results, efficiently. The application also has a demonstration graph tool that allows users to analyse the results of an output file and thus see the graphical visualisation of their results. This ultimately shows that Science Gateways can be developed and used to support complex simulations in an incredibly easy to use manner.

As mentioned earlier, procedures for both the authentication and authorisation to the Africa Grid Science Gateway involves a standard which is similar to all the embedded applications. A user of the Science Gateway can access its main page by using the web link at https://sgw.africa-grid.org/. To access and make use of any application on the Science Gateway, such as the Infection Model portlet, users have to send a request to obtain federated credentials issued by identity providers. If a request is granted, users can sign-on to the Science Gateway and run jobs (via the top right corner of the Science Gateway) simply by using their new federated credentials. This will present a user with the page consisting of the different identity federations and they can choose the one they belong. Furthermore, the user is re-directed to another page composed of various identity providers. Different identity providers have been embedded within a particular identity federation and (similar to the identity federations) users can select the identity provider which they belong from their respective identity federation. When a user has successfully obtained a federated credential, it may be used to access the Science Gateway from the log-in page. After each successfully log-on, users can access the desired application such as the Infection Model portlet at https://sgw.africa-grid.org/infection-model which is shown in Figure 5.3.



MyJobs

Active Jobs List | Done Jobs List

The table below shows the status of your jobs.
Statuses are automatically updated every 15 minutes so there is no need to reload this page more frequently.
However, if you don't see your jobs in the table within a reasonable amount of time (a couple of hours at most), click on Help in the the MyWorkspace portlet and notify us the problem.
Once your jobs have finished, you have 96 hours to retrieve their output. Beyond that time, the output of your jobs will automatically be deleted from the Science Gateway in order not to fill its storage with undesired stuff.

Copy | Print | Save | Download Job output    Search:

Show 10 entries    First Previous 1 Next Last

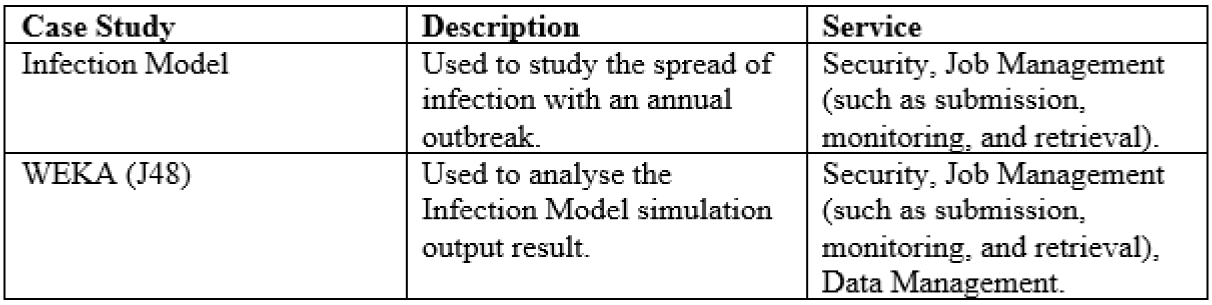
| Info job | Application Name | User Description | Started on (UTC) | Status |
|---|---|---|---|---|
| | Repast Infection | Experiment 3 18/7/2016 - 16:53:27 | 2016-07-18 15:53:38.0 | RUNNING |
| | Repast Infection | Experiment 1 9/5/2016 - 10:40:22 | 2016-05-09 09:40:21.0 | 📁 |
| | Repast Infection | Experiment 1 20/4/2016 - 14:49:17 | 2016-04-20 13:50:00.0 | 📁 |

Showing 1 to 3 of 3 entries    First Previous 1 Next Last

Figure 5. 4 MyJobs portlet showing an ABMS job in RUNNING and DONE mode

The first page that is presented after a successful login to the Science Gateway is the landing page of the desired application (See Figure A-7). This is where the description of the Infection Model portlet can be found. The run icon at the bottom corner of the page presents the main page of the Infection Model portlet at https://sgw.africa-grid.org/run-repast where users can specify all the input parameters defined in Section 5.2 and run their experiments as seen in Figure 5.3. It is worth noting that the page can only be accessed by users who have successfully completed the sign-on process which was earlier described. After the parameters have been specified the experiment is run by clicking the provisioned submit button. Consequently, the Grid and Cloud engine, using the JSAGA, can submit the jobs on different DCIs without users knowing the details of the implementation of the underlying middleware. After each submission, users are notified that jobs have been successfully submitted. To check the status of each jobs on the MyJobs portlet, a dedicated portlet where the statuses of all “RUNNING” jobs and “DONE” jobs (as shown in Figure 5.4) can be found. A “DONE” job status is represented by a small folder icon indicating that the job is ready for collection and users can download the Infection Model output results for subsequent analysis.

Table 5.1 Services Utilised by the Different Case Studies



| **Case Study** | **Description** | **Service** |
|---|---|---|
| Infection Model | Used to study the spread of infection with an annual outbreak. | Security, Job Management (such as submission, monitoring, and retrieval). |
| WEKA (J48) | Used to analyse the Infection Model simulation output result. | Security, Job Management (such as submission, monitoring, and retrieval), Data Management. |

## *5.9 Case Study Overview – WEKA (J48) Classifier (1)*

In this section, a second case study is ported to the Science Gateway. This case study is the Waikato Environment for Knowledge Analysis (WEKA). More specifically, the J48 algorithm of WEKA is analysed, designed, implemented and ported on the Science Gateway to further examine and implement some of the services that were identified in the literature. The J48 algorithm is utilised (via the Science Gateway) to analyse any simulation output result obtained from the Infection Model portlet described in the earlier sections. This sequence of events can potentially be used to demonstrate how Science Gateways may be used to support workflow management. WEKA is a state-of-the-art facility designed to aid in the application of machine learning techniques to real-world data sets (Garner 1995). It is an open source software for data mining under the GNU General public licence. It was developed at the University of Waikato in New Zealand to be open source and is thus, freely available. It consists of different machine learning algorithms for data mining tasks. Data mining which involves the systematic analysis of large data set is a technique that is often used to drill a database to extract meaning from available data (Bhargava, Sharma et al. 2013). The data required by WEKA may be obtained from various sources such as files, URLs and databases and it supports different file formats such as ARFF (which is WEKA's file format), CSV, LibSVM's format, and C4.5's format (Hall, Frank et al. 2009). Using the WEKA interface, users can perform operations such as pre-processing, association, filtering, classification, clustering visualisation and regression. These operations constitute the various techniques/algorithms that are available for data mining. As such, the WEKA system is a collection of inter-dependent programs which are bound together by a common user interface. These programs are categorised into data set processing, machine learning schemes and output processing (Garner 1995). The consequence of having a generic interface that WEKA provides (for different operations and techniques required for data mining tasks) is that wealth of interactive tools for data manipulation, result visualisation, database linkage, and cross-validation and comparison of rule sets are readily available to complement the basic machine learning tools (Holmes, Donkin et al. 1994).

For the purpose of this study a classification algorithm, known as the J48 decision tree-inducing algorithm (WEKA implementation of C4.5) is explored. There are some prominent machine learning algorithms that are used in modern computing applications and a common use for these algorithms often involve decision-based classification and adaptive learning over a training set (Drazin, Montag 2012). The J48 Algorithm which is one of such prominent machine learning algorithms is a decision tree that serves as a decision modelling tool which graphically displays the classification process of a given input for a given output class labels. In data mining such decision tree models are used to examine the data and induce both the tree and its rules used in making predictions (Bresfelean 2007). Decision trees such as the J48 are commonly used to classify data into distinct groups which generate the strongest separation in the values of the dependent variable and thereby providing an easily interpretable separation. Another commonly used decision tree construction algorithm is the ID3 (an attribute-based machine learning algorithm) which creates a decision on a training set of data. The C4.5 algorithm is an evolution of the ID3 algorithm which contains supplementary programming that helps to address the issues encountered in the ID3 algorithm. However, this study will primarily focus on how J48 algorithm (via a Science Gateway interface) is applied to the Infection Model simulation output results or any other real-world data sets and executed in a distributed environment via Science Gateway interfaces.

Consequently, the system design and implementation of the J48 portlet will ensure that users/scientists can upload their output file to the Science Gateway, select the necessary filters and set the appropriate test options. Finally, the J48 algorithm can be applied to data sets executed on the distributed infrastructure. Similar to the Infection Model portlet, this design is achieved by capturing all the functionalities associated with the J48 algorithm in a user interface (portlet), which is then ported on the Science Gateway. Furthermore, all the dependencies and libraries associated with the J48 algorithm is deployed in a pre-configured VM on the infrastructure. The VM is dedicated to executing all WEKA related jobs within a distributed environment.

## *5.10 Define Portlet Functionalities (2)*

This section constitutes the functionalities of WEKA portlet from the users' point of view as well as defines the different user requirements which justify the need to have a portlet for performing WEKA analysis. This sort of analysis can either be performed on the output of the Infection Model portlet obtained from executing an ABMS job on DCIs (via the Science Gateway), or any other related datasets. In order to specify ways in which a Science Gateway may be used to support WEKA analysis, the different ways of using WEKA must be identified. Usually, a user should be able to upload a file via the portlet user interface, select the appropriate filters and test options associated with WEKA and apply the required algorithm to each dataset (which in this case is the J48). Based on the user requirements specification for this portlet therefore, the following deductions are made. A potential user will:

- Be able to easily access the WEKA portlet page on the Science Gateway.
- Be able to upload the Infection Model simulation output result or any other related datasets (via the Science Gateway) for WEKA analysis.
- Be able to select appropriate filters, test options and algorithms associated with WEKA analysis and apply to the uploaded dataset.
- Be able to perform and submit these jobs on DCIs.
- Be able to monitor jobs for statuses and updates.
- Be able to download outputs/results of WEKA experiments.

All the aforementioned user requirements are incorporated at the graphical user interface of the WEKA portlet and ported for use on the Africa Grid Science Gateway. The processes involved in the design and implementation of this portlet are discussed in the following section.

## *5.11 Requirements*

This section will discuss the requirements for the realisation of a fully functional WEKA portlet. It is important to note here that the WEKA portlet will make use of a similar Science Gateway framework used to develop the Infection Model portlet which provides a standard set of requirements for the software, system and Science Gateway for all its applications. This Science Gateway framework (otherwise known as the futuregateway API) is an evolution of the CSGF. The different requirements (such as software (3.1), hardware (3.2) and Science Gateway (3.3)) which was defined for the Infection Model portlet in Section 5.5 therefore apply.

## *5.12 WEKA – J48 Portlet Design (6)*

The WEKA - J48 portlet design is similar to the design processes described for the Infection Model portlet. However, this design will include additional functionalities such as the upload of files, selection of filters and test options as well as the application of the J48 algorithm. Therefore, the different stages that will realise these functionalities (in addition to those defined for the Infection Model portlet) are as follows:

1) The user seeks to access the Science Gateway.
2) Science Gateway takes the user through an authentication and authorisation procedures.
3) A user submits his/her credentials to the Science Gateway for access.
4) Based on the user roles and privileges, Science Gateway presents the desired portlet/application to the user (J48 portlet).
5) The user can view the portlet, upload files, select the appropriate filters, test option and algorithm and submit jobs to e-Infrastructure.
6) Science Gateway (using a pilot script) can collect the input parameters specified via the user interface and submits job to e-Infrastructure.
7) Science Gateway gives job statuses and users can monitor and download the outputs of jobs.
8) Procedure ends.

From the above sequence of steps, five main building blocks (which are similar to the Infection Model portlet) are derived to represent and implement the initial prototype architecture. These include user access, user interface, pilot-script.sh, pre-configured virtual machine, and myjobs portlet. The steps above are similar to the ones defined for the Infection Model portlet, except for step (5) where users also need to view portlet, upload input data, select appropriate filters, test option and algorithm and submit jobs to e-Infrastructure. Step (5) and (6) captures the execution process defined in Figure 5.5 (i.e. upload files, select the appropriate filters, choose test option and algorithm and the final job execution on DCIs as well as the use of a pilot script to collect specified input parameters).

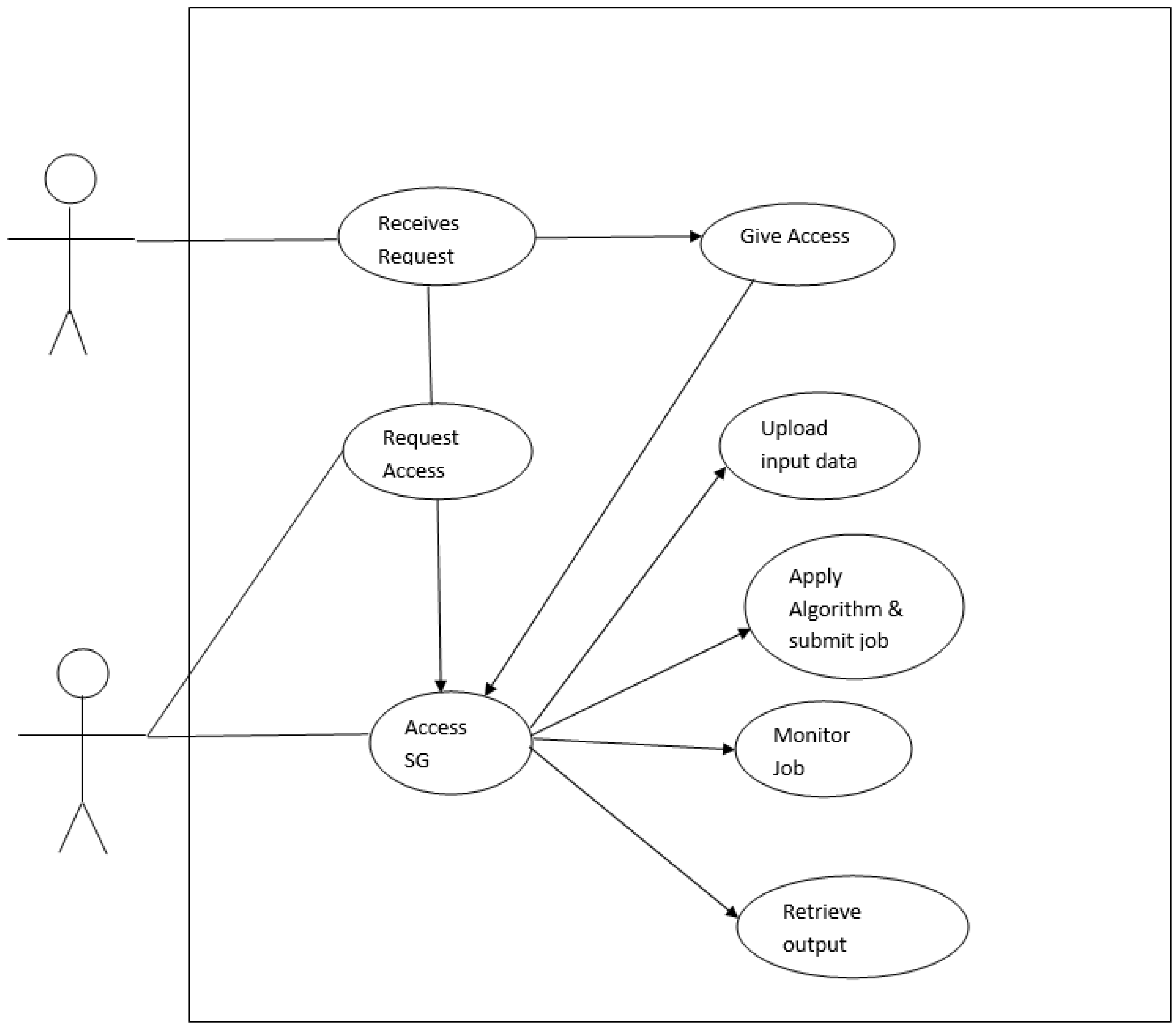


Figure 5. 5 WEKA Use Case Diagram

## *5.13 Develop/Implement the WEKA – J48 Portlet (7)*

It was earlier mentioned that the software and hardware requirements for the WEKA - J48 case portlet are similar to the requirements that were defined for the Infection Model portlet in Section 5.5 above. The design process of the WEKA - J48 portlet as well as the adopted Science Gateway Framework have therefore been discussed in Section 5.5. The next step will describe the implementation of the portlet using the Future Gateway technologies (which is an evolution of the CSGF). In the Future Gateway technologies however, portal technologies are not tightly coupled with the Science Gateway framework. Developers may utilise existing portal or any web technologies which is available to them. However, similar to the Infection Model portlet, the Liferay portal framework was utilised for the implementation of the WEKA - J48 Classifier, although the Future Gateway provides a solution which is portal independent and communities of practice may utilise existing portal or technology that already serves their branch of science. The WEKA – J48 was deployed in a portlet named WEKA - J48 (See https://sgw.africa-grid.org/j48). It utilises several classes such as the AppPreferences class (for storing all the values of portlet preferences), the ConfigurationActionImpl class, the FutureGatewayClient class, the Response class and most importantly the WJ48Portlet class (which is the main portlet class that extends and overrides the GenericPortlet class methods). It also defines additional java classes such as the AppInput class, the InputFile class, the Link class, the OutputFile class, and the Task class. In addition to the different classes used by the WEKA – J48 portlet, different Java Server Pages (JSP) are used for presenting the desired views to the Science Gateway user. This primarily includes the view page (which presents the user interface of the portlet) and the submit page (which is used for informing users of each successful job submission).

### 5.13.1 WEKA (J48) Classifier Portlet Modes

The following sections will discuss in detail the implementation of the WEKA – J48 portlet by expanding on both the aforementioned classes and the different JSP pages that were utilised in the implementation of the portlet. Even though the portlet specification defines three main portlet modes (VIEW, EDIT and HELP), the WEKA – J48 portlet is designed to include only two modes (i.e. the view and the submit pages).

#### 5.13.1.1 The View Page

The view page consists of the user interface of the J48 portlet. It incorporates the basic user requirements needed for WEKA analysis as specified at the user requirements stage. This page starts by defining the general use of the portlet as well as a general description of the WEKA application. It will utilise the multi-part form to provision the upload of data sets or the output files which are required for WEKA analysis. Furthermore, it facilitates the selection of the desired classifier that users may need to apply on data sets (which in this case is the J48 classifier) as well as the test option necessary for the analysis. It also facilitates a submit button for the submission of jobs to the different DCIs. This page was developed and utilised both at the Science Gateway level (production) and at the portlet developmental stage.

#### 5.13.1.2 The Submit Page

The submit page is mandatory and implemented both at the Science Gateway level and the development stage of the portlet. Similar to the Infection Model portlet, the submit page of the WEKA – J48 portlet informs the user of any successful job submission being made towards the DCIs.

### 5.13.2 WEKA - J48 portlet Classes

There are several Java classes used in the implementation of the JSON files that are required for the execution of the WEKA application. These include the AppPreferences Class, the ConfigurationActionImpl Class, the FutureGatewayClient Class, the Response Class and the WJ48Portlet class. Other classes which are specific to the execution of jobs (in a distributed environment) include the AppInput Class, the InputFile Class, the Link Class, the OutputFile class and the Task Class.

#### 5.13.2.1 The AppPreferences

The AppPreferences class defines all the values needed by the portlet to submit jobs to the Future gateway. This class stores information such as the FutureGateway Host (fgHost), FutureGateway Port (fgPort), FutureGateway API version (fgAPIVersion), Application Id (applicationid) and the pilotScript. The class defines several methods to set and get the different values which are required to submit jobs towards a distributed environment.

#### 5.13.2.2 ConfigurationActionImpl

This class consists of the exchanges between the processAction and the render method. It defines a method that is used to store all the application preferences such as the fgHost, fgPort, fgAPIVersion, application id and the pilotscript.

#### 5.13.2.3 FutureGatewayClient

The FutureGatewayClient defines the attributes of a particular client which are required by the target host (i.e. FutureGateway). This may include the fgEndpoint, fgAPIversion, the WebResource (apiResource), and the client. It also defines the different set() and get() methods which aid in submitting jobs in a distributed environment. In addition, this class has a createTask() method (that accepts a parameter such as appInput) which will call the create task API and thus pass the input file. It defines a method to return the id of a specific task as defined in the task class.

#### 5.13.2.4 AppInput Class

This class is the representation of the task to be performed which contains all the necessary attributes needed by the FutureGateway to submit WEKA jobs within a distributed infrastructure. The FutureGateway API returns a JSON file whenever a task (consisting of the AppInput class) is submitted. This JSON file represents an instance of the task class. Each submission to the distributed infrastructure will instantiate these attributes. These attributes include the Application (which is an identifier for the application in the FutureGateway), Description (used to describe the application), Arguments, Input_files (files that are needed by the application) and the Output_files (which consists of files created by the application). It also consists of methods for getting and setting the aforementioned AppInput objects. A JSON property is included to instruct the library that get() method will create outputs in JSON format.

#### 5.13.2.5 InputFile

Similar to the AppInput, this class also contains the necessary attributes that the FutureGateway utilise to execute jobs in a distributed environment. This class has two major attributes such as the name (which is used to specify the name of the input file that is uploaded) and the status. It also defines the set() and get() methods that enable the retrieval of these attributes.

#### 5.13.2.6 OutputFile Class

The attributes of the OutputFile class is returned by the FutureGateway API's. These attributes include: Name (which is the name of the input file which has been specified within the InputFile) and the URL (which is returned by the FutureGateway when each output is ready). The set () and get () methods for both Name and URL (which are used for retrieving attributes) are also defined within the class and the FutureGateway generates the required URL when the outputFile is ready.

#### 5.13.2.7 Task

The task also includes all the required attributes needed by the FutureGateway to submit jobs within a distributed environment. These attributes are returned to the user by the FutureGateway API. These attributes include the Status, Application, Date, Description, Output_Files, Links, User, Input_Files, id, and arguments. Within the class, the different set () and get () methods of the attributes are defined and utilised by the FutureGateway API's. The Status attribute is important as it represents the status of each task. When a job is submitted, it is equivalent to creating a task in the FutureGateway which in turn has a Status.

#### 5.13.2.8 WJ48Portlet Class

This is the main Java class of the WEKA - J48 portlet. This class extends the generic portlet class and overrides the different class methods.

### 5.13.3 Execution of the WEKA application to distributed infrastructure using the FutureGateway API

FutureGateway APIs are services that enable the interaction of web applications with the associated back-end distributed infrastructures, thus providing the necessary platform for the front-end web services and the back-end interfaces. A simple demonstration of how the FutureGateway API's is used to submit and execute a WEKA - J48 job in distributed environment is seen in the analysis below.

To submit a WEKA - J48 job, a JSON file containing the description of the application is defined. A CURL command is then used to send this file to the FutureGateway API which is described in Appendix C9.

From the illustration in Appendix C9, the CURL command (using the POST method) is used to send the JSON file which describes the WEKA application to the FutureGateway. The JSON file consists of the application to be performed, the description of the application, input_files needed by the FutureGateway, the arguments necessary to execute the application and the expected output_files that is returned by the FutureGateway. All the aforementioned attributes of the JSON file are defined within the Task Class (defined above) as well as the AppInput Class. As such, this JSON file represents an instance of the task class. Generally, whenever an application needs to be sent to the FutureGateway, it takes the format defined above, however, the arguments depend on the type of analysis to be performed. For the above

WEKA application, the missing value filters, weather.data and weka.classifiers.J48 classifiers have been passed as arguments. Also, the InputFile class has two main attributes such as the Name and the Status of the application. In the JSON file above the name of the input file that was uploaded is weather.arff. This indicates to the FutureGateway to expect an input file named weather.arff.

It was earlier noted that the FutureGateway returns an output each time a JSON file is passed using the CURL command. This command is used to make an HTTP request to the FutureGateway server. Once the weather.arff file is passed to the FutureGateway using the CURL command (above) the output which the FutureGateway returns is illustrated in Appendix C10.

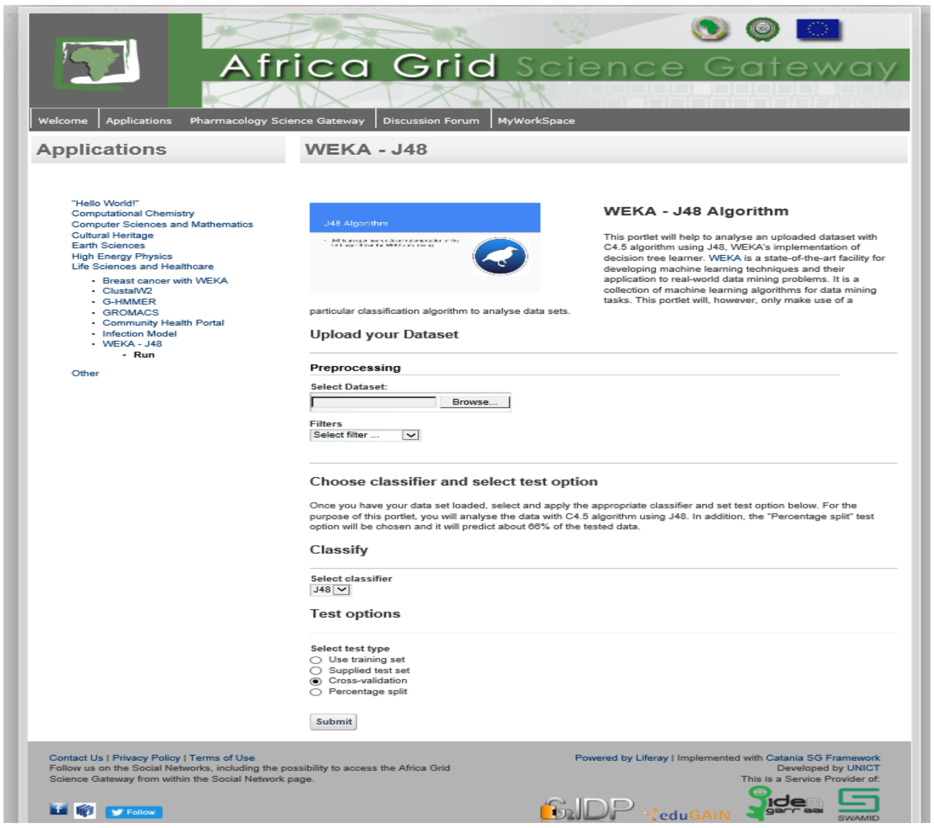
The output returned by the FutureGateway is a JSON file that contains the attributes of the WEKA Task Class. From the FutureGateway output, the status of the Task is given as “WAITING”. This is as a result of the status of the weather.arff file which is returned as: “NEEDED” even though the status of the pilot_script.sh file has already returned: “READY”. The weather.arff file whose status is “NEEDED” is the file that needs to be uploaded to the FutureGateway server and until this file is available, the Task can not start executing, and hence the status will remain as “WAITING”. Once the weather.arff file has been uploaded (via the WJ48 portlet) on the Science Gateway and the submit button has been utilised, this will call the submit process action that creates the AppInput. Once the input file (weather.arff) has been uploaded and ready and the status is no longer “WAITING” the CURL command can then be used to submit the file to the FutureGateway. This is illustrated below as:

```
curl -i -F "file[]=@weather.arff"
"http://151.97.41.48/v1.0/tasks/103/input?user=afabiyi"
```

Therefore, by specifying the input sandbox consisting of all the files which may be uploaded and by calling the upload file method in the WJ48 portlet class, the weather.aff file is uploaded to the FutureGateway. The deployment (8) of the WEKA – J48 portlet on the Liferay portal (i.e. the generation and execution of WEKA war files to run on a glassfish domain under the Liferay portal) is the same as the procedures described for the Infection Model portlet above.

## *5.14* Test the Portlet (9)

The WEKA application has been deployed in a portlet known as WEKA - J48 portlet. It is ported on the Africa Grid Science Gateway along with the Infection Model portlet and other portlets belonging to different communities of practice (See Figure 5.6). Also, this portlet can be accessed at https://sgw.africa-grid.org/j48.


Africa Grid Science Gateway

Welcome | Applications | Pharmacology Science Gateway | Discussion Forum | MyWorkSpace

**Applications**

"Hello World!"
Computational Chemistry
Computer Sciences and Mathematics
Cultural Heritage
Earth Sciences
High Energy Physics
Life Sciences and Healthcare
- Breast cancer with WEKA
- ClustalW2
- G-HMMER
- GROMACS
- Community Health Portal
- Infection Model
- WEKA - J48
  - **Run**

Other

**WEKA - J48**

J48 Algorithm

**WEKA - J48 Algorithm**

This portlet will help to analyse an uploaded dataset with C4.5 algorithm using J48, WEKA's implementation of decision tree learner. WEKA is a state-of-the-art facility for developing machine learning techniques and their application to real-world data mining problems. It is a collection of machine learning algorithms for data mining tasks. This portlet will, however, only make use of a particular classification algorithm to analyse data sets.

**Upload your Dataset**

**Preprocessing**

Select Dataset: [ ] Browse...

Filters: Select filter ...

**Choose classifier and select test option**

Once you have your data set loaded, select and apply the appropriate classifier and set test option below. For the purpose of this portlet, you will analyse the data with C4.5 algorithm using J48. In addition, the "Percentage split" test option will be chosen and it will predict about 66% of the tested data.

**Classify**

Select classifier: J48

**Test options**

Select test type
- Use training set
- Supplied test set
- [✓ Cross-validation]
- Percentage split

Submit

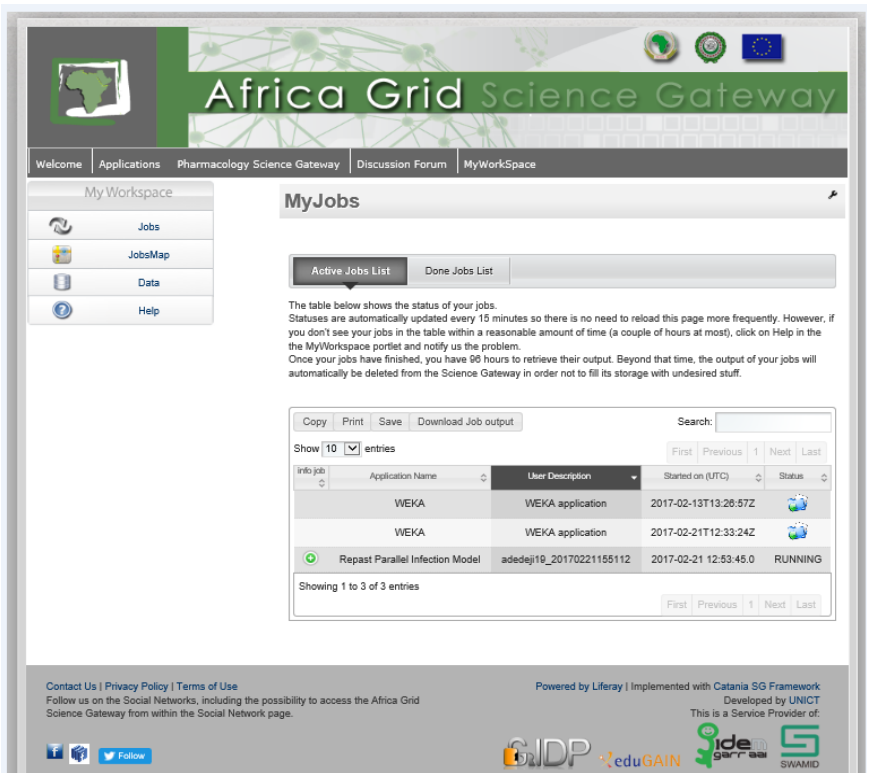
Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid Science Gateway from within the Social Network page.

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT
This is a Service Provider of:

Figure 5. 6 WEKA (J48) Application Portlet Page

This portlet may be used by scientists to analyse specific datasets with the C4.5 algorithm using J48 which is WEKA's implementation of decision tree learner. More specifically, the generated dataset which is obtained from the Infection Model portlet is analysed using the WEKA – J48 and thus performing data mining operation on the output of simulation results. The portlet could help users conduct their experiments by using the functionalities (i.e. file upload, filters, desired classifier, test option, and job submission) as defined on the view page described above.

Similar to the Infection Model Portlet and other portlets which has been embedded on the Africa Grid Science Gateway, the process of authentication and authorisation to the portlet has already been explained in great detail in Section 5.8 above. When a user has successfully logged on to the Science Gateway, the WEKA - J48 portlet may be accessed at https://sgw.africa-grid.org/run-j48 and used to performed data mining on a given data set. Such analysis may only be performed on .arff and .csv file formats. The landing page of the portlet presents a general description of the application and the run icon at the bottom corner is the link to the main page of the application. The WEKA – J48 portlet main page in Figure 5.6 includes all the functionalities needed to submit and perform data mining operations on Infection Model simulation output result. Similar to the Infection Model portlet, users are notified after each successful job submission and results is retrieved from the MyJobs portlet. This stores the status of the WEKA jobs in a "RUNNING" mode as well as "COMPLETED", respectively as shown in Figure 5.7. In addition, the analysis of the simulation output result using the WEKA – J48 portlet as well as the discussion of the output analysis can be found in Appendix G.



MyJobs

Active Jobs List | Done Jobs List

The table below shows the status of your jobs.
Statuses are automatically updated every 15 minutes so there is no need to reload this page more frequently. However, if you don't see your jobs in the table within a reasonable amount of time (a couple of hours at most), click on Help in the the MyWorkspace portlet and notify us the problem.
Once your jobs have finished, you have 96 hours to retrieve their output. Beyond that time, the output of your jobs will automatically be deleted from the Science Gateway in order not to fill its storage with undesired stuff.

Copy | Print | Save | Download Job output    Search:

Show 10 entries    First | Previous | 1 | Next | Last

| info job | Application Name | User Description | Started on (UTC) | Status |
|---|---|---|---|---|
| | WEKA | WEKA application | 2017-02-13T13:26:57Z | |
| | WEKA | WEKA application | 2017-02-21T12:33:24Z | |
| ⊕ | Repast Parallel Infection Model | adedeji19_20170221155112 | 2017-02-21 12:53:45.0 | RUNNING |

Showing 1 to 3 of 3 entries    First | Previous | 1 | Next | Last

Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid Science Gateway from within the Social Network page.

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT
This is a Service Provider of:

Figure 5. 7 Myjobs portlet showing the output of the WEKA - J48 jobs in RUNNING and COMPLETED modes.

## *5.15 Case Study Overview (1) - A Visualisation Portlet for the Infection Model Simulation Output Results*

Visualisation is a desirable feature to have within Science Gateways. It often helps communities of practice in examining the outcome of their experiments by visualising input/output data for the purpose of generating and extracting new insight and information based on the output/result of a job. More so, applications solving complex problems could generate terabytes or even petabytes of datasets and such volume of data therefore pose significant challenges for data analysts. Scientific visualisation therefore represents a crucial step in understanding, interpreting and verification of the outcome. It also accounts for a process which includes the qualitative, comparative, as well as the quantitative stages of data analysis, result presentation and the public engagement that ensues. Science Gateways can therefore be used as the medium to seamlessly integrate datasets as well as the tools needed for its interactive visualisation. The ABMS visualiser thus presents an example of how CSGF can visualise the results or outcome of a scientific experiment and provision it via the Science Gateway.

To facilitate data analysis activities on each simulation output result of the Infection Model, a simple visualisation tool is provisioned on the Science Gateway. This tool will enable users to create a visualisation of the simulation output results obtained from the execution of the Infection Model portlet. For each simulation run that is performed on the Science Gateway, users are able to generate a graphical view of networks among the different agents within a specified population size.

## *5.16 Define Portlet Functionalities (2)*

This section constitutes the functionalities of a visualisation portlet from the point of view of potential users. The major functionality of the portlet involves performing data analysis on the Infection Model simulation output results. These visualisations are meant to help users in examining complex system data and events. In order to specify ways in which a Science Gateway can be used to support simulation output results, different ways of using a visualisation tool must be specified. Usually, a user is able to access the portlet interface and select a file or dataset in the appropriate file format (such as CSV), via the portlet user interface. Furthermore, a user is able to upload the file to generate a graphical view of networks among the different agents within a specified population size. Finally, a user will also be able to edit the chart after the graph has been generated. Based on the user requirements specification for this portlet therefore, the following deductions are made. A potential user will:

- Be able to easily access the visualisation portlet page on the Science Gateway.
- Be able to select the Infection Model simulation output result (in CSV format), via the Science Gateway, for subsequent analysis.
- Be able to upload the Infection Model simulation output file on the Science Gateway and generate a graphical view of the network between all the different agents.
- Be able to edit the resultant chart after the graph has been generated.

The above user requirements are incorporated at the graphical user interface of the visualisation portlet and ported on the Africa Grid Science Gateway. In order to implement a visualisation application as a service on the Science Gateway, two main approaches are considered. This includes the client side and the server side visualisation. For the server side visualisation, a computer program is used on the server to render the visual representation of the data. This visual representation can then be sent to the client as an image to be viewed with a web browser. In the client-side implementation however, all the implementation is done on the client side and the data is transferred to the client and visualised by specific software such as a web browser. For the Infection Model visualiser therefore, the client side implementation is used to render a graphical view of each simulation output result via a Science Gateway interface.

## *5.17 Visualiser Portlet Design (6)*

The visualiser portlet design is similar to the design processes described for the Infection Model portlet. However, this design will include functionalities to upload files (such as the Infection Model simulation output result), view and edit graph. Therefore, the different stages that will realise these functionalities (in addition to those defined for the Infection Model portlet) is as follows:

1) The user seeks to access the Science Gateway.

2) Science Gateway takes the user through an authentication and authorisation procedures.

3) A user submits his/her credentials to the Science Gateway for access.

4) Based on the user roles and privileges, Science Gateway presents the desired

portlet/application to the user (Visualiser portlet).

5) The user can view the portlet, upload files and generate graph.

6) The user can edit the resultant graph.

7) Procedure ends.

From the above sequence of steps, two main building blocks is derived to represent and implement the prototype architecture. These include user access and the user interface. The steps above are similar to the ones defined for the Infection Model portlet, except users are not required to submit, monitor and download jobs. However, users can upload their input data (such as the Infection Model simulation output results), view and edit graph as shown in Figure 5.8.

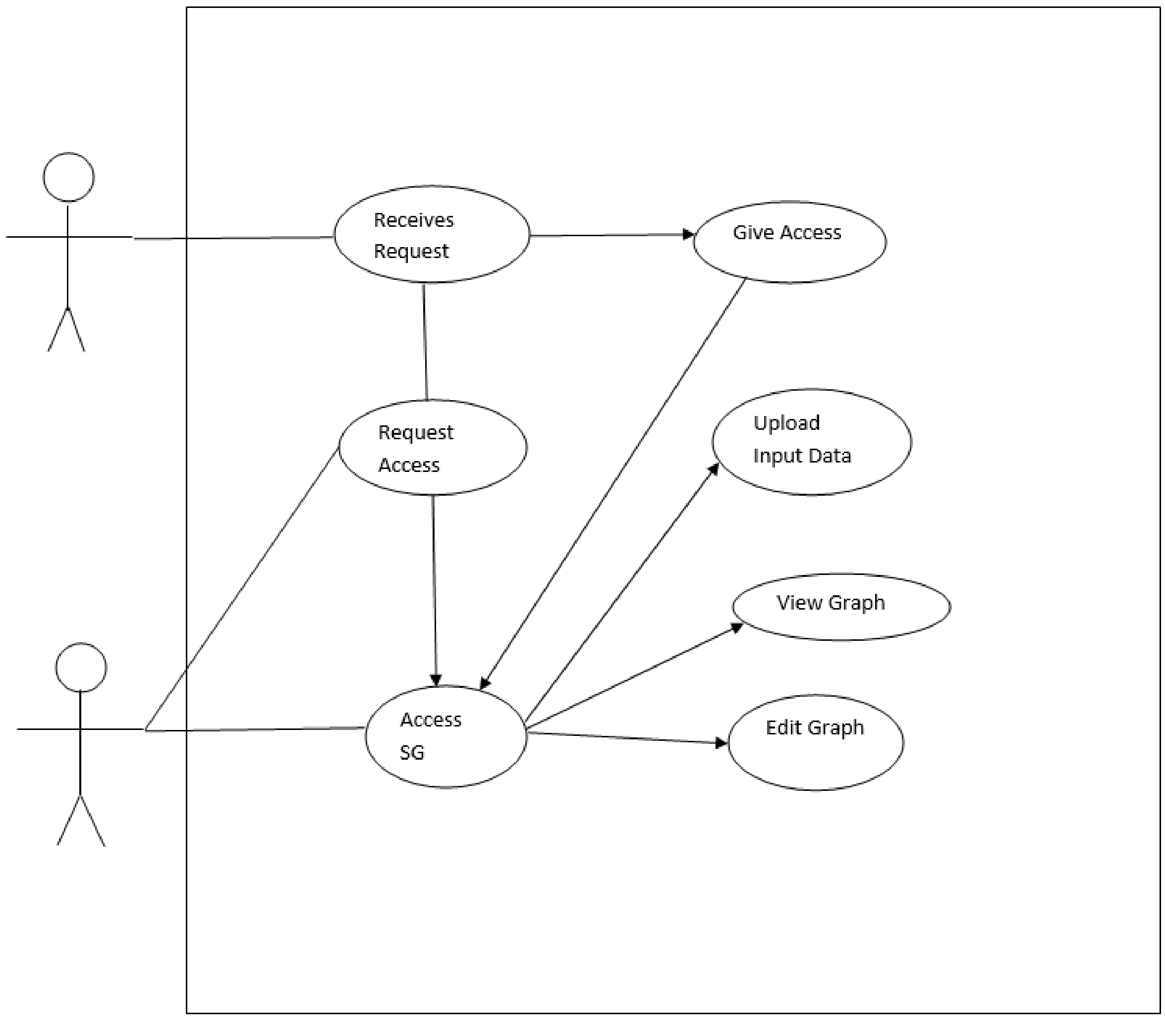


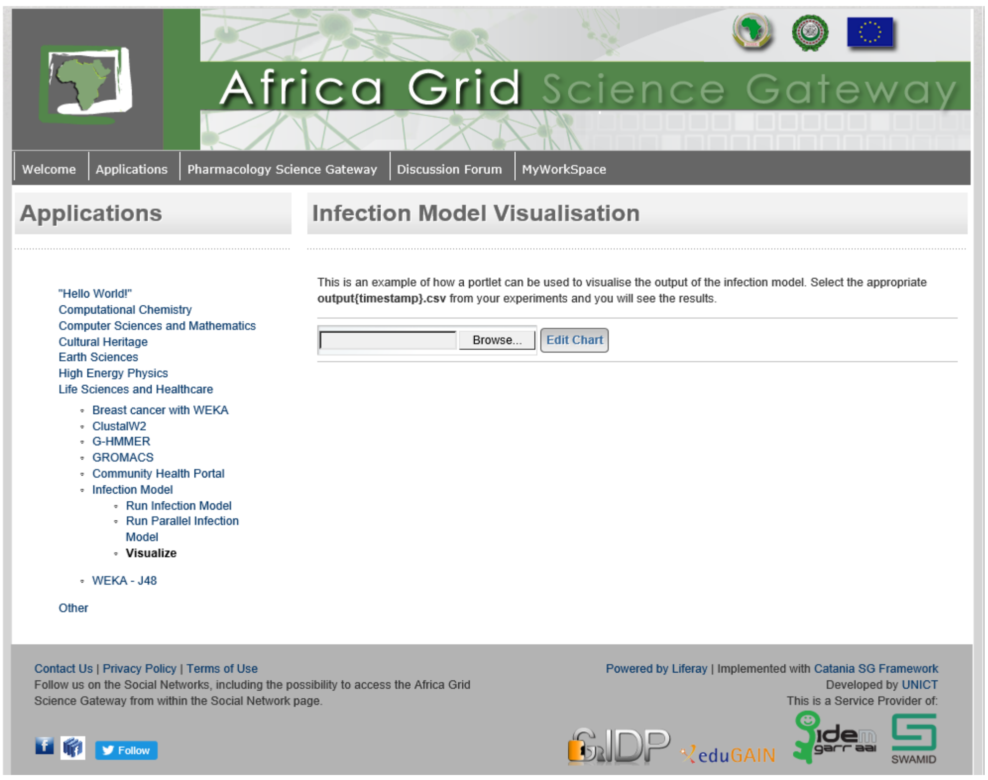
Figure 5. 8 Use Case Diagram for the Visualiser Portlet

The visualiser portlet is different from other portlets (utilised in this study) in that it represents a service which is implemented and rendered on the client side. It is developed to support the activities of the Infection Model portlet such as generating a graphical view of the simulation output results. Therefore, it does not require the execution of jobs on the DCI, neither does it need to be customised from specific Science Gateway framework. This portlet simply make use of the JQuery-CSV and google visualisation API for its implementation. The procedures for implementating the portlet will be described in the following section.

## *5.18 Develop/Implement the Portlet (7)*

To implement the visualiser portlet, the JQuery-CSV and google visualisation API is utilised as opposed to the APIs provided by the Science Gateway framework as seen in the case of the Infection Model portlet. The jQuery-CSV library enables the transformation of a sting of csv data into the appropriate format for Google's visualisation library. The details of the implementation of the visualizer portlet (using the JQuery-CSV and google visualisation API) is briefly discussed in this section as follows:

The implementation process begins with the utilisation of the various functions that the google visualisation API support. It uses the ChartWrapper class to wrap the chart and handle all loading, drawing and Datasource querying for the class. This class is used to expose methods for setting values on the chat as well as drawing it. It can also be used to save a chart for reuse. It also simplifies reading from a data source as it does not require the need for a query callback handler. The fixOptions function is used to set the options on the chart wrapper so that it draws correctly. The chart editor is used to automatically enable animations which does not look right with the ChartRangeFilter as well as set the hAxis.viewWindowMode to 'pretty', which may not work well with continuous axes. The ChartEditor class is used to open an in-page dialog box that enables a user to customise a visualisation on the fly. To utilise the ChartEditor, the charteditor package in google.charts.load() is loaded. In addition, a ChartWrapper object that defines the chart for the user to customise is created. Furthermore, a new ChartEditor instance is created and the ChartEditor.openDialog() is called having passed in the ChartWrapper. To update the chart in the code, the setChartWrapper() is called. Finally, the drawChart() function is used to transform the CSV file into a format that is suitable for google's visualisatioin library.


Africa Grid Science Gateway

Welcome | Applications | Pharmacology Science Gateway | Discussion Forum | MyWorkSpace

Applications

"Hello World!"
Computational Chemistry
Computer Sciences and Mathematics
Cultural Heritage
Earth Sciences
High Energy Physics
Life Sciences and Healthcare
- Breast cancer with WEKA
- ClustalW2
- G-HMMER
- GROMACS
- Community Health Portal
- Infection Model
  - Run Infection Model
  - Run Parallel Infection Model
  - **Visualize**
- WEKA - J48

Other

Infection Model Visualisation

This is an example of how a portlet can be used to visualise the output of the infection model. Select the appropriate **output{timestamp}.csv** from your experiments and you will see the results.

Browse... | Edit Chart

Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid Science Gateway from within the Social Network page.

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT
This is a Service Provider of:

Figure 5. 9 Visualiser Portlet Main Page

## *5.19 Test Portlet (9)*

The main page of the visualiser portlet on the Africa Grid Science Gateway can be viewed at https://sgw.africa-grid.org/visualize-infection-model-result (also See Figure 5.9). Analysis is done simply by making use of the Infection Model demonstration graph tool which allows users to see the graphical visualisation of the results. A simple demonstration of this service is a JQuery-CSV and google visualisation API where data file may be loaded in order to visualise large data sets. This can thus help produce a fully functional visualisation tool which may be ported on the Science Gateway and utilised by a particular application specific domain or community of practice. To visualise any output, the text file is uploaded to produce a graph of one element over the other (e.g. population variance over time) as shown in Figure 5.10. The ABMS visualiser is used to capture images and visualise the output of the ABMS Infection Model. This presents an interface where users can simply upload and visualise the output of the ABMS Infection Model such as the infection_model_simulation_output.txt file.

Consequently, when a job is ready and the simulation output result is collected, a user can upload the result using the Infection Model visualisation tool on the Science Gateway, and a graphical view of the job output is generated. Further analysis may also be performed on the simulation output results by applying machine learning techniques using the WEKA application tool which has been discussed earlier in Section 5.9.

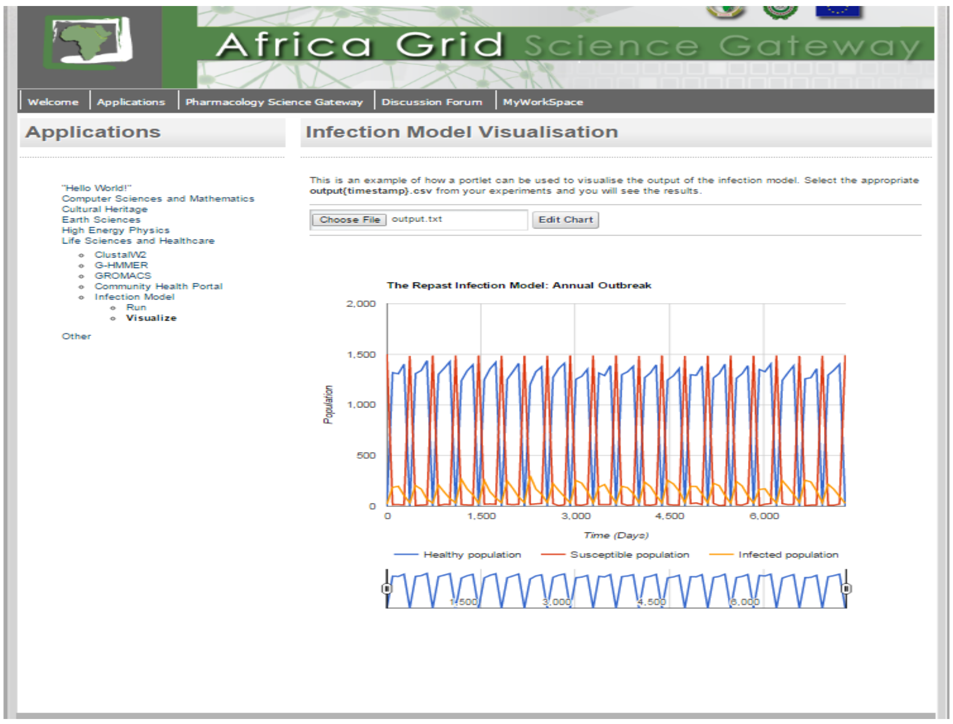

This is an example of how a portlet can be used to visualise the output of the infection model. Select the appropriate **output{timestamp}.csv** from your experiments and you will see the results.

Figure 5. 10 Visualiser Portlet showing a Graphical View of the Infection Model Simulation Output Result

## 5.20 Summary

In this Chapter, the details of the design and implementation of the adopted scientific software applications (ABMS, WEKA and Infection Model Visualiser) are analysed. In the description of the process involved in the design and implementation of the first case study, the portlet functionalities and the general requirements of the portlets (software, hardware and the Science Gateway requirements) are presented. This was closely followed by the detailed steps/processes involved in the actual design and implementation of the ABMS portlet. In addition, the deployment, demonstration and test of the actual Infection Model portlet is performed and the results are presented.

In the later stages of this Chapter, a second case study is ported on the Science Gateway. This case study is WEKA's J48 Classifier which was adopted to help with analysing the simulation output results obtained from the Infection Model portlet. Furthermore, it can also be used to evaluate the method of the Infecion Model portlet by comparing the approach taken to implement both portlets. It therefore plays different roles in serving as a viable case-study as well as help in performing analysis on the simulation output results (obtained from the Infection Model case study) or other associated data sets. In a similar manner to the Infection Model, the steps involved in the design and implementation of the second use-case (WEKA's J48 Classifier), as well as the processes involved in the use of the actual portlet, test and results are discussed in great detail. Finally, the processes involved in porting a visualiser portlet (for analysing the Infection Model portlet simulation output results) was also discussed which is similar to the first two case studies.

# Chapter 6: DESIGN AND IMPLEMENTATION OF MESSAGE FOR THE EXECUTION OF MULTIPLE EXPERIMENTS

## *6.1 Overview*

This Chapter discusses the design and implementation of the parallel approach for both portlets (i.e. the Infection Model portlet and the WEKA - J48 Portlet) discussed in Chapter 5. It begins with a simple description of the parallel version of the Infection model portlet, as well as the general requirements (in addition to the ones that were discussed in Chapter 5) and the design and implementation processes that was utilised in the realisation of this version of portlet.

Following from the previous portlets therefore, this Chapter builds on the work in Chapter 5 by developing applications in Science Gateways for the execution of jobs (in parallel) both at the user interface level (for the Infection Model portlet) and the DCI level (for the Infection Model Portlet and the WEKA – J48 portlet). Chapter 6 is therefore organised as follows. First, the general description of the parallel approach used for both portlets was discussed in Section 6.2. Section 6.3 highlights the portlet functionalities and Section 6.4 describes the general requirements, in addition to the ones defined for the sequential versions of the portlets in Chapter 5. Section 6.5 outlines the design process that was utilised in this version of the portlet. This includes the different job compositions (such as the Workflow N-1, Job collection and Job parametric) that made up the parallel portlet.

Furthermore, Section 6.6 describes the process involved in the implementation of this version of portlet (by using the different JSP pages, the different java classes and the adopted cloud and grid engine method) for the execution of parallel jobs. The resultant portlet and its test was presented in Section 6.7. In Section 6.8, the second layer of parallelism which was done at the DCI level for both Infection Model and WEKA – J48 portlet were highlighted.

## *6.2 The Infection Model Portlet (For Multiple Experiments)*

The parallel version of the Infection Model portlet also utilises the different input parameters similar to the ones defined in the sequential version of the portlet discussed in Chapter 5. That is, this version of the portlet also consists of input parameters such as simulation period, recovered count, infected count and susceptible count. However, in the implementation of the parallel version (which is explained in more detail in the following section) the Infection Model portlet allows users to run multiple jobs, simultaneously (i.e. users are able to specify multiple experiments to run within a distributed environment). Once a user has specified the desired number of jobs to be executed in parallel using the portlet user interface, a table consisting of the different field for each parameter is automatically generated which can then be populated by specifying the different input parameters.

The parallel implementation of the Infection Model portlet is divided into two levels. The first level parallelism is the user interface layer of the portlet (already described above), where users can specify and submit multiple jobs (simulation runs) at any given period. However, the parallelism at this level will only be implemented for the |Infection Model portlet. The second level parallelism however is implemented at the DCI layer which sees an efficient use of HPC resources. At this level, parallelism is performed for both the Infection Model portlet and the WEKA – J48 portlet. Both portlets are deployed to run on multiple instances of VMs with several core processors as opposed to simply just running on a single instance of machines. As such, VMs with multiple cores (at different cloud sites) are utilised for the execution of both versions of the portlet.

## *6.3 Portlet Functionalities*

The functionalities of the Infection Model portlet have been defined in Section 5.4 (in Chapter 5). However, based on the general portlet requirements and the components of this version of portlet, the following deductions are made for the user interface requirements. A potential user of the portlet will:

- Be able to easily access the Infection Model portlet page on the Science Gateway.
- Be able to specify the number of experiments to run (in parallel) and the different input parameters on the Science Gateway.
- Be able to execute and submit jobs of the Infection Model on DCIs.
- Be able to monitor jobs for statuses and updates.
- Be able to download outputs/results of simulation experiments.

## *6.4 General Requirements*

The portlet requirements such as the software, hardware and the Science Gateway requirements (for the parallel version) are similar to the ones defined for the sequential versions of both portlets in Section 5.5. For the parallel version however (as part of the user requirements) users will be able to specify input parameters on their own accord without being limited to a defined set of experiments. Furthermore (as discussed in the hardware requirements section in Chapter 5) VMs with multiple cores are utilised at the DCI layer to execute jobs in parallel. Also, as already mentioned in the above Section 6.3, the user interface is designed such that a user can specify and execute multiple experiments at any given time. Furthermore, the excess availability of cloud resources at the time of developing this version of the portlet ensure that users are not restricted in the number of experiments they may want to perform.

## *6.5 Infection Model (Parallel) Portlet Design*

The design process for this version of portlet is largely similar to the one that was discussed in the first version of the Infection Model portlet. It utilises the five major building blocks (discussed in Chapter 5) which were derived from the initial prototype architecture. These include the user access, the user interface, the pilot-script.sh, pre-configured VMs, and MyJobs portlet. These building blocks which form a major part of the design process of this version of the portlet have already been explained in great detail in Chapter 5. However, in addition to the aforementioned building blocks, this design process incorporates a major approach that enables the execution of multiple Infection Model experiments within a distributed environment. The embedded approach is seen as part of the process in which CSGF framework conducts the execution of parallel jobs in general. The different stages that will realise the above functionalities is described below as follows:

1) The user seeks to access the Science Gateway.
2) Science Gateway takes the user through an authentication and authorisation procedures.
3) A user submits his/her credentials to the Science Gateway for access.
4) Based on the user roles and privileges, Science Gateway presents the desired portlet/application to the user.
5) The user can view the Infection Model portlet.

6) The user can specify the number of jobs to run (in parallel) as well as the different input parameters and submit jobs to DCIs.
7) Science Gateway (using a pilot script) can collect input parameter being specified through the user interface and submits to e-Infrastructure.
8) Science Gateway gives job statuses and user can monitor and download the outputs of jobs.
9) Procedure ends.

For the purpose of implementing this version of the portlet, the CSGF framework defines and incorporates several ways of executing parallel jobs (otherwise known as special jobs) within a distributed environment. These different approach is discussed below as follow:

### 6.5.1 Job Collection

Job collections are set of independent jobs which run in parallel. This parallel application spawns N sub-jobs. When all the jobs are successfully completed the whole collection becomes DONE. The result of the execution of a job collection in a DCI is an archive containing all the output files for each job as seen in Figure 6.1.

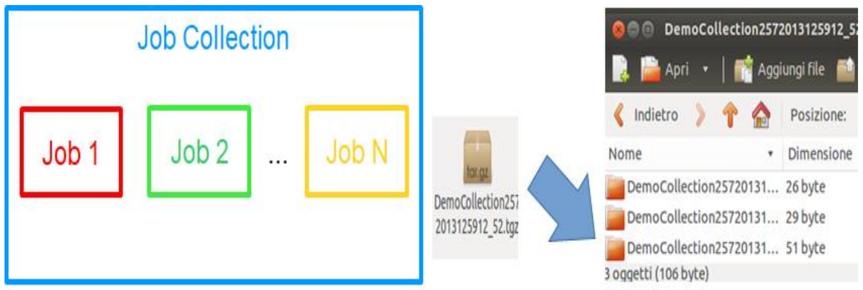
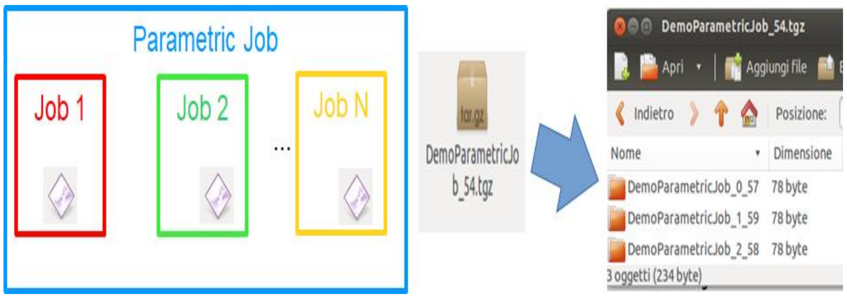

Job Collection
Job 1
Job 2
...
Job N
tar.gz
DemoCollection257 2013125912_52.tgz
DemoCollection2572013125912_5
Apri
Aggiungi file
Indietro
Posizione:
Nome
Dimensione
DemoCollection25720131... 26 byte
DemoCollection25720131... 29 byte
DemoCollection25720131... 51 byte
3 oggetti (106 byte)

Figure 6.1 Job Collection

### 6.5.2 Parametric Job

The parametric jobs are set of jobs that run in parallel which have the same executable (i.e. pilot-script.sh) but consists of different arguments (such as the input parameters). This type of parallel application spawns N sub-jobs and when all jobs are successfully completed, the whole parametric job becomes DONE. Similarly, the result of the parametric job is an archive consisting of the output file for each job as seen in Figure 6.2.

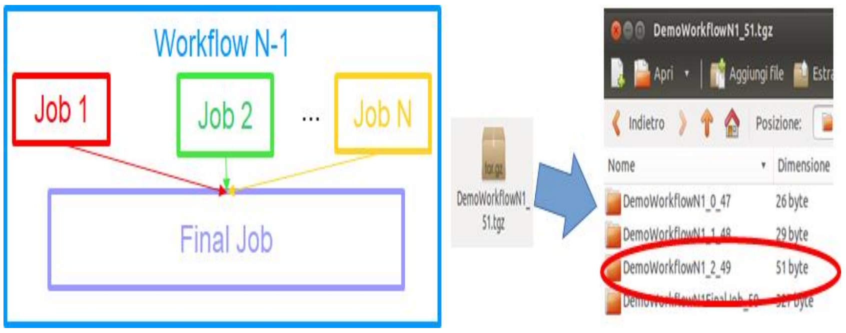


```
Nome                       Dimensione
DemoParametricJob_0_57     78 byte
DemoParametricJob_1_59     78 byte
DemoParametricJob_2_58     78 byte
3 oggetti (234 byte)
```

Figure 6.2 Parametric Job

### 6.5.3 Workflow N-1

Workflow N-1 is a special kind of job that consists of two different levels of jobs. The first level of jobs is a set of independent jobs that run in parallel and the final (collector) jobs aggregate all the first level job outputs and generate a file output. As such, workflow N-1 is a parallel application that spawns N sub-jobs and waits until they are correctly completed before submitting a new job. The input files of this new job are effectively the outputs of the N sub-jobs. When all the first level jobs are successfully completed then the final job is submitted. When the final job becomes DONE, the whole workflow is considered to be completed as shown in Figure 6.3.



```
Nome                          Dimensione
DemoWorkflowN1_0_47           26 byte
DemoWorkflowN1_1_48           29 byte
DemoWorkflowN1_2_49           51 byte
DemoWorkflowN1FinalJob_50     327 byte
```

Figure 6.3 Workflow N-1

Generally, to manage special (parallel) jobs with Catania Grid and Cloud Engine, the parametric job and the workflow N-1 are designed to be subclasses of the job collection class as illustrated in Figure 6.4. This is mainly due to the simple reason that most behaviours of both parametric job and workflow N-1 are common to the job collection.

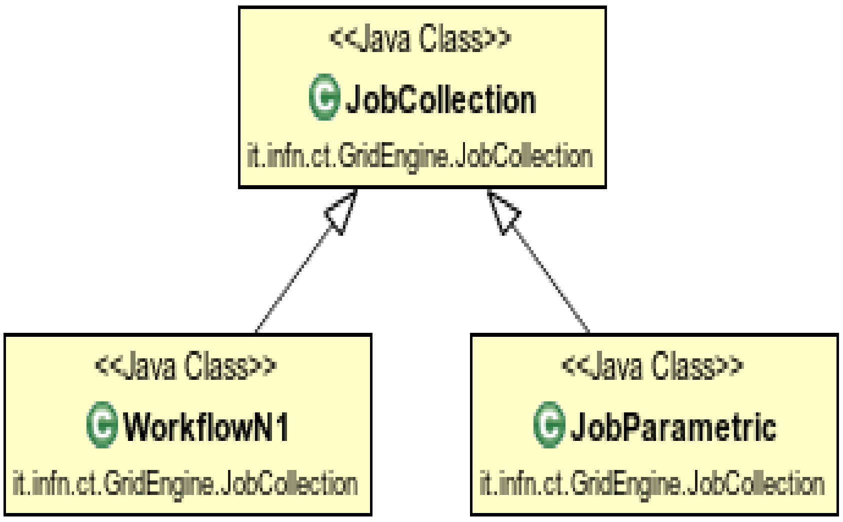

<<Java Class>>
JobCollection
it.infn.ct.GridEngine.JobCollection
<<Java Class>>
WorkflowN1
it.infn.ct.GridEngine.JobCollection
<<Java Class>>
JobParametric
it.infn.ct.GridEngine.JobCollection

Figure 6.4 Showing the JobCollection as a superclass of both Workflow N-1 and JobParametric

The JobCollection java class consists of different attributes and handles the behaviour of the job collection. It is responsible for performing actions such as job creation, status updating and closing the collection when all the jobs are DONE. Figure 6.5 shows some methods of this class which handles all the functionalities for developing scientific applications (in parallel) in Science Gateways.

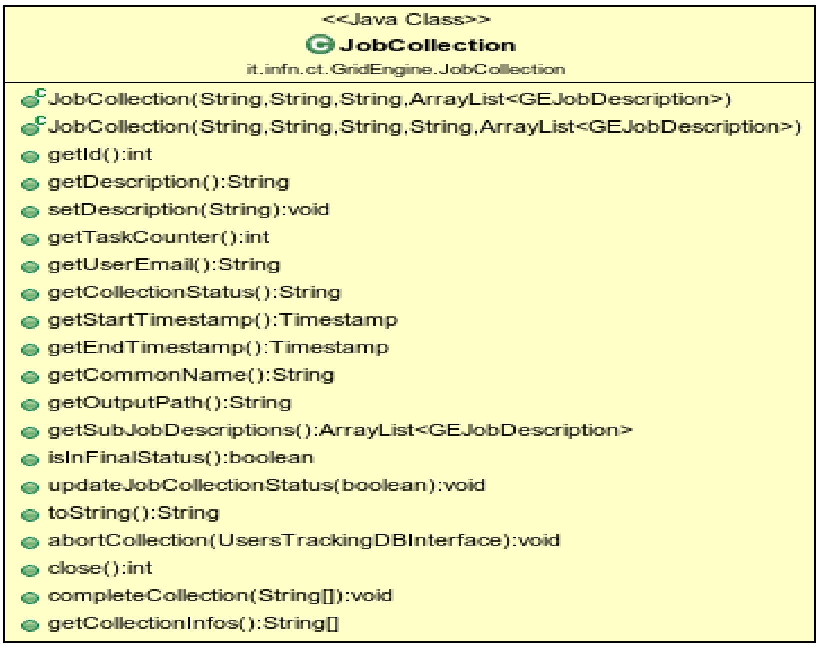

<<Java Class>>
JobCollection
it.infn.ct.GridEngine.JobCollection
JobCollection(String,String,String,ArrayList<GEJobDescription>)
JobCollection(String,String,String,String,ArrayList<GEJobDescription>)
getId():int
getDescription():String
setDescription(String):void
getTaskCounter():int
getUserEmail():String
getCollectionStatus():String
getStartTimestamp():Timestamp
getEndTimestamp():Timestamp
getCommonName():String
getOutputPath():String
getSubJobDescriptions():ArrayList<GEJobDescription>
isInFinalStatus():boolean
updateJobCollectionStatus(boolean):void
toString():String
abortCollection(UsersTrackingDBInterface):void
close():int
completeCollection(String[]):void
getCollectionInfos():String[]

Figure 6.5 JobCollection Java Class

The JobParametric java class handles the behaviour of the parametric job. It extends the job collection class and inherits all the behaviours and attributes that are defined in this class. In addition, it defines a new attribute (as shown in Figure 6.6) that specifies the single executable for the parametric jobs. It also defines a method that returns the executable as shown below:

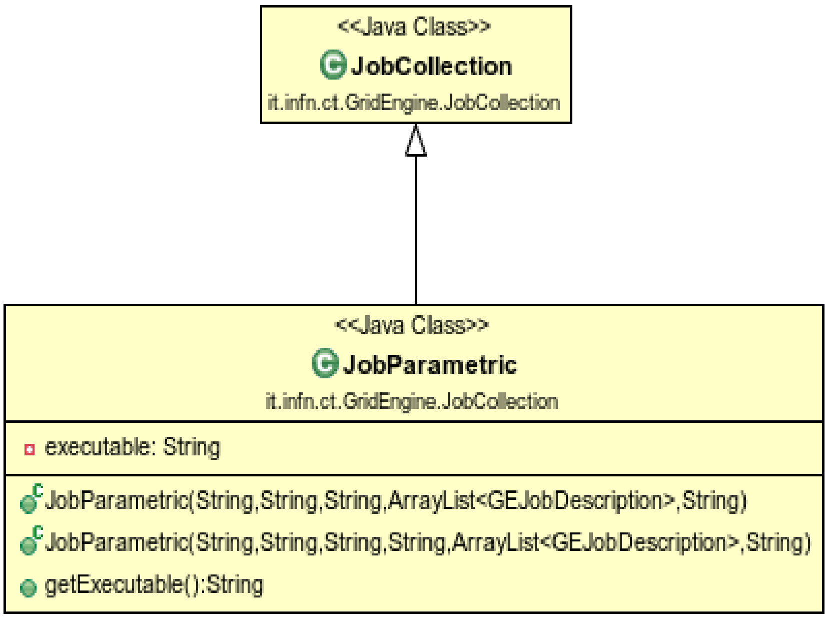
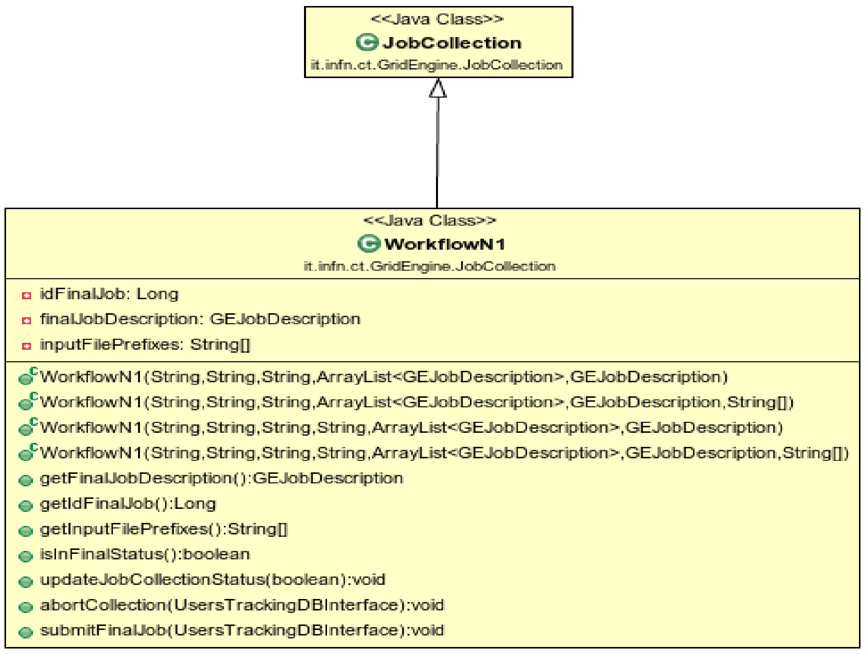

<<Java Class>>
JobCollection
it.infn.ct.GridEngine.JobCollection
<<Java Class>>
JobParametric
it.infn.ct.GridEngine.JobCollection
executable: String
JobParametric(String,String,String,ArrayList<GEJobDescription>,String)
JobParametric(String,String,String,String,ArrayList<GEJobDescription>,String)
getExecutable():String

Figure 6.6 JobParametric Java Class

The workflow N-1 java class handles the behaviour of the workflow N-1. It extends and inherits all the attributes and methods defined in the JobCollection class as shown in Figure 6.7. In addition, it defines new attributes such as the final job id, the GEJobDescription and the inputFilePrefixes as illustrated in the Figure 6.7 below. It also overrides some JobCollection methods such as the updateJobCollectionStatus (since the behaviour of the workflow N-1 class is different from the job Collection class).



<<Java Class>>
JobCollection
it.infn.ct.GridEngine.JobCollection
<<Java Class>>
WorkflowN1
it.infn.ct.GridEngine.JobCollection
idFinalJob: Long
finalJobDescription: GEJobDescription
inputFilePrefixes: String[]
WorkflowN1(String,String,String,ArrayList<GEJobDescription>,GEJobDescription)
WorkflowN1(String,String,String,ArrayList<GEJobDescription>,GEJobDescription,String[])
WorkflowN1(String,String,String,String,ArrayList<GEJobDescription>,GEJobDescription)
WorkflowN1(String,String,String,String,ArrayList<GEJobDescription>,GEJobDescription,String[])
getFinalJobDescription():GEJobDescription
getIdFinalJob():Long
getInputFilePrefixes():String[]
isInFinalStatus():boolean
updateJobCollectionStatus(boolean):void
abortCollection(UsersTrackingDBInterface):void
submitFinalJob(UsersTrackingDBInterface):void

Figure 6.7 Workflow N-1 Java Class

To develop a portlet that can submit and execute ABMS jobs (in parallel) within a distributed environment, the parametric job portlet is utilised. This choice is taken because the Infection Model has one single executable (pilot-script.sh) and is specified with different input parameters which therefore suits the attributes of a parametric job. As mentioned earlier the JobParametric class is a subclass of the JobCollection class and utilises some of the methods within the JobCollection class (in addition to the methods defined within its own class) for the implementation of the Infection Model portlet. The actual implementation of this portlet is carried out in the following Section 6.6.

## *6.6 Develop/Implement the Portlet*

Similar to the first version of the Infection Model portlet, the ABMS parallel portlet utilises the Liferay Portal Framework for the development of the user interface and the Catania Grid and Cloud Engine for the execution of jobs within a distributed environment. The Infection Model portlet was deployed in a portlet named “Parallel Infection Model”. In a similar manner to the sequential version of the portlet discussed in Chapter 5, it utilises several classes such as the AppInfrastructureInfo class (for storing the required information necessary to submit a job to a given infrastructure), the AppPreferences class (for storing all the values of portlet preferences) and most importantly the InfectionModelParallelPortlet class (which consist of the main portlet class that extends and overrides the GenericPortlet class methods). In addition to the aforementioned java classes, different JSP was used for presenting the desired views to the Science Gateway users. These include the view page, the edit page, the help page and the submit page, which have all been discussed in Chapter 5. However, for the implementation of this version of the portlet, only the view page and the InfectionModelParallelPortlet class are utilised and explained in great detail. The other the java classes (AppPreferences and AppInfrastructureInfo) follows the same procedures that were defined in the sequential version of the portlet. Finally, the relationship between the main class (InfectionModelParallelPortlet) and how it interacts with the view page is similar to the sequential version of the portlet described in Chapter 5.

### 6.6.1 The View Page

The view page of the parallel version consists of the standard user interface of the ABMS portlet. Similar to the sequential version this portlet also incorporates most of the functionalities that were specified at the user requirements stage which are described in Chapter 5. It begins with a brief description of the parallel version of the portlet and the different ways in which the portlet is utilised. However, rather than having a pre-defined set of jobs where users can select and run experiments, this version has been designed such that users can simply insert the number of parallel jobs they wish to run. Subsequently, a table consisting of the different input parameters is generated which may be used to populate the various fields according to their specification. (Here, the input parameters are not pre-defined with the assumption that there is an unlimited resources at user’s disposal for the execution of jobs in a distributed environment).

### 6.6.2 The InfectionModelParallelPortlet Class

The same methods discussed in the sequential version of the portlet such as processAction() and the render() methods and the corresponding view methods (doView(), doEdit() and doHelp()) which are discussed in Chapter 5 are utilised in the InfectionModelParallelPortlet class. However, the method which the GridEngine uses to submit jobs in the sequential version is different from the one used in the parallel version of the portlet. All the methods discussed in the parametric jobs in Section 6.5 above are used to submit and execute the ABMS jobs. This entire process will therefore be illustrated as follows:

The infrastructures needed by the ABMS portlet to submit and execute jobs are assigned and enabled as follows:

InfrastructureInfo[] infrastructuresInfo = appPreferences.getEnabledInfrastructures();

In addition, the job description is defined within the submitJobCollection method and a new array list of the job description object (otherwise known as descriptions) is created. This array consists of the description of each job which belongs to the jobCollection. Furthermore, the set() method is used to specify the standard output and the standard error. Finally, the description object is added to the list as illustrated in Appendix C11.

To submit the parametric job, the JobParametric object (known as a collection) is created. Also, the constructor for the jobParametric class which is similar to the jobCollection is defined. Furthermore, the job descriptions and the string representing the single executables are passed as arguments. This is illustrated below:

```
JobCollection collection = new JobParametric (appInput.getUsername(),
appInput.getJobLabel(), "/tmp", descriptions, pilotScript);
```

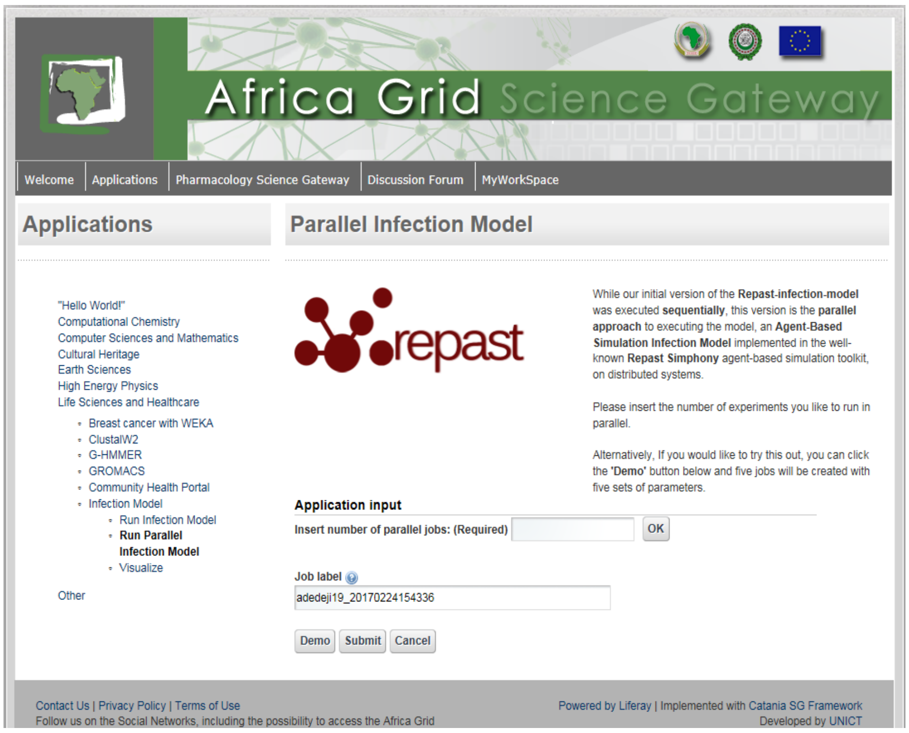
In addition, the GridEngine JobCollectionSubmission job submission object (otherwise known as tmpJobCollectionSubmission) is created. The constructor of the JobCollectionSubmission is defined and the parametric job object (collection) is passed as an argument. This is illustrated in Appendix C12. The above illustration shows that the development environment requires the database parameters (DBNM, DBUS, DBPW) which are defined and passed as arguments to the constructor. The production environment however, which comes after the else part, only requires the parametric job object (collection) as its argument. Notice that it does not take the database parameters as its arguments. Consequently, the Infection Model job can be submitted to the different DCIs by calling the submitJobCollection() method on the JobCollectionSubmission object (tmpJobCollectionSubmission) as follows:

```
tmpJobCollectionSubmission.submitJobCollection (enabledInfrastructures,
portalIPAddress, applicationId);
```

## *6.7* Test the Portlet

The parallel version of the Infection Model is deployed in a portlet known as "Parallel Infection Model". This was ported to the Africa Grid Science Gateway along with the sequential version of the Infection Model and WJ48 portlets (See Figure 6.8). Similar to the sequential version, this portlet is used by potential users to conduct their ABMS experiments in a distributed environment. By making use of this portlet, users can specify input parameters, execute jobs and obtain results via the MyJobs portlet. However, rather than facilitating users to specify and submit one single job or simulation run at a given period of time (as seen in the first version of the system) this portlet will allow users to specify multiple jobs at any given period.

The process of authentication and authorisation to the Science Gateways follow a similar approach to the sequential version of the portlet which was earlier described in Chapter 5. Parallel job executions can simply be achieved by specifying the number of experiments to be performed in parallel, via the portlet user interface. The 'OK' button automatically generates a field in tabular form based on the number of experiments that was specified by the user as demonstrated in the Figure 6.9.



Parallel Infection Model

While our initial version of the **Repast-infection-model** was executed **sequentially**, this version is the **parallel approach** to executing the model, an **Agent-Based Simulation Infection Model** implemented in the well-known **Repast Simphony** agent-based simulation toolkit, on distributed systems.

Please insert the number of experiments you like to run in parallel.

Alternatively, If you would like to try this out, you can click the '**Demo**' button below and five jobs will be created with five sets of parameters.

**Application input**

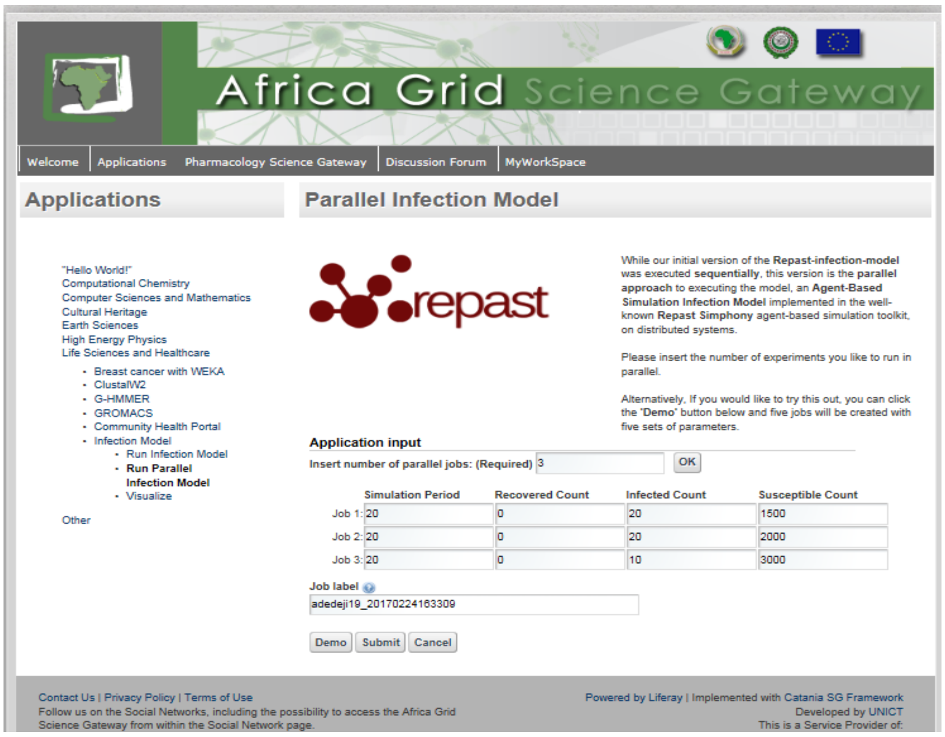
**Insert number of parallel jobs: (Required)**

Job label: adedeji19_20170224154336

Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT

Figure 6.8 Parallel Infection Model Application Portlet Page



Parallel Infection Model

While our initial version of the **Repast-infection-model** was executed **sequentially**, this version is the **parallel approach** to executing the model, an **Agent-Based Simulation Infection Model** implemented in the well-known **Repast Simphony** agent-based simulation toolkit, on distributed systems.

Please insert the number of experiments you like to run in parallel.

Alternatively, If you would like to try this out, you can click the '**Demo**' button below and five jobs will be created with five sets of parameters.

**Application input**

**Insert number of parallel jobs: (Required)** 3

| | Simulation Period | Recovered Count | Infected Count | Susceptible Count |
|---|---|---|---|---|
| Job 1: | 20 | 0 | 20 | 1500 |
| Job 2: | 20 | 0 | 20 | 2000 |
| Job 3: | 20 | 0 | 10 | 3000 |

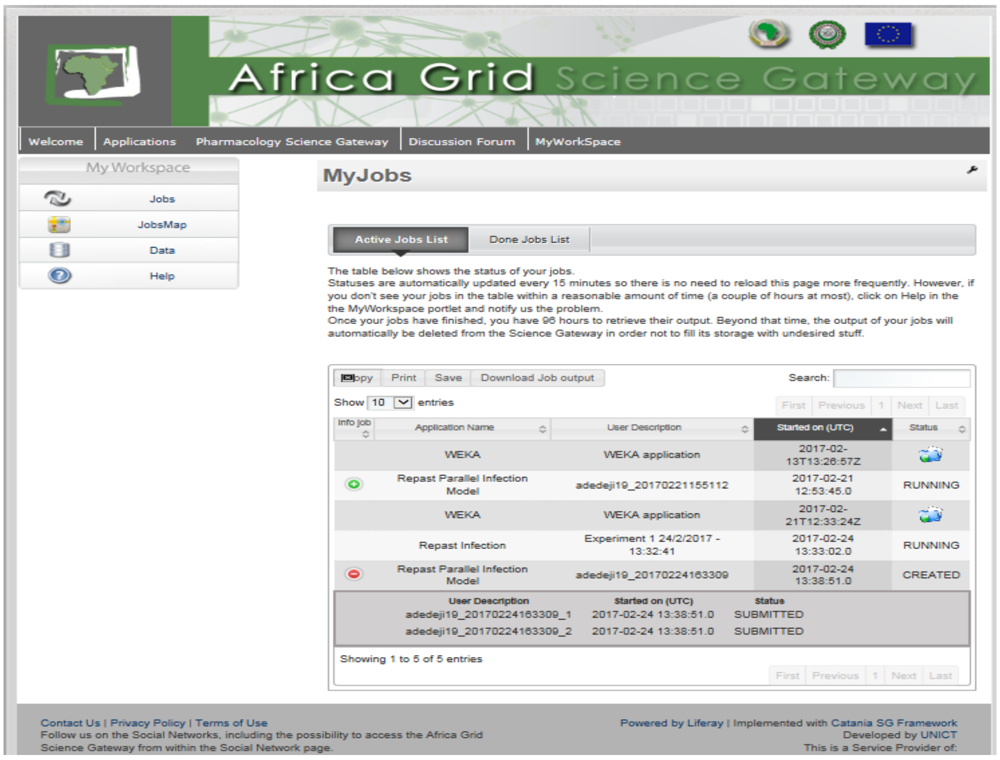
Job label: adedeji19_20170224163309

Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid Science Gateway from within the Social Network page.

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT
This is a Service Provider of:

Figure 6.9 Parallel Infection Model Page showing three (3) Simulation Runs

The above Figure 6.9 shows that three (3) experiments were specified using the "insert number of parallel jobs field". This automatically generates three (3) fields where the different input parameters (Simulation Period, Recovered Count, Infected Count, and Susceptible Count) are specified. It was earlier stated that (for this version of the portlet) a parametric approach is adopted such that the parallel job will consist of the same executable (pilot-script.sh), but with different arguments (input parameters). Therefore, the three (3) experiments will spawn three (3) different jobs as seen in Figure 6.9. After each job submission, users can retrieve jobs from MyJobs Portlet. This portlet will display the status of all three (3) jobs either as CREATED, RUNNING, and DONE (See Figure 6.10).



The table below shows the status of your jobs.
Statuses are automatically updated every 15 minutes so there is no need to reload this page more frequently. However, if you don't see your jobs in the table within a reasonable amount of time (a couple of hours at most), click on Help in the the MyWorkspace portlet and notify us the problem.
Once your jobs have finished, you have 96 hours to retrieve their output. Beyond that time, the output of your jobs will automatically be deleted from the Science Gateway in order not to fill its storage with undesired stuff.

| Info job | Application Name | User Description | Started on (UTC) | Status |
|---|---|---|---|---|
| | WEKA | WEKA application | 2017-02-13T13:26:57Z | |
| ⊕ | Repast Parallel Infection Model | adedeji19_20170221155112 | 2017-02-21 12:53:45.0 | RUNNING |
| | WEKA | WEKA application | 2017-02-21T12:33:24Z | |
| | Repast Infection | Experiment 1 24/2/2017 - 13:32:41 | 2017-02-24 13:33:02.0 | RUNNING |
| ⊖ | Repast Parallel Infection Model | adedeji19_20170224163309 | 2017-02-24 13:38:51.0 | CREATED |

| User Description | Started on (UTC) | Status |
|---|---|---|
| adedeji19_20170224163309_1 | 2017-02-24 13:38:51.0 | SUBMITTED |
| adedeji19_20170224163309_2 | 2017-02-24 13:38:51.0 | SUBMITTED |

Showing 1 to 5 of 5 entries

Contact Us | Privacy Policy | Terms of Use
Follow us on the Social Networks, including the possibility to access the Africa Grid Science Gateway from within the Social Network page.

Powered by Liferay | Implemented with Catania SG Framework
Developed by UNICT
This is a Service Provider of:

Figure 6.10 MyJobs portlet showing different job statuses

A job which has been executed in parallel is said to be successful when all the three (3) individual jobs have successfully completed and their statuses have changed to a small folder icon as shown in Figure 6.10. When the job has successfully completed, it can then be retrieved via the MyJobs portlet for subsequent analysis.

## *6.8 Parallel Implementation of both the Infection Model Portlet and the Weka Portlet in the Distributed Environment Layer*

The above sections describe in detail the parallel implementation of the Infection Model portlet at the user interface level. The parallelism at this level (user interface level) enables potential users to specify and submit multiple experiments at any given time towards a distributed environment as opposed to the portlet that was discussed in Chapter 5 where users can only specify and submit one experiment. However (for the WEKA – J48 portlet) such level of parallelism is not applicable or required since potential users always have to upload files (to be used for WEKA analysis) on the Science Gateway. Therefore, the only parallel approach for the WEKA – J48 portlet is implemented at the DCI level where multiple core processors are utilised for the execution of WEKA jobs. This can be contrasted to using single cores to implement the sequential version of the portlets described in Chapter 5 where VMs with 1 core each at different cloud sites were utilised.

For the implementation of the parallel version of the portlets however, both the Infection Model and WEKA applications are deployed to run on VMs with multiple instances. As such, VMs with multiple cores at different cloud sites are utilised for the execution of this version of portlets. This implementation therefore takes two forms:

1) Currently, the Infection Model portlet has five simulation runs on the Science Gateway, but an additional five simulation runs are replicated to make a total of ten. The parallelisation can therefore be performed as follows:
   Experiment 1: 10 replications / 10 per instance / 1 instance used.

   Experiment 2: 10 replications / 5 per instance / 2 instances used.

   Experiment 3: 10 replications / 2 per instance / 5 instances used.

   Experiment 4: 10 replications / 1 per instance / 10 instances used.

2) For the WEKA – J48 portlet, cores with multiple instances were utilised for the execution of specific data sets (such as weather data or the simulation output result) to be used for WEKA analysis which may be uploaded via the WEKA – J48 portlet. See discussions of the analysis of the simulation output result using WEKA – J48 in Appendix G.

These experiments are done in a similar fashion to the performance test that was conducted using the gUSE/WS-PGRADE platform (which represents another Science Gateway Framework that was described in Chapter 2) where the same set of parallelism was performed for the Infection Model on the Science Gateway. The performance test on both infrastructures (CSGF and the gUSE/WS-PGRADE Science Gateway Framework) can therefore provide an avenue to compare these two Science Gateway frameworks.

This study (in large parts) has concentrated on the accessibility and availability of scientific software applications to communities of practice (via the use of DCI resources) by creating a MESSAGE methodology for developing such scientific applications in Science Gateways. Usually, executing scientific applications in a DCI environment is not always easy and may often involve the use of some complex or sophisticated programming languages for the execution of jobs. The entire process which usually requires a steep learning curve may be a hindrance to adopting these technologies. Consequently, this thesis has focused on ensuring that scientists and researchers across the world can access and use scientific applications running on the distributed computing resources of e-Infrastructures. By using the DCI resources in this way to enable easy access to scientific applications, how can other (related) research outputs such as the simulation output results be easy to access and readily available to communities that need them? An approach used to enable this process is discussed in a section in Chapter 7.

## *6.9 Summary*

In this Chapter, the approach that was taken to design and implement the parallel version of both the Infection Model portlet as well as the WEKA – J48 portlet were discussed. This Chapter begins by presenting a description of the parallel portlet as well as discuss the general requirements of the portlet (in addition to the requirements described in the previous Chapter). It also presents the functionalities of the portlet by building on the functionalities described for the first version of the portlet.

In addition, it presents the different layers that make up the parallel approach for the execution of jobs in a distributed environment. These involve two levels of parallelism such as the user interface layer (which was implemented for the Infection Model portlet and where users can specify multiple simulation runs at any given time) and the DCI layer (that was implemented for both the Infection Model Portlet as well as the WEKA – J48 portlet). The DCI layer made use of VMs with multiple cores (at different cloud sites) to execute jobs which are triggered by each portlets.

Furthermore, special kind of parallel jobs which comprise of Workflow N-1, Job collection and Job parametric are thoroughly discussed in this Chapter. Ultimately, the implementation of the portlet (using the different JSP pages, the classes involved and the adopted cloud and grid engine method) are later presented. Furthermore, it also discusses the use of the portlet to execute and submit multiple simulation jobs at any given time.

The next Chapter will now attempt to evaluate the Infection Model portlet and the WEKA – J48 portlet as well as the MESSAGE methodology that was used in implementation of both portlets. The first evaluation is done by comparing the development processes of both portlets, while the second approach will utilise real life scientific use-cases from two EU projects (such as the CloudSME project and the Sci-GaIA projects) for the evaluation process.

# Chapter 7: PORTLET EVALUATION

## *7.1 Overview*

This Chapter discusses the evaluation of the portlets and the methods that were utilised throughout this research to test different aspects of the MESSAGE methodology proposed in Chapter 4. In conducting this evaluation, varieties of approach were adopted. The first approach compares the methodology used to implement both Infection Model portlet and the WEKA – J48 portlet. The second approach (on the other hand) compares the methododology used in implementing individual portlets with the approach taken to develop different scientific applications in two EU funded projects. Both projects have been tasked with developing Science Gateways for the various communities of practice that may need them. This series of evaluation in this chapter will consequently lead to a revised version of the MESSAGE methodology discussed in Section 7.6.

Therefore, on the back of the previous Chapters, this Chapter is organised as follows. First, the overall approach that was adopted to evaluate the methods used in developing both portlets is analysed in Section 7.2. Section 7.3 describes the first approach used for the evaluation process, which compares the method for developing both the first case study (the Infection Model portlet) and the second case study (WEKA – J48 portlet). Furthermore, it compares the methododology used to develop both case studies with the developement of the use-cases of the first EU project (Sci-GaiA project). In the Sci-GaiA project, the use-cases were developed using similar Science Gateway framework (CSGF/FutureGateway API) that was adopted in the development of both the Infection Model portlet and the WEKA – J48 portlet. This was discussed in Section 7.4. Similarly, Section 7.5 compares the methodology for implementing both use-cases with the approach which was utilised in the development of all the applications of the second EU-project (CloudSME project). The applications in this project utilised a different Science Gateway Framework otherwise known as the gUSE/WS-PGRADE Science Gateway Framework. Finally, the MESSAGE methodology which was proposed in Chapter 4 was re-visited and (based on the evaluation) a revised MESSAGE methodology for developing scientific software applications in Science Gateways was proposed in Section 7.6. This Chapter concludes (in Section 7.7) with the use of the Infection Model simulation output result to demonstrate the easy access and retrieval of other associated research outputs or results. This demonstration was done by using enabling technologies such as Open Access Document Repository (OADR), Digital Object Identifiers (DOI) and the Open Researcher and Contributor ID (ORCID), in the context of e-Infrastructures.

## *7.2 Evaluation Process*

To determine the relevance as well generalise the approach that was taken in this research, an evaluation of the entire research process is analysed. The Cambridge English dictionary defines evaluation as: "to judge or calculate the quality, importance, amount or value of something". In terms of evaluating IT system however, (Farbey, Land et al. 1999) defines IT evaluation as "a process or group of parallel processes which take place at different points in time or continuously for searching and for making explicit, quantitatively or qualitatively, all the impacts of an IT project and the programme and strategy of which it is a part". In light of the above definition, a thorough evaluation of the method that was utilised in this research will therefore help to establish the validity and potential of the method and approach and thus provide an avenue for generalisation. The criteria which the researcher has followed to evaluate the overall method is established in this section as follows.

Reflective Assessment was utilised in the evaluation of the ideas and principles described in the MESSAGE methodology. This process was done by reflecting on the lessons learnt in applying the approach discussed in the MESSAGE methodology to the different case studies utilised throughout this research. Consequently, some important questions (such as: what the implementations reveal) came to light. Furthermore, by comparing the experiences in creating both the Infection Model and WEKA - J48 portlets helps to determine the different approaches that were taken, or whether the same approach has simply been utilised for the implementation of both portlets. In addition, other questions that came to light from these assessments also include: Is there enough evidence with the three different implementations and are there enough information to substantiate the claims that were set out in the MESSAGE methodology? Furthermore, the methodologies that were used in other Science Gateway projects were also evaluated either by referring to project manuals consisting of the implementation details of the use-cases, or through direct interactions with developers that were involved in those projects. Subsequently, the information obtained from this interactions were then compared with the approach described in the MESSAGE methodology.

To measure the quality, effectiveness and most importantly the generalisability of the approach that was utilised in this thesis, a comparative method was adopted. To test the ideas and methods that were developed in this thesis, two different case studies were developed and implemented. Firstly, the methodology that was utilised in the development of both case studies (Infection Model and WEKA) were compared and contrasted. By reflecting on the lessons learnt in the development/implementation of both portlets (having utilised the proposed MESSAGE methodology) raises different questions. Secondly, some real-life case studies implemented using the CSGF/FutureGateway Science Gateway Framework are classified and analysed. Furthermore, the method and approach that was utilised in the realisation of these case studies are discussed. Consequently, the methodology that was employed in the realisation of both case studies is compared with the method that was utilised throughout the first EU project. Lastly, in a similar manner to the real-life case studies developed using the CSGF/FutureGateway Science Gateway framework, some use-cases from the Cloud based Simulation platform for Manufacturing and Engineering (CloudSME) project which were implemented using the gUSE/WS-PGRADE Science Gateway framework approach were also analysed. The approach that was adopted in the development of the different use-cases of this project (CloudSME) is also compared with both case studies that were utilised in this study. Since the scientific applications of the CloudSME project were implemented using a different Science Gateway approach (known as the gUSE/WS-PGRADE), this comparison is therefore considered to be key in the generalisation of the proposed MESSAGE methodology. Finally, once a systematic comparison of the different sets of case studies is performed using the three different approach described above (and the evaluation of the Science Gateway methodology is complete), the MESSAGE methodology proposed in Chapter 4 is revised.

As mentioned earlier, two different sets of real life case studies were employed in the evaluation of the MESSAGE methodology. The first set consists of the case studies in the Africa Grid Science Gateway which were implemented using the CSGF/FutureGateway Science Gateway framework approach, while the second set includes case studies from the CloudSME project that were implemented using the gUSE/WS-PGRADE Science Gateway framework. The Africa Grid Science Gateway is a standard web 2.0 demonstrative platform that was developed to show the different applications which were earlier identified by the past ei4Africa and the current Sci-GaIA projects. Sci-GaIA is a H2020 project which was funded by the European Commission (DG CONNECT) in order to promote the development and use of e-Infrastructures in Africa. CloudSME (on the other hand) is yet another Science Gateway project which establishes a cloud-based simulation platform that enables the use of state-of-the-art, simulation technology as Software as a Service (SaaS) in the cloud. As such, manufacturing and engineering SMEs can get access to a pay-per-use, one-stop-shop for simulation applications that are built on top of several Cloud and HPC resources and technology layers. It is important to note here that the case studies in the Africa Grid Science Gateway (which were implemented using the CSGF/FutureGateway Science Gateway framework) are directly related to this research since a similar Science Gateway framework was adopted for the implementation of the different case studies. This is explained in more detail in the following sections.

## *7.3 The Design and Implementation of the Infection Model Portlet and WEKA - J48 Portlet (Re-visited)*

The Infection Model portlet is designed to enable users to specify different input parameters to execute jobs in a distributed environment. As such, each specification represents a single replication (or run) for each experiment to be executed. Subsequently, experiments are performed and results are obtained using VMs with a single core processor in a Cloud environment. The parallel version of the Infection Model portlet is designed to perform multiple experiments (multiple runs/replications) via the Science Gateway interface. Parallelisation is therefore implemented both at the user interface level as well as the DCI layer. Consequently, replications/runs are executed over VMs with multiple core processors within a Cloud environment.

The WEKA – J48 portlet on the other hand was designed to enable users to analyse specific input data, or the simulation output result obtained from the Infection Model portlet, via the Science Gateway. Similar to the sequential version of the Infection Model portlet, the WEKA – J48 portlet is used to perform experiments using VMs with single core processor within a cloud environment. In addition, the parallel version of the WEKA – J48 portlet (via the DCI layer) can enable the execution of WEKA jobs over multiple core processors within a cloud environment. While the design of both portlets follow a similar approach such that same logical steps were followed in their design (i.e. the sequence of steps that is used to realise the functionalities of both portlets which was defined in Chapter 5), their implementation however followed slightly different approaches. The similarities as well as the differences are consequently discussed in the following sections.

### 7.3.1 The design of the Infection Model Portlet and WEKA Portlet

To test the approach described in the MESSAGE methodology and to subsequently execute different aspects of the Science Gateway framework of services (using the proposed MESSAGE methodology) a case study (an Infection Model) was utilised in the early sections of Chapter 5. Furthermore, a second case study was later introduced in the latter part of Chapter 5 which is used to further test and investigate the different ideas that were described in the proposed MESSAGE methodology. Based on the nature of both case studies, their design and implementation followed precise patterns as discussed in Chapter 5.

In the design of both portlets (Infection Model and WEKA – J48), the general approach which was taken was such that potential users of both portlets will be able to perform specific operations based on the user requirements specifications defined in Chapter 5. Both portlets follow a general sequence of steps for the realisation of the functionalities which are defined based on the user requirement sections. Furthermore, this sequence of steps give rise to six major building blocks which are initially derived to represent or implement the prototype architecture of the first portlet (Infection Model). This includes user access, user interface, pilot-script.sh, pre-configured virtual machine, myjobs portlet and the ABMS Visualiser (in the case of the Infection Model portlet). However, these six building blocks were later found to be paramount and applicable to the design of the second case study (WEKA). The different stages which was used for the design of both portlets could therefore be described as follows:

1) The user seeks to access the Science Gateway.
2) Science Gateway takes the user through an authentication and authorisation procedures.
3) A user submits his/her credentials to the Science Gateway for access.
4) Based on the user roles and privileges, Science Gateway presents the desired portlet/application to the user.
5) A user is either able to view portlet, specify different input parameters and submit jobs to DCIs (in the case of the Infection Model portlet), or view portlet, upload files, and select appropriate filters, test options and algorithm and then submit their jobs to e-Infrastructure (in

the case of the WEKA – J48 portlet).
6) Science Gateway, using a pilot script, can collect input parameter being specified through the user interface and submits to e-Infrastructure.
7) Science Gateway gives job statuses, and a user can monitor and download the outputs of jobs.
8) User is able to generate graph view of simulation output results (in the case of the Infection Model portlet).
9) Procedure ends.

As seen from the above sequence of steps, starting from the need to access the Science Gateway to the point where an execution has been performed (and the overall job procedure ends) the different stages used for the realisation of both scientific software applications remain the same. The only difference is the type of job being performed in step five, i.e the specification of the different input parameters and the submission of jobs to DCIs (in the case of the Infection Model portlet) and the file uploads, selection of appropriate filters, test options and algorithm, and submission of jobs to DCIs (in the case of the WEKA – J48 portlet). Also, users may want to visualise the output of simulation results which was facilitated for the Infection Model portlet only, using another portlet (otherwise known as the visualiser portet). Apart from the two points mentioned above, the logical sequence of steps for both portlets remain the same. The revelation from the above sequence of steps is such that the approach taken to design the Infection Model as well as the WEKA – J48 portlet is generic to both sets of scientific applications.

### 7.3.2 The Implementation of the Infection Model Portlet and the WEKA Portlet

The Infection Model portlet and WEKA portlet adopted a similar Science Gateway framework (described in Chapter 2) for developing both scientific software applications in Science Gateways. However, while the Infection Model application adopted the CSGF methods for both the sequential and parallel versions of the portlet, the WEKA – J48 portlet (on the other hand) utilised the FutureGateway API technologies to execute jobs in a distributed environment. These jobs may be executed using the method of the GridEngine Multi-Infrastructure Job Submission (in the case of the CSGF adopted for the Infection Model portlet) and the various methods defined for the FutureGateway APIs which were utilised for the implementation of the WEKA – J48 portlet (as discussed in Chapter 5).

For the CSGF, the Grid and Cloud Engine (a generic software module which uses standard technologies to interconnect the Scientific Gateway presentation layer with the underlying distributed infrastructures) is tightly coupled with the JSAGA (a java library) to physically enable access to various DCIs. This method of communication is only possible with Java libraries. For the FutureGateway however, portlets are not directly linked to the Grid Engine but instead utilises the Restful API via another module known as the API server. In this evolution of the CSGF, the API server is tasked with connecting to the GridEngine to execute user request on the distributed Infrastructure. The Grid and Cloud Engine are modified to support many JSAGA adaptors and the ones implemented for the FutureGateway includes SSH (for remote hosts and clusters), rOCCI (for generic clouds and EGI fedCloud), WMS (for EMI-gLite Grid Infrastructure) and Tosca (for SaaS/PaaS/IaaS). These different adaptors ensure that the FutureGateway provides executor interfaces accordingly. As a result, the API server can interact with any other SAGA implementation or a different module altogether in order to target specific DCIs.

By using these different methods (as seen in the CSGF and the FutureGateway APIs) for accessing web-based computational services, the following points have therefore come to light:

1. A steady move from the use of rudimentary methods such as command line interfaces for interacting with DCI resources.

2. Web-based content management technologies such as Liferay portal and their associated portlets when tightly coupled with a specific execution module (which can provide physical access to DCIs such as the Grid and Cloud Engine of the CSGF), can help enable easy access to computational web-based services. The CSGF approach provides a solution in which a portal technology (such as Liferay portals) is proposed. It also makes use of the Java implementation of the SAGA standard and therefore ensures that any method of communication with the targeted DCIs is only possible via the java libraries. This approach is adopted for both the sequential and the parallel version of the portlets that were developed for the Infection Model.

3. FutureGateway (on the other-hand) is largely developed to enable a solution that is not completely dependent on portal technologies. Therefore, communities of practice can utilise an existing portal or any web technology (which may already be in use) in their own local

domain. Furthermore, API server is responsible for interacting with the Grid Engine and not portlets (as seen in the case of CSGF). In addition, the use of Restful APIs for the interaction with the targeted infrastructures is performed using different programming languages as opposed to the CSGF approach where the java libraries is tightly coupled with SAGA. The WEKA – J48 portlet is therefore developed using this approach. Even though a portal framework (such as Liferay) was also chosen as the platform of choice for developing the WEKA – J48 portlet, this option is more flexible and can therefore be used with any other web technologies.

The different approaches show a steady evolution in the way in which web-based access to computational services is viewed, from the early days of windows client using Java programs to web-based content using Liferay portals or similar content management system and to the more recent use of RESTful-APIs for the interactions with DCI resource.

## *7.4 The Infection Model Portlet and the WEKA portlet in comparison with real life use-cases of the Sci-GaiA project*

Section 7.3 briefly discusses the approaches that were taken to develop both portlets by comparing the methods used in the design and implementation of both the Infection Model portlet and the WEKA – J48 portlet. In this section however, an attempt is made to evaluate both versions of the Infection Model portlet and the WEKA – J48 portlet within the context of real case study scenarios. This evaluation is performed by comparing the adopted approaches used in the creation of both portlets with some real-life case studies developed and deployed on different Science Gateways for different communities of practice. More importantly, these case studies have been developed using either the CSFG or the more recent FutureGateway APIs.  They were supported by Sci-GaiA - a project aimed at creating a sustainable foundation of educational materials and the procedures for the management of Science Gateways and e-Infrastructures in Africa and beyond.

These sets of case studies specify different requirements such as security requirements, user requirements, software requirements, data requirements and system requirements as defined by the communities who need them. The mandatory security requirements necessary to access these applications include the different privacy levels, certification and access mechanism which the Science Gateway adopts to safeguard the different scientific software application on the Science Gateway. The mechanisms in place for both the Catania Science Gateway Framework and the FutureGateway API for authenticating and authorising users to the scientific software applications on the Science Gateway (as adopted by the aforementioned scientific applications) have been thoroughly discussed in Chapter 2. The user requirements, on the other hand, consist largely of the GUI requirements where users can specify, run and execute jobs on different DCIs. The need to utilise some web-based content (such as Liferay portal) or any other web-based content management already in use by a specific scientific domain to facilitate access to DCIs have earlier been explained. Furthermore, data requirements may involve all data related activities between the frontend and the backend or the runtime data transfers (and storage during execution) which are facilitated by the Grid and Cloud Engine of the Catania Science Gateway Framework (discussed in Chapter 2).

The required software stack may depend on different scientific software applications and may vary from an application server (For hosting the individual components/applications) and the need for database connectivity to keep track of users, service actions and states. Finally, the system requirements may vary from one scientific software application to another and may include access to different DCIs such as clusters, grids, desktop grids and cloud resources or resources such as sensors/instrumentation, data repository, data analysis support, etc. These DCI resources may help communities of practice to alleviate the need for more CPU, memory or storage requirements as required by each scientific software application. These resources (as required by these communities) are therefore summarised in Table 7.1 below.

Table 7.1 presents a summary of the different scientific applications that are supported by the Sci-GaiA project. It shows the classification of different applications based on the scientific domain which they belong and most importantly the different DCIs that were exploited by each of these applications.

Table 7.1 Summary of the use-cases from both the Sci-GaiA project and the e-Infrastructures for Africa project, and the corresponding e-Infrastructure resources that were utilised.

| **Scientific Use-Cases** | **SW Access Support** | **Data Repo Support** | **Support for Powerful Comp. Resources** | **Data Analysis is Support** | **Peer Collaboration Support** | **Connection to Sensors and Instrumentation** | **Scientific Domain** | **Main Characteristics** |
|---|---|---|---|---|---|---|---|---|
| Education E-library & MOOC platform | YES | YES | NO | NO | NO | NO | IM | Publications, Reviews, E-library |
| Addis Ababa Riverside Monitoring System | YES | YES | YES | YES | NO | NO | Earth | River, Riverside Monitoring |
| e-AgriSERVICOMM | YES | YES | YES | YES | NO | NO | Agriculture | Disseminate information to farmers |
| SGW-based plant Tissue | YES | YES | YES | YES | NO | NO | Agriculture | Plant tissue data |
| Documentation & Archiving of Open Data | YES | YES | NO | NO | NO | NO | IM | Meta-Data Archiving, Documentation of survey Data sets. |
| ACEPRD Plant Repository | YES | YES | NO | NO | NO | NO | Health | Pharmaceuticals Plant Derived Medicines, Plant Repository |
| Real Time Meteorological Data Repository | YES | YES | NO | NO | NO | YES | Earth | Atmospheric Dataset |
| Framework for NGS Data Analysis | YES | NO | YES | YES | NO | NO | Health | Next Generation Sequencing, Data Processing |
| Interactive Pipeline for Genome | YES | YES | YES | YES | NO | NO | Health | Genome/DNA Study |
| Drug, Design, Discovery, Development Platform Repo | YES | YES | NO | NO | NO | NO | Health | Drug Products/Repo |
| Web Based Predictive Defuzzifier | YES | NO | YES | YES | NO | NO | Health | Fuzzy Logics |
| IBIS Framework | YES | NO | YES | YES | YES | NO | Health | Biological Data |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| MIPAR Extension | YES | YES | YES | YES | NO | NO | Health | Medical Images/Image Repo |
| REPAST | YES | YES | YES | NO | NO | NO | Health | Infection Model |
| iGrid Portal | YES | YES | NO | NO | NO | NO | IM | Smart Grid |
| WRF Integration | YES | YES | YES | YES | NO | YES | Earth | Weather Research, Forecasting Model. |
| WEKA | YES | NO | YES | NO | NO | NO | Health | Breast Cancer Datasets/Data Mining Tasks |
| Kenya National Public Health Gateway | YES | NO | YES | YES | NO | NO | Health | Data on Motorbike accidents, Public Health Experts |
| Speech Recognition System | YES | YES | YES | NO | NO | NO | IM | Speech Recognition System |
| TTA Collaboration Platform | YES | YES | YES | NO | YES | NO | IM | Collaboration Platform |

In the above Table 7.1 are the different set of scientific use-cases/applications which span four (4) different sets of scientific domains. These scientific domains include health, agriculture, environment, and information management. A classification was therefore performed based on these different scientific domains and their need for different DCIs to conduct their scientific experiments. Scientific needs may therefore fall into one or more categories such as:

1) The need to support access to software applications for performing experimentation.
2) The need to support access to large data sources.
3) The need for more powerful computational resources.
4) The need to connect to specialised instrumentation for analysis.
5) The need to collaborate with other scientists across the world.

Conventionally, Science Gateways enable scientific domains to seamlessly exploit the aforementioned distributed computing and storage resources via a common interface which is normally configured to fulfil the specific requirements of the particular community of practice. As a result, scientists or potential users can focus on their scientific goals and thus worry less on the required DCIs. The diverse set of scientific applications (together with their associated e-Infrastructural needs) ensure that processes such as software provisioning are complex and difficult to manage. The above Table 7.1 shows that several scientific applications require different types of e-Infrastructure resources. These e-Infrastructures are used to support several e-Science activities and operations such as the global collaborations between scientists and researchers across many various fields, fuelled by sophisticated software applications and supported by advanced Information and Communication Technologies (ICT). These ICT which combines to collectively form the e-Infrastructures include but not limited to high-speed research communication networks, powerful computational resources (such as dedicated high-performance computers, clusters, and large numbers of commodity PCs), Grids and Cloud technologies, data infrastructures (data sources, scientific literature), sensors, web-based portals, scientific gateways and mobile devices. All these resources have been captured and adapted as required by the different communities shown in Table 7.1.

The above Table 7.1 indicates that majority of the communities need easy access to scientific software applications as well as an extra access to data repository. These software applications are enabled via a Science Gateway to allow multiple users across the world to access and execute applications in an e-Infrastructure environment. Science Gateways which is at the very heart of this research are often integrated via a portal or a suite of applications (usually in a graphical user interface) which could be further customised to meet the needs of specific communities such as the ones listed above. The resources which are represented in Table 7.1 are mostly back-end services which are required for the execution of the scientific software applications and thus form a major part in the development of applications within Science Gateways in general.

The need for the easy access and use of scientific software application was identified by most of the different communities of practice as shown in Table 7.1 where the support for an easy access to these applications is prioritised by the majority of the communities. Furthermore, the support for more powerful computational resources is also of utmost importance as shown in Table 7.1. Across many different fields the need for an interdisciplinary global collaboration between scientists and researchers (which has been fuelled by these sophisticated software applications) ensures the need for powerful computers to run these applications. A few of the communities only require the access to sensors and instrumentation. However, a considerable amount of these scientific applications also require an extra support for both data repository as well as the capability for future data analysis activities.

Consequently, the different scientific applications represented in Table 7.1 are divided into two categories. The first category of applications represents the communities that require an easy access and use of their scientific application in the context of e-Infrastructures. In addition, they also require the need for powerful computers to execute these applications. This category of applications was developed using the Catania Science Gateway Framework approach which is tightly coupled with a particular web-based content management (such as the Liferay portal) and adopts the Catania Grid and Cloud Engine for interacting with the various DCIs via the JSAGA interface. The method that was used in the development of Infection Model portlet (which has been explained in great detail in Chapter 5 and briefly discussed in Section 7.3) is similar to the approach that was used to develop the last seven use-cases in Table 7.1. The second category of applications (the first 13 use-cases in Table 7.1) represent the communities that mostly require (in addition to the need for an easy access to their scientific applications and the use of powerful computers for their execution) the access to data repository as well as data analysis capabilities. These applications therefore constitute the second group of scientific applications. This group of applications were developed using the FutureGateway API approach where portlets development are not tightly coupled with a particular content management system and communities of practice may therefore opt to utilise an existing portal or any other web technologies already in use in their local domain. Furthermore, the use of Restful APIs for interacting with the targeted infrastructures is done using varieties of programming languages as opposed to the use of Java libraries as seen in the CSGF approach. This approach used in developing these applications is similar to the one used for porting the WEKA application to the Science Gateway which was briefly discussed in Section 7.3 and detailed in Chapter 5. Although the Liferay portal framework was used in the development of the WEKA – J48 portlet, this approach for developing applications has the flexibility of adopting other similar web technologies. It also has the flexibility of using APIs to execute jobs on the DCIs.

In light of the above discussions, it is clear the proposed MESSAGE methodology in Chapter 4 took into consideration only the computational aspect of job execution. However, majority of the scientific applications in Table 7.1 falls under the second category where the easy access and use of scientific applications and powerful computational resources are equally as important as the need to have data repository and data analysis capabilities when porting scientific software applications in Science Gateways. Consequently, these additional capabilities are included as part of the revised MESSAGE methodology for developing scientific software applications in Science Gateways which is reviewed in the latter sections of this Chapter.

## *7.5 The Infection Model Portlet and the WEKA Portlet in comparison to real life use-cases of the CloudSME project*

Similar to the Sci-GaiA project which supports the execution and management of different scientific applications, the CloudSME simulation platform also supports the development and deployment of scientific software applications across multiple clouds and mostly to facilitate the use of High-Performance Computing. In particular, many simulation software vendors who have identified different needs (in the context of e-Infrastructures) for their simulation applications have developed cloud-based versions of their software. This activity is performed in close collaboration with their end-user partners in the CloudSME case study experiments with a view to developing applications with partners and potentially across the European manufacturing and engineering sector.

More importantly, the development and deployment of the different scientific software applications which are discussed in this section are supported by a well-known Science Gateway framework (otherwise known as the gUSE/WS-PGRADE) which has been explained in great detail in Chapter 2. The CloudSME project therefore presents different set of case studies which are developed and executed on DCIs using a different Science Gateway Framework other than the Catania Science Gateway Framework and FutureGateway APIs.

Similar to the case studies that were supported in the Sci-GaiA project, the CloudSME case studies have specified different security requirements, user requirements, software requirements, data requirements and system requirements. The different scientific software application needs and requirements have been explained in Section 7.4 above. The CloudSME case studies which were supported in the project include twelve (12) distinct case studies and comprises of different communities such as Environment fluid mechanics, Business process modelling by simul8, Simulation application templates by simul8, 3D scan insole design, Aircraft maintenance by 2Moro solutions, Accessing Cloud for emissions reduction (ACER), Cloud-based simulation models for freight transport intermodals terminals, ComfortFeet in safety shoes, Cloud-based workflow for CFA and FEA simulation, Development and application of Cloud-based simulation template for the Craft brewing industry, Simulation-based optimisation of business processes and Inventory analysis and simulation forecasting cloud-based software.

The majority of the use-cases which were supported in the aforementioned CloudSME project are mostly simulation software applications. Therefore, they share similar or same functionalities, requirements and DCI resources as well as other resources which are required for their development, deployment and the subsequent execution in a distributed environment. All the aforementioned use-cases provide simulation and modelling support capabilities for the different communities in the context of e-Infrastructures. Since the use-cases are fundamentally similar in nature (i.e. simulation software applications), their e-Infrastructure requirements usage is mostly summarised below as:

- The need to provide support for a Graphical User Interface service where users can develop and execute complex workflows as well as monitor their results. This capability is enabled by the WS-PGRADE (the graphical user interface service). The available GUIs include the Workflow Developer UI, End-User User Interface and the Application Specific User Interface. There is also the option of running the GUIs on the cloud or on the desktop.
- The need to provide mechanisms for customising portals according to specific requirements based on specific scientific domain (which is mainly powered by Liferay portal).
- The need to secure access to these resources using mechanisms such as certificate and access mechanisms, etc. as provided by the WS-PGRADE/gUSE framework.
- The need to provide support for runtime data transfers and storage during these simulation activities as well as provide support for archived data storage and management which could range from 100MB to 1TB per user.

- The need to provide support for powerful computational resources in order to ensure that the connection between the frontend and backend is reasonably fast most especially, where running simulation experiments are concerned.
- The need to execute these scientific simulation software on federated Cloud resources.
- Additionally, to ensure that users can create and run workflows on various DCIs (supported for the aforementioned applications), the DCI Bridge service (described in Chapter 2) which enables access to most of the popular European DCIs is utilised. This therefore provided the needed support for accessing various DCIs.

The above e-Infrastructure need or use for each scientific simulation application in the CloudSME project draws some parallels with the requirements that were defined and utilised throughout the development and implementation of both the Infection model portlet and the WEKA – J48 portlet. It is also consistent with the requirements that were defined in the Sci-GaiA projects which was earlier discussed. The comparison of the Infection Model portlet and the WEKA – J48 portlet with the CloudSME project and the Sci-GaiA project, and the similarities in the methods used therefore show the generic nature of the proposed MESSAGE methodology in Chapter 4. As such, this process has demonstrated the generic nature of the MESSAGE methodology such that it is applicable to the different scientific software applications executed and managed across two of the Science Gateway frameworks that was discussed in Chapter 2.

The MESSAGE methodology in Chapter 4 however, took into consideration just the computational aspect of developing and deploying scientific software applications in Science Gateways. But, as revealed by the scientific case studies of the Sci-GaiA projects and the CloudSME projects, while some scientific software applications require extra support for data analysis (most especially in the Sci-GaiA project), majority of the scientific software applications require an extensive support for data repositories in the case of both projects. Nonetheless, the MESSAGE methodology remains a generic methodology which may be utilised to develop scientific software applications in Science Gateways by using the different Science Gateway Frameworks discussed in Chapter 2. However, for the revised MESSAGE methodology, data repository/data analysis capabilities are incorporated as part of the overall process. This will consequently serve as the revised MESSAGE methodology for developing scientific software applications in Science Gateways which is discussed in the following section.

## *7.6 The Revised Methodology for Developing Scientific Applications in Science Gateways*

In the above Section 7.4 and 7.5 attempts are made to analyse the adopted methods in the development of the different sets of scientific software applications in Science Gateways. More so, the general requirements and functionalities needed for the development and deployment of these applications are discussed. These different sets of applications are developed using two Science Gateway frameworks. Consequently, the development of these applications was done by customising the Science Gateway framework which the different projects have adopted (namely the CSGF/FutureGateway API framework and the gUSE/WS-PGRADE Science Gateway framework). Therefore, the development of these applications followed a similar pattern based on the Science Gateway framework that was adopted. However, the system functionalities and the e-Infrastructure resources in use may vary depending on the type of scientific application being developed. Furthermore, the adopted portal/web framework, web container and the utilised database management system are discussed. In addition, the job executors that were adopted by each application are also analysed. The adoption and use of a particular Science Gateway framework by each of these applications ensures the automatic use of a job executor which could either be DCI-bridge (in the case of the use-cases of the CloudSME project) or the Grid/Cloud engine/JSAGA/FutureGateway API (in the case of the use-cases of the Sci-GaiA project).

For the Sci-GaiA project however, in addition to the e-Infrastructure resources which are commonly in use, most of the scientific software applications also require access to data repository and data analysis capabilities. These are peculiar to the communities of practice who want to utilise all aspects of their research (such as data, software, results/research outputs) and make them visible to other communities for validation, re-use and collaboration purposes.

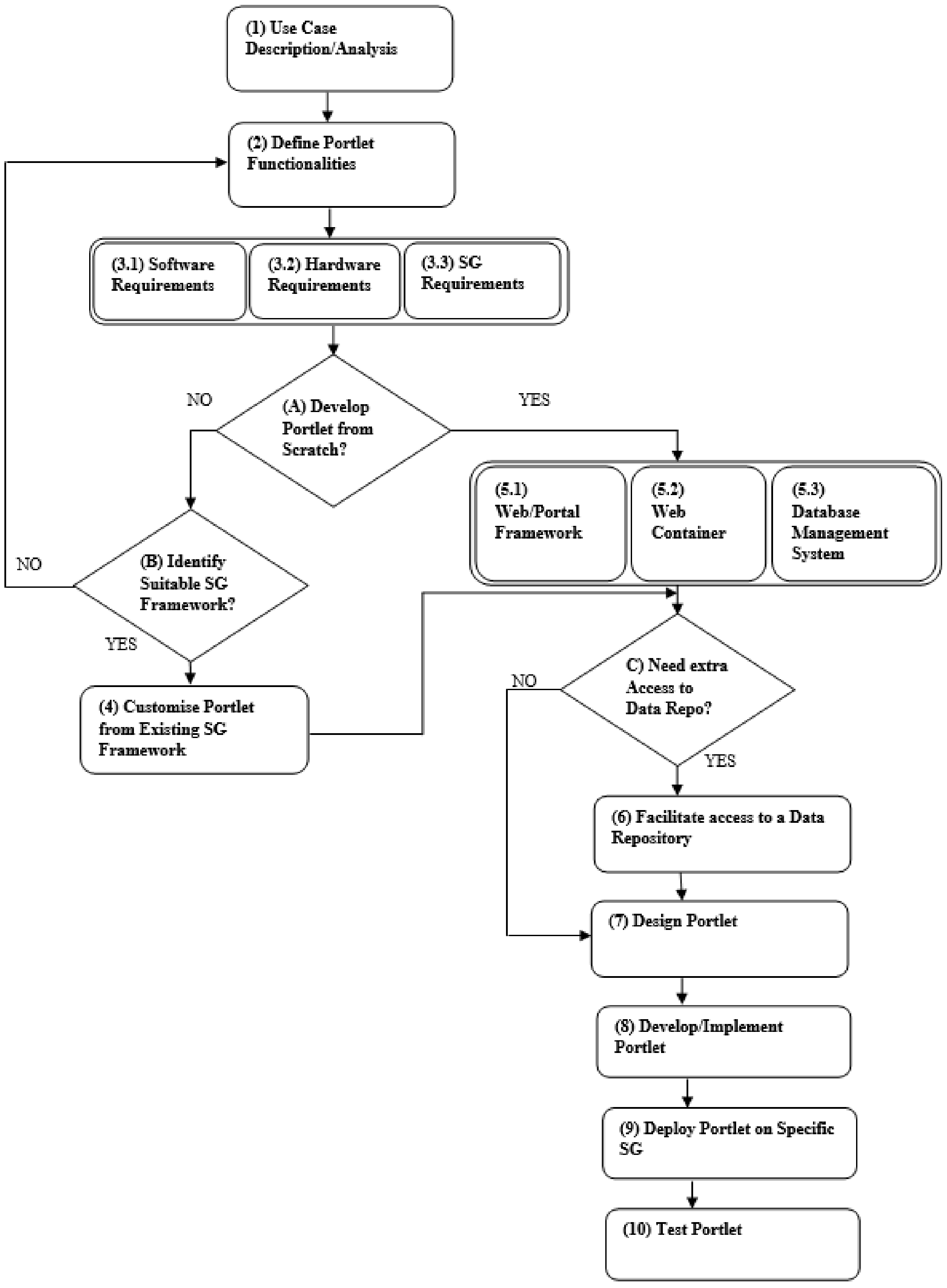

(1) Use Case Description/Analysis
(2) Define Portlet Functionalities
(3.1) Software Requirements
(3.2) Hardware Requirements
(3.3) SG Requirements
NO
(A) Develop Portlet from Scratch?
YES
(5.1) Web/Portal Framework
(5.2) Web Container
(5.3) Database Management System
NO
(B) Identify Suitable SG Framework?
YES
(4) Customise Portlet from Existing SG Framework
NO
C) Need extra Access to Data Repo?
YES
(6) Facilitate access to a Data Repository
(7) Design Portlet
(8) Develop/Implement Portlet
(9) Deploy Portlet on Specific SG
(10) Test Portlet

Figure 7.1 The Revised Methodology for Developing Scientific Software Applications in Science Gateways (MESSAGE)

The MESSAGE methodology in Figure 7.1 shows the revised approach to the MESSAGE methodology for developing scientific software applications in Science Gateways which was proposed and discussed in Chapter 4. The proposed MESSAGE methodology describes the processes involved in developing an application in Science Gateways, from the case study description to the final deployment of portlet on the Science Gateway. However, the proposed MESSAGE methodology focused more on developing customised user interfaces for different user communities as well as the computational aspect of job execution. In the Sci-GaiA project that was discussed in Section 7.4, the majority of the scientific software applications also require the capability to connect with data repositories to store different kinds of research outputs such as data, software, results, etc. In essence, jobs require access to data storage when they are executed. Therefore, the MESSAGE methodology in Chapter 4 is revised to facilitate this capability as shown in Figure 7.1. Consequently, more than just attempting to make scientific software applications easily accessible and readily available worldwide, this approach also facilitates the accessibility and availability of the associated research outputs which are generated as a result of utilising associated scientific software applications.

The idea of making research outputs easily accessible and available (worldwide) is known as open science. Open science aims to promote open access to all research artefacts. The issues surrounding open science, and efforts or endeavours to make scientific research easily accessible and available to the scientific communities that could benefit from its use is what the Sci-GaiA project aims to achieve. In addition to Sci-GaiA, there are similar initiatives that aim to promote open science, such as the Open Knowledge Foundation (OKF) and the Center for Open Science.

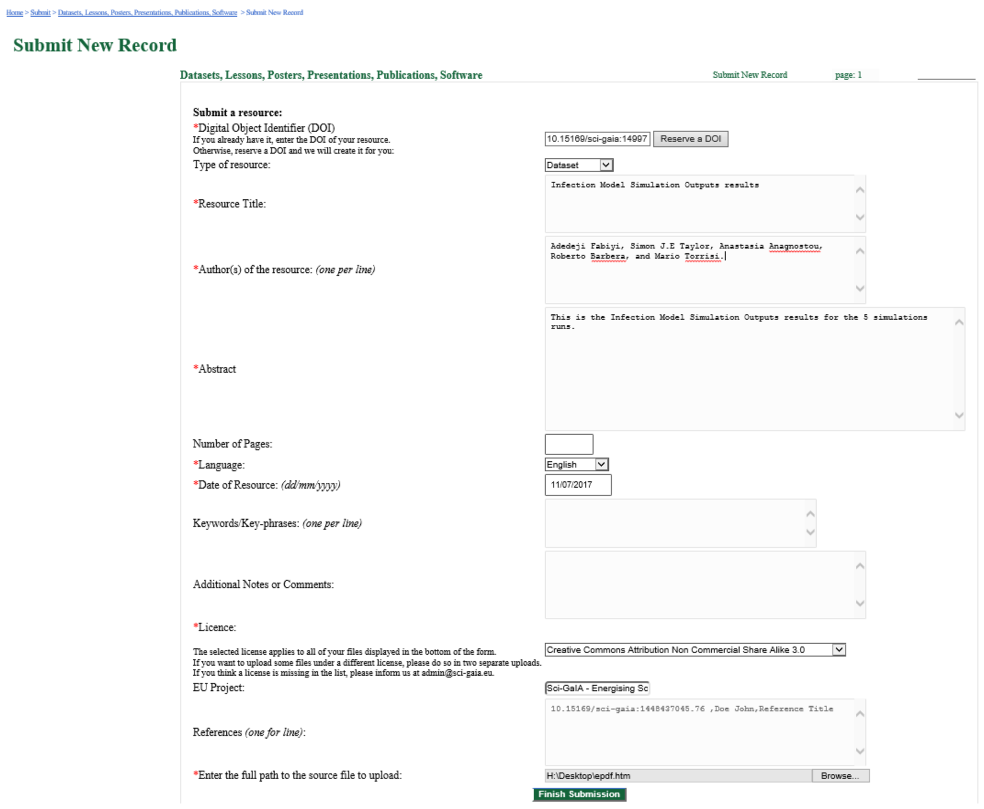
It is important to note here that while the MESSAGE methodology in Figure 7.1 provides a foundation for open science, the concept of open science is actually beyond the scope of this research. In developing the processes which can potentially aid with the development and deployment of scientific software applications and make them accessible and available to other communities, the associated research outputs such as the related data and final results (generated from the execution of the aforementioned scientific applications) can also benefit from a similar approach. This approach will ensure that all associated research products are easily accessible and readily available for the purpose of future validation, collaboration and re-use. In the following section, the Infection Model simulation outputs is used to demonstrate the procedures which are required to make research outputs easily accessible and available (worldwide) by utilising enabling technologies such as the Open Access Document Repository (OADR), Digital Object identifiers (DOI) and the Open Researcher and Contributor ID (ORCID) in the context of e-Infrastructures.

## *7.7 Availability, easy access and/or retrieval of scientific research outputs in the context of an e-Infrastructure: Demonstration of the Infection Model results (outputs)*

It was earlier mentioned that potential users of an e-Infrastructure may need the platform for a number of activities such as access to more powerful computer to run applications, specialised HPC resources, specialised instrumentation for analysis, large data sources, sensors for data collection and access to software. By using a valid credential/certificate the procedures for making an ABMS software readily available and easily accessible (via a science gateway interface) have (thus far) been presented in this thesis.

In addition to making ABMS scientific software easily accessible and readily available to international communities (by leveraging on the different DCI resources), other related research outputs such as ABMS model, data and final results (such as the simulation output results) can also be made available and accessible by using appropriate e-Infrastructure resources. A typical ABMS experiment may include the use of the software package, the simulation runs, generated data and output collection for subsequent analysis. All of these aforementioned entities could (more or less) serve as the outputs necessary for modelling/simulation experiments. Another research output may be the final report of an experiment (such as a scientific paper) which could be published in a conference or journal (Taylor, Fabiyi et al. 2016). Usually, the aforementioned research outputs are not readily available for public use except for published papers which could be available for free or subject to certain access agreement. However, to encourage the replication, reproduction and reuse of scientific research, such research outputs and methods must be openly available/shared. Consequently, this can benefit other aspects of research such as collaboration and validation processes.

In this section, the simulation output (results) obtained from the different simulation runs of the Infection Model portlet is used to demonstrate how research outputs are accessible and readily available to research communities. This approach is enabled by using different technology stacks such as Digital Object identifiers (DOI), Researcher Registries (RR), and the Open Access Data Repositories (OADR). An Open Access Data Repository (OADR) is a repository for storing or depositing all research outputs. Conventionally, all objects stored on the OADR have DOIs assigned to them. DOIs are persistent digital identifiers of objects which are usually assigned by agencies with permission to assign them. As such, the outputs of a simulation (or data) also have DOIs assigned. All the aforementioned technology stacks have been made available by an initiative which promotes open science (Energising Scientific Endeavour through science gateways and e-Infrastructures in Africa - Sci-GaIA) to make accessible and available all the relevant research output. Consequently, the following section will demonstrate the use of DOI and OADR on the simulation outputs/results of the Infection Model portlet and how these research outputs are accessible and readily made available to the communities that need them.


Home > Submit > Datasets, Lessons, Posters, Presentations, Publications, Software > Submit New Record

Submit New Record

Datasets, Lessons, Posters, Presentations, Publications, Software — Submit New Record — page: 1

**Submit a resource:**

| | |
|---|---|
| *Digital Object Identifier (DOI)<br>If you already have it, enter the DOI of your resource.<br>Otherwise, reserve a DOI and we will create it for you: | 10.15169/sci-gaia:14997 [Reserve a DOI] |
| Type of resource: | Dataset |
| *Resource Title: | Infection Model Simulation Outputs results |
| *Author(s) of the resource: *(one per line)* | Adedeji Fabiyi, Simon J.E Taylor, Anastasia Anagnostou, Roberto Barbera, and Mario Torrisi. |
| *Abstract | This is the Infection Model Simulation Outputs results for the 5 simulations runs. |
| Number of Pages: | |
| *Language: | English |
| *Date of Resource: *(dd/mm/yyyy)* | 11/07/2017 |
| Keywords/Key-phrases: *(one per line)* | |
| Additional Notes or Comments: | |
| *Licence:<br>The selected license applies to all of your files displayed in the bottom of the form.<br>If you want to upload some files under a different license, please do so in two separate uploads.<br>If you think a license is missing in the list, please inform us at admin@sci-gaia.eu. | Creative Commons Attribution Non Commercial Share Alike 3.0 |
| EU Project: | Sci-GaIA - Energising Sc |
| References *(one for line)*: | 10.15169/sci-gaia:1448437045.76 ,Doe John,Reference Title |
| *Enter the full path to the source file to upload: | H:\Desktop\epdf.htm [Browse...] |

Figure 7.2 Open Access Repository showing the “Submit New Record” Page

The OADR may be located at http://oar.sci-gaia.eu/. Similar to the other DCIs, depositing and submitting documents to the OADR requires users of the infrastructure to be an authorised user of the repository. The authentication process to the repository also follows a similar approach to the Infection Model portlet on the Africa Grid Science Gateway which was earlier described in Chapter 5. By clicking on the submit tab on the OADR (See Appendix A.5), different resource types such as datasets, images, audio/video recordings, posters, presentations and software can be uploaded. However, for this demonstration the five different datasets which are obtained from the simulation runs of the Infection Model portlet is explored. On the submit page, datasets is chosen as the type of document to be submitted and (on the proceeding page) ‘Datasets Sci-GaIA’ is selected as the category where the content should be submitted (See appendix A.6). Subsequently, by clicking on the ‘Submit new record’ button, a page which consists of the submission form which is used for reserving a DOI for the Infection Model portlet outputs is presented as shown in Figure 7.2. A DOI may be entered either in the DOI input field (if the researcher already has one) or generated automatically by clicking the ‘Reserve a DOI’ button. In the case of the Infection Model simulation output results, DOIs were simply reserved for the five datasets.

Furthermore, the resource title, author(s) of the resource, type of resource, category of the project, abstract and date of the resource and the applicable licence for the resource are specified for the Infection Model outputs. Finally, the Infection Model outputs are uploaded and submitted to the OADR by clicking on the “finish submission button”. After this submission, each simulation output result is assigned a reference (such as DATASETSSCIGAIA-2017-003) as well as a URL (https://oar.sci-gaia.eu/record/577). As the submission and publication of the research product on the OAR are not automatic, the simulation output results are inserted in a bibliographic task queue where subsequent execution can take place. This is to ensure that the managers of the OAR can analyse the submitted research products such as its meta-data information. Subsequently, the publication is made and the research output can thus become visible worldwide. An example of a reseach output that was deposited in the OAR for a graphical visualisation tool for REPAST Infection Model document can be found at http://dx.doi.org/10.15169/sci-gaia:1460675843.23 and the virtual appliance to simulate an Infection Model implemented with using REPAST simphony can be found at http://dx.doi.org/10.15169/sci-gaia:1455182324.71.

To demonstrate the accessibility of such research outputs for the Infection Model therefore, the list of outputs and their corresponding DOIs from an earlier work by (Taylor, Fabiyi et al. 2016), in which the same sets of Infection Model research outputs were deposited on the OAR are presented below as follows:

REPAST Infection Model Virtual Appliance http://dx.doi.org/10.15169/sci-gaia:1455182324.71

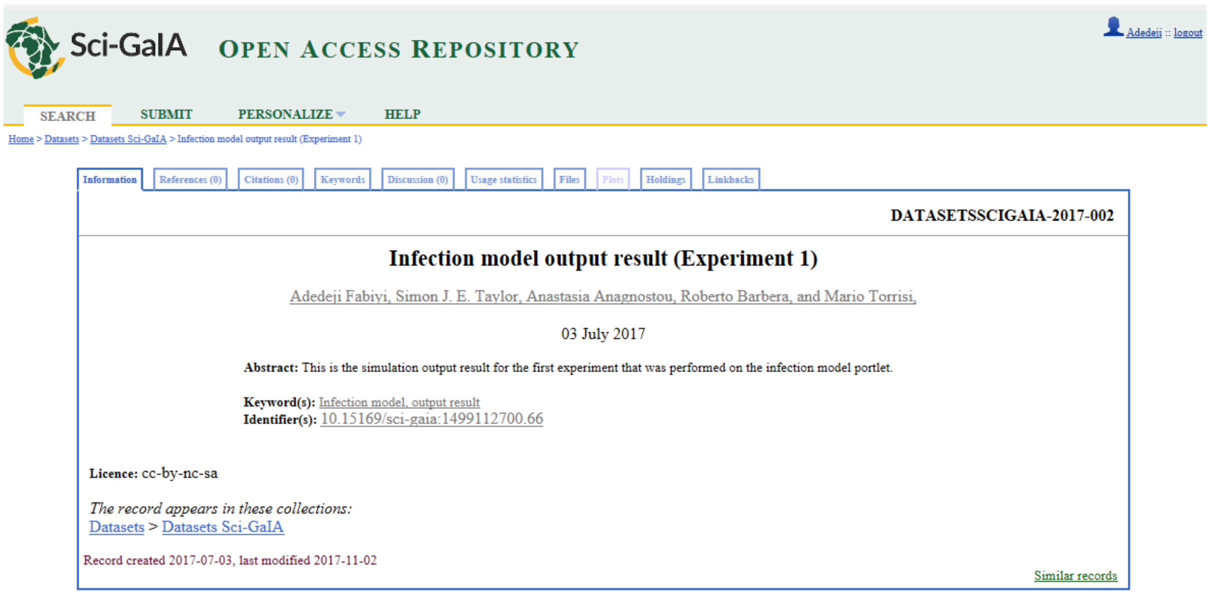
Graphical Visualisation Tool for REPAST Infection Model http://dx.doi.org/10.15169/sci-gaia:1457432416.29

REPAST Infection Model Experiment 1 Results http://dx.doi.org/10.15169/sci-gaia:1499112700.66

REPAST Infection Model Experiment 2 Results http://dx.doi.org/10.15169/sci-gaia:1499114566.48

REPAST Infection Model Experiment 3 Results http://dx.doi.org/10.15169/sci-gaia:1499114874.12
REPAST Infection Model Experiment 4 Results http://dx.doi.org/10.15169/sci-gaia:1499115090.39
REPAST Infection Model Experiment 5 Results http://dx.doi.org/10.15169/sci-gaia:1499115487.71

The general information for this submission as well as the references to the research outputs have been represented in Figure 7.3.



DATASETSSCIGAIA-2017-002

Infection model output result (Experiment 1)

Adedeji Fabiyi, Simon J. E. Taylor, Anastasia Anagnostou, Roberto Barbera, and Mario Torrisi,

03 July 2017

**Abstract:** This is the simulation output result for the first experiment that was performed on the infection model portlet.

**Keyword(s):** Infection model, output result
**Identifier(s):** 10.15169/sci-gaia:1499112700.66

**Licence:** cc-by-nc-sa

*The record appears in these collections:*
Datasets > Datasets Sci-GaIA

Record created 2017-07-03, last modified 2017-11-02

Figure 7.3 Information Describing the Submitted Research Output

Finally, the Infection Model research outputs which are now visible and easily accessible worldwide is uniquely identified to a particular researcher who has deposited the research outputs on the OADR. This is made possible using the Open Researcher and Contributor ID (ORCID) (also see https://orcid.org/) which provides a persistent digital identifier that ensures that one researcher is distinguished from another. In addition, the ORCID also provides an automatic link between all of an individual researcher' professional activities and thus ensures that all their work are recognised. By making use of this unique ID therefore ensures that researchers are able to claim their work/outputs. Thus the researcher's ORCID is orcid.org/0000-0002-7797-8272 and the link to the public record submitted is http://orcid.org/0000-0002-7797-8272.

To conclude this Chapter, the different aspects of the MESSAGE methodology which has been tested through the implementation of the case studies in Chapter 5 is highlighted. It is important to note here that only one part of the MESSAGE methodology has been tested due to the decision of the researcher to customise an existing Science Gateway framework as opposed to developing the Science Gateways from scratch. As such, the use case description (1), define portlet functionalities (2) and the different software (3.1), hardware (3.2), and Science Gateway (3.3) are demonstrated. Also, at the decision point in (A), the researcher opted not to develop the portlet from scratch which consequently led to the decision point in (B) where a suitable Science Gateway framework is identified. At this point, the decision to customise the portlet from an existing Science Gateway framework (4) is made. Also, the easy accessibility and availability of the associated research outputs was demonstrated, therefore the decision point in (C) explores the possibility of an extra need for data repository. The use of data repository in (6) was therefore included as part of the MESSAGE methodology and was demonstrated using the simulation output results. In addition, the different stages involved in the actual implementation of the portlet using a suitable SDLC was demonstrated in the design of the portlet (7), develop/implement the portlet (8), deploy the portlet on specific Science Gateway (9) and test the portlet (10). However, the other part of the MESSAGE methodology (in web/portal framework (5.1), web container (5.2) and database management system (5.3)) where the researcher needs to develop the Science Gateways from scratch by specifying and setting up the different configurations (for the web/portal framework, web container and database management system) will be tested as part of the future work in this research.

## *7.8 Summary*

In this Chapter, the method that was used to develop the Infection Model portlet as well as the WEKA – J48 portlet was thoroughly evaluated. There are different approaches which have been utilised for the evaluation of both portlets. The first approach involves comparing the method for developing both the Infection Model portlet and the WEKA – J48 portlet. Secondly, the method used for developing both portlets was compared with the development of several applications of two different EU projects (i.e. the Sci-GaiA project and the CloudSME project).

While the Sci-GaiA project developed its use-cases using similar Science Gateway framework to the Infection Model and the WEKA application, the use-cases of the

CloudSME project on the other-hand utilised a different Science Gateway Framework (known as the gUSE/WS-PGRADE Science Gateway Framework). To generalise the MESSAGE methodology, its application and use by other Science Gateway framework (discussed in Chapter 2) is therefore justified. This was demonstrated by comparing with the scientific applications that were developed using gUSE/WS-PGRADE Science Gateway Framework in the second EU project. Consequently, a revised MESSAGE methodology is proposed based on the evaluation that was performed throughout this Chapter. This Chapter wraps up with a demonstration of how research outputs such as the Infection Model simulation output results (in addition to their associated scientific applications) can be easily accessible and readily available to scientists and researchers for further scientific activities such as for collaboration and/or validation purposes. Such enabling technologies utilised in the context of e-Infrastructures as discussed in this Chapter include OADR, DOI and ORCID.

# Chapter 8: CONCLUDING DISCUSSION AND REMARKS

## *8.1 Overview*

In this Chapter, a summary of the major highlights of the entire thesis is discussed in Section 8.2. Also, based on the research aim and objectives that were identified in Chapter 1, Section 8.3 discusses how those objectives were met in order to achieve the aim of the thesis. More importantly, the contributions of this research is emphasised and discussed in Section 8.4. Furthermore, while the research limitations and reflections are presented in Section 8.5, Section 8.6 (on the other hand) discusses the future research possibilities based on conducting this study.

## *8.2 Research Summary*

This research study was motivated by the fact that several potentials and capabilities of the advanced Information and Communication Technologies (ICT) may not be adequately harnessed due to the approaches and methods provided by traditional means for accessing them. Several issues relating to easy accessibility and availability of these resources therefore present a major setback for the adoption of these resources by the communities that may benefit from their use. The concept of Science Gateways which are "community specific set of tools, applications and data which are integrated via a portal or suite of applications, usually in a graphical user interface and further customised to meet the needs of a specific community" have been introduced to simplify access and to overcome the initial difficulties that were faced by the users of the infrastructures (Ardizzone *et al.*, 2012).

In light of this, the hypothesis in this research is therefore the feasibility of creating a methodology for the development of scientific software applications in these Science Gateways which can subsequently be used for accessing these infrastructures. To achieve this, a research methodology which consists of a mix of DSR as well as a reflective evaluation study was identified very early on in this research. Three distinctly different research types based on the research questions and the research objectives were used in the course of this study. These research types include Exploratory, Design and Development, and Reflective Evaluation. A careful study of the different methodologies however, reveals that the DSR cycle maps well with the aforementioned research types which further justify the use of the adopted research methodology.

According to the literature, the ease of access to distributed resources is one of the major needs of most communities of practice (with respect to the adoption of several of these DCIs) as the perceived complexities of understanding and managing the underlying infrastructure and software could prove challenging. In addition, a further review of the literature and the major highlights obtained from Chapter 2 reveals the high level functionalities of generic Science Gateways. These include security management, job management, workflow management and data management. In Chapter 4, these functionalities are simply referred to as the generic framework of Science Gateway services.

The MESSAGE methodology (in Chapter 4) was created by considering two major aspects of Science Gateways namely the front-end and the back-end components. The front-end component is typically designed to serve users and to provide the necessary user interface for target user communities. The back-end component, on the other hand, enables the necessary DCI mechanisms and manages jobs as well as service calls. The sequence of steps that were used to realise this methodology (to cater for both software and hardware components) begin with the general requirement specifications (such as software, hardware and the Science Gateway requirements) which are typically required to support specific scientific software applications. Furthermore, the process emphasised the need to define system functionalities, which could be specific to the needs of a particular community (and could vary from one community to another), or generic to different communities of practice.

In the proposed MESSAGE methodology, the next stage of the process flow describes Science Gateway approaches which could be utilised to develop and deploy fully functional applications in Science Gateways. The first approach is to develop the portlet from scratch with the aid of specific portal or web application framework while the second approach involves customising application specific Science Gateways from existing Science Gateway frameworks. There are several Science Gateway frameworks available to customise from, most of which were discussed in Chapter 2. In addition, the need to adopt different software stacks (such as the portal framework, the web application container and the database management system), in the development of these scientific software applications was emphasised. The process flow concludes with the latter stages of the MESSAGE methodology which captures the step involved in the actual software creation (within a portal context), from the design of the portlet to its deployment on the Science Gateway. The relevant aspects of the adopted SDLC (Waterfall model) was adapted to capture this phase.

In order to implement the MESSAGE methodology, three different scientific applications (Infection Model, WEKA - J48 classifier and the Visualiser) were utilised. The Infection Model application which was implemented using the well-known Agent-Based modelling simulation platform (REPAST) Simphony was ported on the Africa Grid Science Gateway. The Infection Model (which studies the behaviour of infections with an annual outbreak) was utilised in order to implement several aspects of the MESSAGE methodology and to demonstrate how scientists may easily access an ABMS simulation application, worldwide. The first version of the Infection Model portlet presents the sequential version of the system which is used to execute jobs (sequentially) in a distributed environment. The second application (i.e. the WEKA - J48 classifier) on the other hand was utilised for two main reasons. First of all, it serves as a case study to be used to evaluate the methods and processes which was used in developing the Infection Model. Secondly, the WEKA - J48 classifier was chosen in order to demonstrate the process of workflow management service in Science Gateways. This demonstration is such that the Infection Model simulation output results will be used as the input file to be analysed by the WEKA - J48 portlet and this processes will be managed by the Science Gateway and any associated workflow management system.

Another aspect of this research is the parallel implementation of the Infection Model portlet which was ported to the Science Gateway. This second version of the Infection Model portlet allows users to run multiple experiments simultaneously. Furthermore, the parallelism that was performed at the DCI layer sees the use of VMs with multiple cores to execute these scientific applications. A similar level of parallelism was also performed for the WEKA - J48 portlet albeit at the DCI layer.

The MESSAGE methodology and the methods and approaches that were utilised for developing both scientific applications (Infection Model and WEKA - J48 classifier) in the Africa Grid Science Gateway were evaluated using case studies from two different EU projects. These different sets of use-cases were developed and implemented using two different Science Gateway Frameworks. These Science Gateway frameworks are the Catania

Science Gateway framework/FutureGateway APIs and the gUSE/WS-PGRADE Science Gateway frameworks. While the first set of case studies were developed using a similar Science Gateway framework which was used to develop the Infection Model portlet and the WEKA – J48 portlet, the second sets of case studies on the other-hand were developed using the gUSE/WS-PGRADE Science Gateway Framework. The decision to compare the approaches described in the MESSAGE methodology with case studies that were developed using different Science Gateway Frameworks was considered really crucial to the whole evaluation process as it helps to generalise the MESSAGE methodology.

The evaluation that was performed based on comparing the method that was used for developing different applications of the EU projects with the Infection Model and the WEKA J48 classifier shows that the majority of the use-cases in the EU projects require support for data repositories/data analysis capabilities. The MESSAGE methodology is thus revised to include this capability. In addition, the use of data repository in this way to store other research outputs such as (software, data, and results) encourages collaboration, validation and re-use of research outputs. This notion is referred to, in one of the EU projects, as Open Science. In essence, the use of Open Access Document Repository and other enabling technologies such as Digital Object Identifiers (DOI) and the Open Researcher and Contributor ID (ORCID) can aid with the easy accessibility and availability of this research outputs. The aforementioned technologies and how they can enable the easy access and retrieval of research outputs are demonstrated by using the Infection Model simulation outputs. Even though the concept of Open Science is beyond the scope of this research, the revised MESSAGE methodology (in Chapter 7) can provide a foundation for Open Science Research.

## *8.3 Research aim and objectives*

The aim of this research was to explore new approaches to developing scientific software applications in Science Gateways. To achieve this aim, the following research questions were addressed:

- (RQ1) What approach can be used to develop scientific software applications in Science Gateways?

  A thorough review of the literature shows that even though there are existing

Science Gateway frameworks for developing application specific Science Gateways, there are no methodologies in place to guide their actual development. The MESSAGE methodology was therefore proposed and implemented to cater for these needs.

- (RQ2) What Science Gateway framework and SDLC approach are appropriate for implementing the proposed methodology for developing scientific software applications in Science Gateways?

  A review of the literature also shows the different Science Gateway frameworks that can be used for the implementation of application specific Science Gateways. However, the CSGF was considered as an exemplar Science Gateway framework which caters for the different Science Gateway functionalities that were identified in Chapter 2. In addition, relevant aspects of the Waterfall software development approach was also utilised.

- (RQ3) Is the developed methodology effective?

  A Reflective Assessment was done in order to evaluate the MESSAGE methodology. By reflecting on the approaches utilised for the development of the different case studies utilised in this research as well as the different approaches adopted in two different EU projects (discussed in Chapter 7), shows that the MESSAGE methodology is effective in its use for portlet development. The generic nature of the MESSAGE methodology was also established.

In order to achieve the aim and subsequently address the above research questions, five objectives were identified. These objectives were met through the different Chapters of this thesis as follows:

1. Conduct a literature review on the state of the art of Science Gateways, Web Portals, Workflow Management Systems, DCIs, and all other technologies that lend themselves to the easy access and use of distributed resources.

   Chapter 2 presented the literature review which provides the state of the art of all the related technologies such as Science Gateways, Web portals, Workflow

Management systems and DCI resources. This Chapter provided the foundation on which the entire thesis was founded.

2. Develop a research methodology that will be utilised throughout this research as well as create a methodology for developing scientific software applications in Science Gateways.

   Chapter 3 discussed the philosophy and nature of the research and went on to justify the selection of the DSR methodology as the adopted research methodology for the entire research. Chapter 4, on the other hand, analysed the major functionalities that a Science Gateway should possess (based on the review of literature in Chapter 2). These functionalities are otherwise known as the generic framework of Science Gateway services. The MESSAGE methodology which could help (potentially) in facilitating one or more of these services for specific communities of practice was later proposed in Chapter 4.

3. Implement the methodology for developing scientific applications in Science Gateways in objective (2), using three different scientific software applications. The first scientific application is the case study that is utilised for the implementation of the proposed ideas and methods described in the MESSAGE methodology, while the second scientific application is the case study that is used for its evaluation.

   In Chapter 5, three different scientific software applications were used to implement the proposed MESSAGE methodology for developing scientific applications discussed in Chapter 3. These scientific applications include an Infection Model, WEKA J48 classifier and the Infection Model Visualiser. The sequential and parallel versions of these applications were ported on the Africa Grid Science Gateway and the process involved in their development was documented throughout Chapter 5 and 6, respectively.

4. Perform a thorough evaluation of both the proposed methodology in objective (2), and the development and implementation of the scientific applications in objective (3), by comparing with the approach that was used in developing real life use-cases of two Science Gateway projects.

   In Chapter 7, the first stage of evaluation that was carried out involved a comparison of the methods and approaches that were used to develop the Infection Model portlet and the WEKA – J48 portlet. The second stage of evaluation however compared the development processes of several use-cases from two EU projects with the proposed MESSAGE methodology for developing Science Gateways in Chapter 4, and subsequently the methododology used in developing both the Infection Model and the WEKA – J48 portlet.

5. Based on the evaluation in objectives (4), revise the methodology for developing scientific software applications in Science Gateways which was proposed in objective (2).

    The evaluation that was performed in objective (4) shows that several of the communities of practice (especially in the Sci-GaiA project) require extra support for data repository and data analysis capabilities. This capability was therefore added to the revised version of the MESSAGE methodology for developing scientific software applications in Science Gateways that was proposed in objective (2).

## *8.4 Research Contributions*

This Section presents the major contribution and three other contributions which were made and discussed throughout this thesis. This thesis generated the following significant contribution as follows:

### 8.4.1 A Methodology for Developing Scientific Applications in Science Gateways

The main contribution of this thesis is the creation of a MESSAGE methodology for developing scientific software applications in Science Gateways. The proposed MESSAGE methodology in Chapter 4 describes the processes involved in developing scientific software applications in Science Gateways, from the general description of the use case, portlet functionalities, specification of different requirements, and the selection of appropriate Science Gateway approach to the adoption and use of relevant aspects of software development methodology. The MESSAGE methodology comprises of two phases. The first phase deals with the actual implementation of portlets with the backends that support DCIs. This is performed by either customising an existing Science Gateway framework (which already comes with those backends) or develop them from scratch by defining their configurations. The second phase is the adoption and use of relevant software development methodology. As such, the MESSAGE methodology is neither a Science Gateway Framework nor a software development methodology. Nonetheless, it combines both approaches in its development processes.

While the MESSAGE methodology provides a general guide for developing scientific software applications across different Science Gateway frameworks, the revised MESSAGE methodology in Chapter 6 however, also facilitates the extra use of data repositories which is used to store different kinds of research outputs such as data, software and results (obtained as a consequence of utilising a Science Gateway to execute scientific applications) for easy accessibility and availability and for the purpose of future research collaborations, validation and reuse.

The MESSAGE methodology can aid developers in accelerating the development of their portlets and thus help save development time and effort. This can be achieved by customising an existing Science Gateway framework (which comprise the part that was demonstrated in this research). The methodology can also help in identifying the similarities between different scientific applications, thereby simplifying the development process. Lastly, the MESSAGE methodology can help to develop a portlet from scratch (which comprise of the parts that will be demonstrated in the future work). However, this approach can take considerable amount of time and effort to implement.

### 8.4.2 A Framework for the Major Science Gateway Services

This thesis also presents functionalities that a Science Gateway should possess (such as security management, job management, workflow management and data management) which are referred (in Chapter 4) as the generic framework of services provided by Science Gateways. These services which were summarised in the review of the literature can either be specific to a community of practice or generic to different communities of practice. Once a community has identified the necessary Science Gateway service or set of services as required by their scientific domain, the revised MESSAGE methodology in Chapter 6 can therefore be used to develop the application on the Science Gateway.

### 8.4.3 Parallel Execution of jobs in Distributed Environments (The Science Gateway Approach)

As revealed in the review of the literature, many communities of practice require effective and efficient means of executing scientific jobs in DCIs. In this thesis, a major Science Gateway capability was proposed and implemented in which communities of practice can execute multiple experiments simply by using the Science Gateway interface. More so, communities that make use of ABMS applications can perform multiple simulation runs at any given time. This capability was implemented by adapting the CSGF Grid and Cloud Engine functionalities to cater for these needs.

### 8.4.4 A Foundation for Open Science Research

This thesis began with an attempt to make scientific software applications easily accessible and available, worldwide. Much later in this research, there was also an opportunity to further investigate ways in which other research outputs (such as data, software and results) can be treated in a similar manner to the scientific software applications (i.e. make them easily accessible and available). The principles of Open Science were demonstrated (using the Infection Model simulation outputs/results) by utilising enabling technologies such as Open Access Document Repository (OADR), Digital Object identifiers (DOI) and the Open Researcher and Contributor ID (ORCID). This research (to a lesser extent) therefore contributes to Open Science by laying a foundation for which future Open Science research may be conducted. It also compliments the work done by other Open Science initiatives such as the Center for Open Science (COS) that aims to promote Open Science research by making use of a variety of software tools, workflows and data storage solutions to help researchers manage, archive, discover and share research more openly. Another initiative that aims to promote Open Science is the Open Knowledge Foundation (OKF) which is a non-profit organisation that advocates open knowledge, open data, transparency and civil participation.

## *8.5 Research Limitations and Reflections*

One limitation of this research is that the MESSAGE methodology was only applied in three case studies. To fully utilise majority of the functionalities of Science Gateways (depicted as services in this thesis), it is desirable to apply the MESSAGE methodology to more case studies and test all the major Science Gateways services that were captured in Chapter 4 such as the workflow management system service.

Another limitation is the use of a single Science Gateway framework (Catania Science Gateway Framework/FutureGateway APIs) for the development and implementation of the three scientific software applications that were utilised throughout this research. It is also desirable to utilise other prominent Science Gateway Frameworks discussed in Chapter 2 (such as the gUSE/WS-PGRADE) to implement some of the Science Gateway services discussed in Chapter 4.

On a side note, this research started with the ambition of creating an avenue for comparing Science Gateway methods and approaches by using some of the Science Gateway Frameworks (discussed in Chapter 2) to implement several of the Science Gateway services identified in Chapter 4 by adopting the MESSAGE methodology. However, given the time specified to finish a PhD study it is now evident that (at the early stages of this research) the researcher may have underestimated the time required to get this done. In the following section however, future work will now consider the aforementioned points.

## *8.6 Future Work*

Firstly, the generic nature of the revised MESSAGE methodology (which was discussed in Chapter 7) has been established. Therefore, other Science Gateway frameworks are able to utilise the MESSAGE methodology for developing applications and services for their associated communities. Future work will attempt to utilise the proposed MESSAGE methodology for developing both scientific applications (i.e. the Infection Model and WEKA) using a different Science Gateway framework other than the Catania Science Gateway Framework/FutureGateway APIs. To do this therefore, the gUSE/WS-PGRADE or any other Science Gateway framework discussed in Chapter 2 will be adopted.

Additionally, beyond just simple job submissions and service calls, applications solving complex problems like scientific simulations require the creation and execution of scientific workflows. Among the different Science Gateway services that were discussed in Chapter 4, the services offered by Workflow Management System (in particular) are yet to be explored. In this research, WEKA J48 classifier was adopted for two main reasons. The first is to serve as a case study that would be used to evaluate the processes and methods involved in developing the Infection Model portlet, and secondly to demonstrate the principles of workflow management using Science Gateways. Future work will therefore investigate the effects of using parameter sweep involving the use of both scientific applications.

A Parameter Study (PS) is a methodical process of changing various model parameters which are then automatically run on one or several simulation models for each parameter combination. To demonstrate this, the Infection Model portlet will be executed (by specifying and executing different simulation runs on the Science Gateway) and the generated results will be used as an input file to be further analysed by the WEKA – J48 portlet on the Science Gateway. The Kepler Workflow Management System (which has been described in great detail in Chapter 2, and serves as a workflow orchestration, composition and construction and focuses on data modelling and analysis) will be utilised along with the Science Gateway interface to execute and manage such complex jobs.

# References

Afgan, E., Baker, D., Coraor, N., Chapman, B., Nekrutenko, A. and Taylor, J., 2010, December. Galaxy CloudMan: delivering cloud compute clusters. In *BMC bioinformatics*, Vol. 11, No. 12, p. S4. BioMed Central.

Akram, A., Chohan, D., Meredith, D. and Allan, R., 2007. CCLRC portal infrastructure to support research facilities. *Concurrency and Computation: Practice and Experience*, *19*(6), pp.751-766.

Alameda, J., Christie, M., Fox, G., Futrelle, J., Gannon, D., Hategan, M., Kandaswamy, G., von Laszewski, G., Nacar, M.A., Pierce, M. and Roberts, E., 2007. The Open Grid Computing Environments collaboration: portlets and services for science gateways. *Concurrency and Computation: Practice and Experience*, *19*(6), pp.921-942.

Aloisio, G., Cafaro, M., Carteni, G., Epicoco, I., Fiore, S., Lezzi, D., Mirto, M. and Mocavero, S., 2007. The grid resource broker portal. *Concurrency and Computation: Practice and Experience*, *19*(12), pp.1663-1670.

Ardizzone, V., Barbera, R., Calanducci, A., Fargetta, M., Ingrà, E., Porro, I., La Rocca, G., Monforte, S., Ricceri, R., Rotondo, R. and Scardaci, D., 2012. The DECIDE science gateway. *Journal of Grid Computing*, *10*(4), pp.689-707.

Ardizzone, V., Bruno, R., Calanducci, A., Fargetta, M., Ingrà, E., La Rocca, G., Monforte, S., Pistagna, F., Ricceri, R., Rotondo, R. and Scardaci, D., 2014. The GISELA science gateway.

Atzei, N., Bartoletti, M. and Cimoli, T., 2017. A survey of attacks on Ethereum smart contracts (SoK). In *International Conference on Principles of Security and Trust*, pp. 164-186. Springer, Berlin, Heidelberg.

Baker, M., 2016. 1,500 scientists lift the lid on reproducibility. *Nature News*, *533*(7604), p.452.

Baker, M., Buyya, R. and Laforenza, D., 2002. 'Grids and Grid technologies for wide-area distributed computing', *Software: Practice and Experience*, 32(15), pp. 1437-1466.

Balasko, A., Farkas, Z. and Kacsuk, P., 2013. Building science gateways by utilizing the generic WS-PGRADE/gUSE workflow system. *Computer Science*, *14*(2), p.307.

Balaz, A., Cauhé, E., Chen, H., Jovanovic, P., Kacsuk, P., Loureiro-Ferreira, N., Lovas, R., Olabarriaga, S., Shahand, S., Sudholt, W., Vudragovic, D., 2013. EGI-InSPIRE Science Gateway Primer. Available at: https://documents.egi.eu/document/1463 (Accessed: 10/08/2016)

Barbera, R., Falzone, A., Ardizzone, V. and Scardaci, D., 2007, June. The GENIUS Grid portal: its architecture, improvements of features, and new implementations about authentication and authorization. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2007. WETICE 2007. 16th IEEE International Workshops on* pp. 279-283. IEEE.

Barbera, R., Bruno, R., Calanducci, A., Messina, A., Pappalardo, M. and Passaro, G., 2013, April. The earthserver project: Exploiting identity federations, science gateways and social and mobile clients for big earth data analysis. In *EGU General Assembly Conference Abstracts*, Vol. 15.

Barbera, R., Fargetta, M. and Rotondo, R., 2011, March. A simplified access to grid resources by science gateways. In *Proceedings of the International Symposium on Grids and Clouds, Taipei (2011), Proceedings of Science (ISGC 2011 & OGF 31)* p. 23.

Barbera, R., Andronico, G., Donvito, G., Falzone, A., Keijser, J.J., Rocca, G.L., Milanesi, L., Maggi, G.P. and Vicario, S., 2011. 'A grid portal with robot certificates for bioinformatics phylogenetic analyses', *Concurrency and Computation: Practice and Experience*, 23(3), pp. 246-255.

Bhargava, N., Sharma, G., Bhargava, R. and Mathuria, M., 2013. 'Decision tree analysis on j48 algorithm for data mining', *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6).

Bilandzic, M. and Venable, J., 2011. 'Towards participatory action design research: adapting action research and design science research methods for urban informatics', *The Journal of Community Informatics*, 7(3).

Binns, J., DiCarlo, J., Insley, J.A., Leggett, T., Lueninghoener, C., Navarro, J. and Papka, M.E., 2007. 'Enabling community access to TeraGrid visualization resources', *Concurrency and Computation: Practice and Experience*, 19(6), pp. 783-794.

Boccara, N. and Cheong, K., 1992. Automata network SIR models for the spread of infectious diseases in populations of moving individuals. *Journal of Physics A: Mathematical and General*, *25*(9), p.2447.

Boehm, B.W., 1988. 'A spiral model of software development and enhancement', *Computer,* 21(5), pp. 61-72.

Boehm, B. and Turner, R., 2003. *Balancing Agility and Discipline: A Guide for the Perplexed, Portable Documents.* Addison-Wesley Professional.

Bogdanski, M., Kosiedowski, M., Mazurek, C., Rabiega, M. and Wolniewicz, M., 2007. 'Flexibility and user-friendliness of Grid portals: the PROGRESS approach', *Concurrency and Computation: Practice and Experience*, 19(6), pp. 827-838.

Bote-Lorenzo, M.L., Dimitriadis, Y.A. and Gómez-Sánchez, E., 2004. Grid characteristics and uses: a grid definition. In *Grid Computing*, pp. 291-298. Springer, Berlin, Heidelberg.

Bresfelean, V.P., 2007. 'Analysis and predictions on students' behavior using decision trees in Weka environment', *Proceedings of the ITI.* , 25-28.

Bruno, R., Allegri, G., Andronico, G., Barbera, R., Bitelli, F., Budano, A., Calanducci, A., Celli, F., Costantini, E. and Fargetta, M., 2013b. 'The agINFRA Science Gateway for Agricultural Sciences', pos proceedings of science, pp. 20 pp.

Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I., 2009. 'Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility', *Future Generation Computer Systems,* 25(6), pp. 599-616.

Cannataro, M., Guzzi, P.H. and Sarica, A., 2013. 'Data mining and life sciences applications on the grid', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* 3(3), pp. 216-238.

Carlsson, S.A., 2006. 'Towards an information systems design research framework: A critical realist perspective', *Proceedings of the First International Conference on Design Science Research in Information Systems and Technology, Claremont, CA.* , 192-212.

Carrubba, C., Fargetta, M., Rotondo, R. and Barbera, R., 2013, March. "Social" Science Gateways to e-Infrastructures. In *The International Symposium on Grids and Clouds (ISGC)* Vol. 2013.

Casarino, C., Russo, G., Candiano, G., La Rocca, G., Barbera, R., Borasi, G., Guatelli, S., Messa, C., Passaro, G. and Gilardi, M.C., 2015. 'A GEANT4 web-based application to support Intra-Operative Electron Radiotherapy using the European grid infrastructure', *Concurrency and Computation: Practice and Experience*, 27(2), pp. 458-472.

Chrabakh, W. and Wolski, R., 2007. 'The GridSAT portal: a Grid Web-based portal for solving satisfiability problems using the national cyberinfrastructure', *Concurrency and Computation: Practice and Experience,* 19(6), pp. 795-808.

Christie, M. and Marru, S., 2007. 'The LEAD Portal: a TeraGrid gateway and application service architecture', *Concurrency and Computation: Practice and Experience,* 19(6), pp. 767-781.

Cobb, J.W., Geist, A., Kohl, J.A., Miller, S.D., Peterson, P.F., Pike, G.G., Reuter, M.A., Swain, T., Vazhkudai, S.S. and Vijayakumar, N.N., 2007. 'The Neutron Science TeraGrid Gateway: a TeraGrid science gateway to support the Spallation Neutron Source', *Concurrency and Computation: Practice and Experience,* 19(6), pp. 809-826.

Collier, N., Ozik, J. and Macal, C.M., 2015. 'Large-scale agent-based modeling with repast hpc: A case study in parallelizing an agent-based model', *European Conference on Parallel Processing.* Springer, 454-465.

Coullon, H. and Limet, S., 2016. The SIPSim implicit parallelism model and the SkelGIS library. *Concurrency and Computation: Practice and Experience*, *28*(7), pp.2120-2144.

Coveney, P.V., Saksena, R.S., Zasada, S.J., McKeown, M. and Pickles, S., 2007. The application hosting environment: lightweight middleware for grid-based computational science. *Computer Physics Communications*, *176*(6), pp.406-418.

Davis, J.P., Eisenhardt, K.M. and Bingham, C.B., 2007. Developing theory through simulation methods. *Academy of Management Review*, *32*(2), pp.480-499.

Deelman, E., Gannon, D., Shields, M. and Taylor, I., 2009. 'Workflows and e-Science: An overview of workflow system features and capabilities', *Future Generation Computer Systems*, 25(5), pp. 528-540.

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P.J., Mayani, R., Chen, W., da Silva, R.F. and Livny, M., 2015. 'Pegasus, a workflow management system for science automation', *Future Generation Computer Systems*, 46, pp. 17-35.

Demeyer, S., 2011, September. Research methods in computer science. In *ICSM*, p. 600.

Drazin, S. and Montag, M., 2012. Decision tree analysis using weka. *Machine Learning-Project II, University of Miami*, pp.1-3.

Elia, D., Nuzzo, A., Nassisi, P., Fiore, S., Blanquer, I., Brasileiro, F.V., Rufino, I.A., Seijmonsbergen, A.C., Anders, N.S., de Oliveira Galvao, C. and de BL Cunha, J.E., 2017. A Science gateway for biodiversity and climate change research. *PeerJ PrePrints*, *5*, p.e2834.

Fabiyi, A.O., Taylor, S.J., Anagnostou, A., Torrisi, M. and Barbera, R., 2016, September. Investigating a Science Gateway for an Agent-Based Simulation Application Using REPAST. In *Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on* pp. 29-36. IEEE.

Fabra, J., Hernández, S., Otero, E., Vidal, J.C., Lama, M. and Álvarez, P., 2015. 'Integration of grid, cluster and cloud resources to semantically annotate a large-sized repository of learning objects', *Concurrency and Computation: Practice and Experience*, 27(17), pp. 4603-4629.

Fahringer, T., Prodan, R., Duan, R., Hofer, J., Nadeem, F., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.L. and Villazon, A., 2007. Askalon: A development and grid computing environment for scientific workflows. In *Workflows for e-Science*, pp. 450-471. Springer, London.

Fallman, D., 2003, April. Design-oriented human-computer interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 225-232. ACM.

Farbey, B., Land, F. and Targett, D., 1999. 'Moving IS evaluation forward: learning themes and research issues', *The Journal of Strategic Information Systems*, 8(2), pp. 189-207.

Foster, I. and Kesselman, C., 2010. The History of the grid. *computing*, *20*(21), p.22.

Foster, I., Kesselman, C. and Tuecke, S., 2001. 'The anatomy of the grid: enabling scalable virtual organizations', *International Journal of High Performance Computing Applications, 15(3): 200-222.*

Foster, I., Zhao, Y., Raicu, I. and Lu, S., 2008, November. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pp. 1-10. Ieee.

Fox, G.C., Gannon, D. and Thomas, M., 2002. A summary of grid computing environments. *Concurrency and Computation*, *14*(13/15), pp.1035-1044.

Garner, S.R., 1995, April. Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand computer science research students conference*, pp. 57-64.

Gesing, S., Hemert, J.V., Kacsuk, P. and Kohlbacher, O., 2011. Special Issue: Portals for life sciences—Providing intuitive access to bioinformatic tools. *Concurrency and Computation: Practice and Experience*, *23*(3), pp.223-234.

Gesing, S., Krüger, J., Grunzke, R., Herres-Pawlis, S. and Hoffmann, A., 2015, June. Challenges and modifications for creating a mosgrid science gateway for us and european infrastructures. In *Science Gateways (IWSG), 2015 7th International Workshop on* pp. 73-79. IEEE.

Gesing, S., Nabrzyski, J. and Jha, S., 2014, June. Gateways to high-perfomance and distributed computing resources for global health challenges. In *Humanitarian Technology Conference-(IHTC), 2014 IEEE Canada International*, pp. 1-5. IEEE.

Goecks, J., Nekrutenko, A. and Taylor, J., 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology*, *11*(8), p.R86.

Görlach, K., Sonntag, M., Karastoyanova, D., Leymann, F. and Reiter, M., 2011. 'Conventional workflow technology for scientific simulation', in *Guide to e-Science.* Springer, pp. 323-352.

Gregor, S., 2002. 'Design theory in information systems', *Australasian Journal of Information Systems*, 10(1).

Gubała, T. and Bubak, M., 2005, September. GridSpace–semantic programming environment for the grid. In *International Conference on Parallel Processing and Applied Mathematics*, pp. 172-179. Springer, Berlin, Heidelberg.

Gupta, P. and Govil, M.C., 2010. Spring Web MVC Framework for rapid open source J2EE application development: a case study. *Interface, 2*(6), pp.1684-1689.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H., 2009. 'The WEKA data mining software: an update', *ACM SIGKDD explorations newsletter,* 11(1), pp. 10-18.

Hethcote, H.W., 1976. Qualitative analyses of communicable disease models. *Mathematical Biosciences*, *28*(3-4), pp.335-356.

Hevner, A. and Chatterjee, S., 2010. *Design research in information systems: theory and practice*, Vol. 22. Springer Science & Business Media.

Hirai, Y., 2017, April. Defining the ethereum virtual machine for interactive theorem provers. In *International Conference on Financial Cryptography and Data Security*, pp. 520-535. Springer, Cham.

Hollingsworth, D. and Hampshire, U., 1995. 'Workflow management coalition: The workflow reference model', *Document Number TC00-1003*, 1.

Holmes, G., Donkin, A. and Witten, I.H., 1994, November. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on* pp. 357-361. IEEE.

Howell, K.E., 2012. *An introduction to the philosophy of methodology*. Sage.

Jablonski, S. and Bussler, C., 1996. *Workflow management: modeling concepts, architecture and implementation* (Vol. 392). London: International Thomson Computer Press.

Jacob, B., Brown, M., Fukui, K. and Trivedi, N., 2005. 'Introduction to grid computing', *IBM redbooks.*

Jha, S., Cole, M., Katz, D.S., Parashar, M., Rana, O. and Weissman, J., 2013. 'Distributed computing practice for large-scale science and engineering applications', *Concurrency and Computation: Practice and Experience*, 25(11), pp. 1559-1585.

Johnson, C., 2006. 'What is research in computing science?' Department of Computer Science, University of Glasgow (Glasgow Interactive Systems Group (GIST)), pp. Available online http://www.cnitblog.com/cyberfan/articles/1662.html (Accessed: 26/08/2017).

Kacsuk, P., Farkas, Z., Kozlovszky, M., Hermann, G., Balasko, A., Karoczkai, K. and Marton, I., 2012. WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities. *Journal of Grid Computing*, *10*(4), pp.601-630.

Kacsuk, P., 2011. 'P-GRADE portal family for grid infrastructures', *Concurrency and Computation: Practice and Experience*, 23(3), pp. 235-245.

Kee, K.F. and Schrock, A.R., 2018. Best social and organizational practices of successful science gateways and cyberinfrastructure projects. *Future Generation Computer Systems.*

Kertesz, A., Otvos, F. and Kacsuk, P., 2014. 'A case study for biochemical application porting in european grids and clouds', *Concurrency and Computation: Practice and Experience,* 26(10), pp. 1730-1743.

Kim, S.H., Kang, D.K., Kim, W.J., Chen, M. and Youn, C.H., 2017. A science gateway cloud with cost-adaptive VM management for computational science and applications. *IEEE Systems Journal*, *11*(1), pp.173-185.

Kiss, T., Kacsuk, P., Kovacs, J., Rakoczi, B., Hajnal, A., Farkas, A., Gesmier, G. and Terstyanszky, G., 2017. MiCADO—Microservice-based Cloud Application-level Dynamic Orchestrator. *Future Generation Computer Systems.*

Kiss, T., Kacsuk, P., Takács, É., Szabó, Á., Tihanyi, P. and Taylor, S.J., 2014. Commercial use of ws-pgrade/guse. In *Science Gateways for Distributed Computing Infrastructures*, pp. 271-286. Springer, Cham.

Kocot, J., Szepieniec, T., Wójcik, P., Trzeciak, M., Golik, M., Grabarczyk, T., Siejkowski, H. and Sterzel, M., 2014. A framework for domain-specific science gateways. In *eScience on Distributed Computing Infrastructure*, pp. 130-146. Springer, Cham.

Kuhn, T.S., 1996. 'The Structure of Scientific Revolution, Chicago.

Laure, E. and Edlund, A., 2012. 'The e-infrastructure ecosystem: Providing local support to global science', *Large-Scale Computing Techniques for Complex System Simulations,* 80, pp. 19.

Lawrence, K.A., Zentner, M., Wilkins-Diehr, N., Wernert, J.A., Pierce, M., Marru, S. and Michael, S., 2015a. 'Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community', *Concurrency and Computation: Practice and Experience,* 27(16), pp. 4252-4268.

Limet, S., Smari, W.W. and Spalazzi, L., 2015. 'High-performance computing: to boldly go where no human has gone before', *Concurrency and Computation: Practice and Experience*, 27(13), pp. 3145-3165.

Macal, C.M. and North, M.J., 2009. 'Agent-based modeling and simulation', *Winter simulation conference.* Winter simulation conference, 86-98.

Macal, C.M. and North, M.J., 2005, December. Tutorial on agent-based modeling and simulation. In *Simulation conference, 2005 proceedings of the winter*, pp. 14-pp. IEEE.

March, S.T. and Smith, G.F., 1995. 'Design and natural science research on information technology', *Decision Support Systems*, 15(4), pp. 251-266.

Matsunaga, A.M., Tsugawa, M.O., Adabala, S., Figueiredo, R.J., Lam, H. and Fortes, J.A.B., 2007. 'Science gateways made easy: the In-VIGO approach', *Concurrency and Computation: Practice and Experience*, 19(6), pp. 905-919.

Mell, P. and Grance, T., 2011. 'The NIST definition of cloud computing'.

Merzky, A., Weidner, O. and Jha, S., 2015. SAGA: A standardized access layer to heterogeneous distributed computing infrastructure. *SoftwareX*, *1*, pp.3-8.

Mingers, J., 2001. 'Combining IS research methods: towards a pluralist methodology', *Information systems research*, 12(3), pp. 240-259.

Myers, M.D., 1997. 'Qualitative Research in Information Systems', *MIS Quarterly. Available: http://www.qual.auckland.ac.nz/ (Accessed: 25/04/2017).*

Novotny, J., 2003. 'The Grid Portal Development Kit', in *Grid Computing.* John Wiley & Sons, Ltd, pp. 657-673.

Novotny, J., Russell, M. and Wehrens, O., 2004. 'GridSphere: a portal framework for building collaborations', *Concurrency and Computation: Practice and Experience*, 16(5), pp. 503-513.

Nuryatno, M.A., 2003. The Call for Paradigm Shift in Qualitative Research from Positivism and Interpretive to Critical Theory. Hermēneia. *Jurnal Kajian Islam Interdisipliner*, *1*(2), pp.24-50.

Ogasawara, E., Dias, J., Silva, V., Chirigati, F., Oliveira, D., Porto, F., Valduriez, P. and Mattoso, M., 2013. 'Chiron: a parallel engine for algebraic scientific workflows', *Concurrency and Computation: Practice and Experience,* 25(16), pp. 2327-2341.

Ogawa, H., Itoh, S., Sonoda, T. and Sekiguchi, S., 2007. 'GridASP: an ASP framework for Grid utility computing', *Concurrency and Computation: Practice and Experience,* 19(6), pp. 885-891.

Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R., Wipat, A. and Li, P., 2004. 'Taverna: a tool for the composition and enactment of bioinformatics workflows', *Bioinformatics (Oxford, England),* 20(17), pp. 3045-3054.

Oleksiak, A., Tullo, A., Graham, P., Kuczyński, T., Nabrzyski, J., Szejnfeld, D. and Sloan, T., 2007. HPC‐Europa single point of access as a framework for building science gateways. *Concurrency and Computation: Practice and Experience*, *19*(6), pp.851-866.

Open Science Collaboration, 2015. Estimating the reproducibility of psychological science. *Science*, *349*(6251), p.aac4716.

Open Science Collaboration, 2012. An open, large-scale, collaborative effort to estimate the reproducibility of psychological science. *Perspectives on Psychological Science*, *7*(6), pp.657-660.

Orlikowski, W.J. and Baroudi, J.J., 1991. 'Studying information technology in organizations: Research approaches and assumptions', *Information systems research,* 2(1), pp. 1-28.

Pierantoni, G., Frost, G., Gesing, S., Olabarriaga, S., Jaghoori, M., Terstyanski, G. and Arshad, J., 2017, June. A model for information and action flows connecting science gateways to distributed computing infrastructures. In *CEUR Workshop Proceedings*, Vol. 1871. CEUR Workshop Proceedings.

Pordes, R., Petravick, D., Kramer, B., Olson, D., Livny, M., Roy, A., Avery, P., Blackburn, K., Wenaus, T., Würthwein, F. and Foster, I., 2007. The open science grid. In *Journal of Physics: Conference Series*, Vol. 78, No. 1, p. 012057. IOP Publishing.

Purao, S., 2002. Design research in the technology of information systems: Truth or dare. *GSU Department of CIS Working Paper*, pp.45-77.

Reluga, T.C. and Medlock, J., 2007. Resistance mechanisms matter in SIR models. *Mathematical Biosciences and Engineering*, *4*(3), p.553.

Robey, D., 1996. 'Research commentary: diversity in information systems research: threat, promise, and responsibility', *Information systems research,* 7(4), pp. 400-408.

Runeson, P. and Höst, M., 2009. 'Guidelines for conducting and reporting case study research in software engineering', *Empirical software engineering*, 14(2), pp. 131-164.

Ruparelia, N.B., 2010. 'Software development lifecycle models', *ACM SIGSOFT Software Engineering Notes,* 35(3), pp. 8-13.

Russell, M., Dziubecki, P., Grabowski, P., Krysiński, M., Kuczyński, T., Szjenfeld, D., Tarnawczyk, D., Wolniewicz, G. and Nabrzyski, J., 2007, September. The vine toolkit: A java framework for developing grid applications. In *International Conference on Parallel Processing and Applied Mathematics,* pp. 331-340. Springer, Berlin, Heidelberg.

Sadashiv, N. and Kumar, S.D., 2011, August. Cluster, grid and cloud computing: A detailed comparison. In *Computer Science & Education (ICCSE), 2011 6th International Conference on* pp. 477-482. IEEE.

Sarang, P., 2009. *Practical liferay: Java-based portal applications development*. Apress.

Sciacca, E., Bandieramonte, M., Becciani, U., Costa, A., Krokos, M., Massimino, P., Petta, C., Pistagna, C., Riggi, S. and Vitello, F., 2013, February. Visivo workflow-oriented science gateway for astrophysical visualization. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on* pp. 164-171. IEEE.

Sciacca, E., Vitello, F., Becciani, U., Costa, A., Hajnal, A., Kacsuk, P., Farkas, Z., Marton, I., Molinari, S., Di Giorgio, A.M. and Schisano, E., 2017. VIALACTEA science gateway for Milky Way analysis. *Future Generation Computer Systems.*

Shahand, S., Benabdelkader, A., Jaghoori, M.M., Mourabit, M.A., Huguet, J., Caan, M.W.A., van Kampen, A.H.C. and Olabarriaga, S.D., 2015. 'A data-centric neuroscience gateway: design, implementation, and experiences', *Concurrency and Computation: Practice and Experience,* 27(2), pp. 489-506.

Shields, P.M. and Rangarajan, N., 2013. *A playbook for research methods: Integrating conceptual frameworks and project management.* New Forums Press.

Simon, H.A., 1996. *The sciences of the artificial.* MIT press.

Smith, C.W. and Abeysinghe, E., 2017, July. The PHASTA Science Gateway: Web-based Execution of Adaptive Computational Fluid Dynamics Simulations. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact,* p. 70. ACM.

Soddemann, T., 2007. 'Science gateways to DEISA: user requirements, technologies, and the material sciences and plasma physics gateway', *Concurrency and Computation: Practice and Experience,* 19(6), pp. 839-850.

Stolterman, E., 2008. 'The nature of design practice and implications for interaction design research', *International Journal of Design,* 2(1).

Taylor, I., Shields, M., Wang, I. and Harrison, A., 2007. The triana workflow environment: Architecture and applications. In *Workflows for e-Science,* pp. 320-339. Springer, London.

Taylor, S.J., Fabiyi, A., Anagnostou, A., Barbera, R., Torrisi, M., Ricceri, R. and Becker, B., 2016, September. Demonstrating open science for modeling & simulation research. In *Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on* pp. 191-192. IEEE.

Thomas, M., 2007. 'Special Issue: Workshop on Grid Computing Portals (GCE 2005)', *Concurrency and Computation: Practice and Experience*, 19(12), pp. 1563-1570.

Thomas, M., Dahan, M., Mueller, K., Mock, S., Mills, C. and Regno, R., 2002. 'Application portals: practice and experience', *Concurrency and Computation: Practice and Experience*, 14(13-15), pp. 1427-1443.

Vaishnavi, V. and Kuechler, W., 2015. *Design Research in Information System*. Available at: http://desrist.org/design-research-in-information-systems/ (*Accessed*: 23/03/2016).

Vaishnavi, V. and Kuechler, W., 2009. 'Design Science Research in Information Systems', *DESRIST.org. Available at: http://desrist.org/desrist (Accessed: 06/07/2017).*

Von Alan, R.H., March, S.T., Park, J. and Ram, S., 2004. 'Design science in information systems research', *MIS quarterly*, 28(1), pp. 75-105.

Von Laszewski, G., Foster, I., Gawor, J. and Lane, P., 2001. A Java commodity grid kit. *Concurrency and Computation: practice and experience*, *13*(8-9), pp.645-662.

Wang, J. and Altintas, I., 2012. 'Early cloud experiences with the kepler scientific workflow system', *Procedia Computer Science*, 9, pp. 1630-1634.

Welch, V., Barlow, J., Basney, J., Marcusiu, D. and Wilkins-Diehr, N., 2007. 'A AAAA model to support science gateways with community accounts', *Concurrency and Computation: Practice and Experience*, 19(6), pp. 893-904.

Wilkins-Diehr, N., 2007. Special issue: science gateways—common community interfaces to grid resources. *Concurrency and Computation: Practice and Experience*, *19*(6), pp.743-749.

Yang, X., Akram, A. and Allan, R.J., 2007. 'Developing portals/portlets using Enterprise JavaBeans for Grid users', *Concurrency and Computation: Practice and Experience*, 19(12), pp. 1633-1641.

Yin, R.K., 2009. *Case study research: design and methods.* 4th edn. London: SAGE.

Youn, C., Baru, C., Bhatia, K., Chandra, S., Lin, K., Memon, A., Memon, G. and Seber, D., 2007. 'GEONGrid portal: design and implementations', *Concurrency and Computation: Practice and Experience,* 19(12), pp. 1597-1607.

Youn, C., Lu, J., Elgamal, A. and Baru, C., 2014. 'Development of web-based science portal for large-scale computing collaboration in earthquake engineering', *Concurrency and Computation: Practice and Experience,* 26(18), pp. 2907-2916.

Zhang, Q., Cheng, L. and Boutaba, R., 2010. 'Cloud computing: state-of-the-art and research challenges', *Journal of internet services and applications,* 1(1), pp. 7-18.

Zhang, C., Kelley, I. and Allen, G., 2007. 'Grid portal solutions: a comparison of GridPortlets and OGCE', *Concurrency and Computation: Practice and Experience,* 19(12), pp. 1739-1748.

Zhao, L., Song, C., Kalyanam, R., Biehl, L., Campbell, R., Delgass, L., Kearney, D., Wan, W., Shin, J., Kim, I.L. and Ellis, C., 2017. GABBs-Reusable Geospatial Data Analysis Building Blocks for Science Gateways.

Zhao, Y., Hategan, M., Clifford, B., Foster, I., Von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T. and Wilde, M., 2007, July. Swift: Fast, reliable, loosely coupled parallel computation. In *Services, 2007 IEEE Congress on* pp. 199-206. IEEE.

# Appendix A

## A1 Software Development Methodologies

**Rapid Application Development**

Rapid Application Development (RAD) is a methodology that uses prototyping as a mechanism for iterative development. In order to cope with change, coupled with the demand for faster but cheaper software development, RAD was introduced. This is an improvement over the more traditional methodologies such as Waterfall and Spiral models. RAD involves the rapid development of a version of the system for the purpose of evaluating at the early stages of the software process. This evaluation is usually done against the defined user requirements to allow further refinements if necessary. RAD is often used in a broader, more generic sense that encompasses a variety of techniques (such as Agile, Scrum, Extreme Programming, Joint Application Development, etc.) aimed at speeding software development. The major advantages that this model could offer include: The process of identifying weaknesses and strengths in a software by giving potential users early opportunity to evaluate and provide feedback, and the use of the system prototype to carry out experiments to measure the feasibility of a proposed design, etc.

**Spiral Model**

In spiral model, the software process is represented as a spiral with each loop demonstrating a phase of the software development process. The spiral model attempts to address two main difficulties of the waterfall model. Firstly, the two design stages are deemed unnecessary in some cases and, secondly, in order to account for software reusability or the identification of issues at an early stage, the top-down approach needs to be more flexible. The model often begins with a cycle in the form of several iterations that spiral out from small beginnings. Each cycle within the spiral often goes through four different stages as shown in Figure A-1 below. These stages include: Determining objectives, identifying and resolving risks, Development and testing, and planning the next iteration.

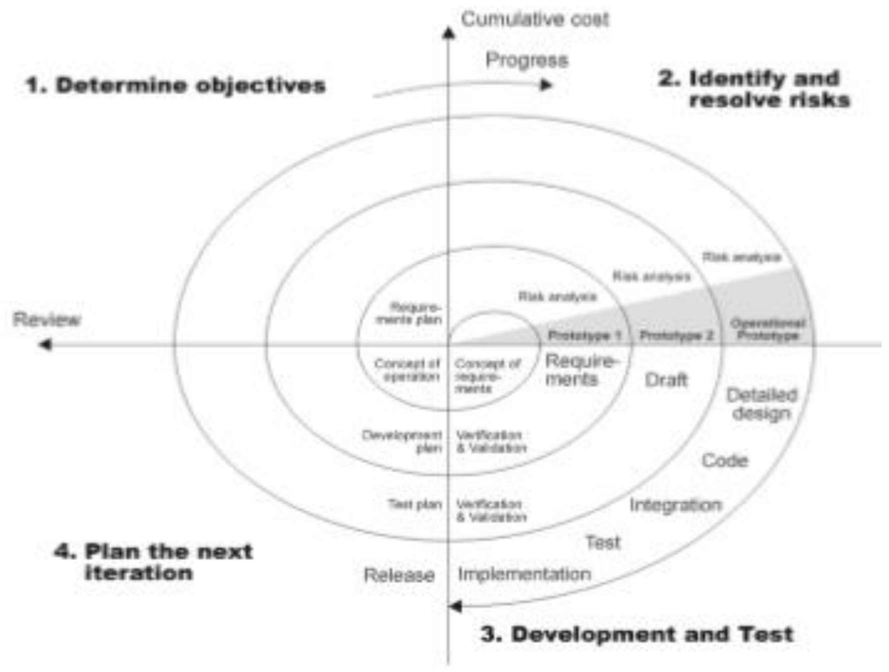


Figure A-1 Spiral life-cycle (Boehm, 1988)

**Waterfall Model**

The waterfall model provides a platform for defining and analysing software requirements prior to any design or development activities. It is used as the basis for most software acquisition standards both in government and in the industry sector. (Boehm and Turner, 2003). The waterfall model is an improvement over an earlier model, known as the stage-wise model. These improvements include the need to provide feedback loops between different stages as well as adapting these feedbacks to successive stages, and also an initial incorporation of prototyping in the software lifecycle. Basically, for waterfall model, cascading occurs from one stage to another, right from when the system requirements have been defined to the operation and maintenance stage. These stages are seen in Figure A-2 below.

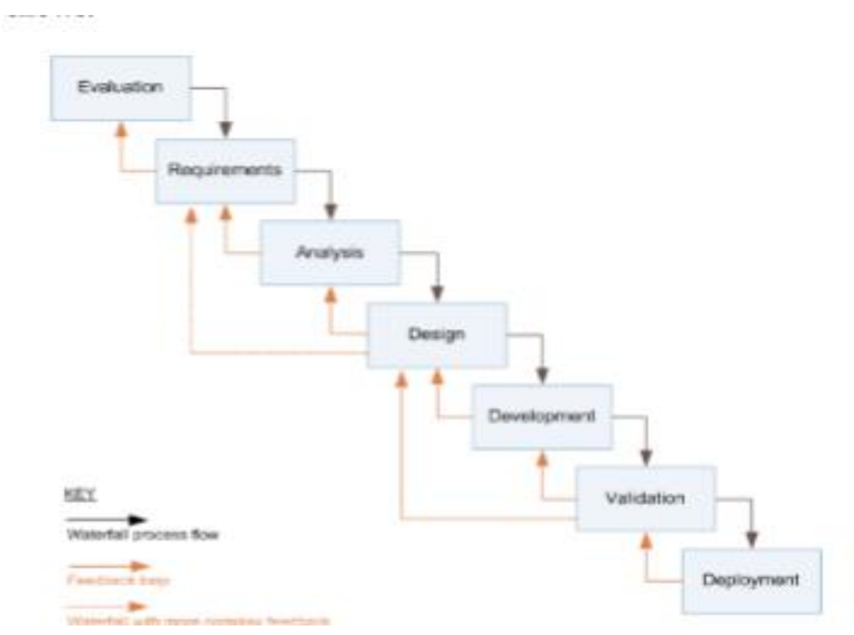


Figure A-2 Waterfall Model with Royce’s iterative feedback by (Ruparelia, 2010)

According to (Ruparelia, 2010), one aspect of waterfall model is its requirement for documentation. This documentation acts as a prerequisite for the commencement of the different stages of the waterfall model which includes: requirement documentation during the requirements stage, preliminary design specification during the preliminary design stage, interface design specification during the design stage, final design specification that is actively revised and updated for each cycle of the design phase, test plan during the design stage, and operational manual or instruction during the deployment stage. Even though these documents give a detailed understanding of what is required at every stage of the cycle, nevertheless, the fully elaborated documents implies the need for the completion of one stage of the cycle before moving on to the next one. Furthermore, the lack of flexibility in adapting to users changing requirements seems to be one drawback that is witnessed in this approach. As a consequence, it is recommended for projects whose requirements are very well defined or unlikely to change during the entire development process.

## A2 Pilot_script.sh

The pilot script for the ABMS Infection Model portlet has two main functions. First, it performs a simulation using the parameters that would be specified or passed in by the user, using the user interface module described above. Secondly, it creates an archive containing the output file for the executed ABMS jobs and collects all generated output files into a single tar.gz file known as simulation_output.txt. This is shown below as follows:

```
+#!/bin/sh
                  +#
                  +# myRepast-infection - portlet pilot script
                  +#
                  +# The following script does:
                  +#   o Perform a simulation using the provided parameters
                  +#   o Create an archive containinG the output file.
                  +#
                  +# Author: adedeji.fabiyi@brunel.ac.uk
                  +#
                  +
                  +SW_NAME="repast" # Software name
                  +
                  +echo "--------------------------------------------------"
                  +echo "Job execution starts on: '"$(date)"'"
                  +
                  +echo "---[HOME directory]-------------------------------"
                  +ls -l $HOME
                  +
                  +echo "---[Working directory]-------------------------"
                  +mkdir output
                  +ls -l $(pwd)
                  +
                  +/bin/bash $HOME/$SW_NAME/simulation.sh $1 $2 $3 $4 $5 > stdout
                  +
                  +#
                  +# Following statement produce the simulation_output file
                  +#
                  +OUTFILE=simulation_output.txt
                  +echo "--------------------------------------------------"  > $OUTFILE
                  +echo "Simulation started at: '"$(date)"'"          >> $OUTFILE
                  +echo ""                                                 >> $OUTFILE
                  +echo "#################[  START LOG  ]##################" >> $OUTFILE
                  +echo ""                                                 >> $OUTFILE
                  +cat stdout                                          >> $OUTFILE
                  +echo "#################[   END LOG   ]##################" >> $OUTFILE
                  +echo ""                                                 >> $OUTFILE
                  +echo "Simulation ended at: '"$(date)"'"                   >> $OUTFILE
                  +echo "--------------------------------------------------" >> $OUTFILE
                  +echo ""                                                 >> $OUTFILE
                  +
                  +#
                  +# Collect all generated output files into a single tar.gz file
                  +#
                  +tar cvfz myRepast-infection-Files.tar.gz $OUTFILE output/ 77097 81145
```

## A3 Application Portlet Pages

### Myjobs Portlet

This is a dedicated module that shows the status of all executed jobs. Running jobs are monitored, and the outputs of all completed jobs could be retrieved for analysis by using a dedicated portlet known as MyJobs. (MyJobs portlet can be seen in Figure 5.4 and 5.7).

### The Edit Page

The Edit page is primarily used by the developer to customise the ABMS application as well as setting the portlet preferences. Since the settings of preferences are done within the portlet context, this page is not required by the Science Gateway user. It is only used for the specification and to set the preferences such as the type of infrastructure (name and description of the infrastructure) where the jobs would run, and thus consisting of all the variables that were stored by the AppInfrastructureInfo class (AppInfrastructureInfo class would be described in a later section).

### The Help Page

The Help page is primarily used at the development stage and not necessarily by the Science Gateway user. It defines the ABMS functionalities i.e. the usage instructions of the portlet. It's also used to display other information such as the portlet licence information.

## A4 Granting Access to the Africa Grid Science Gateway (IDPs and IDFs)

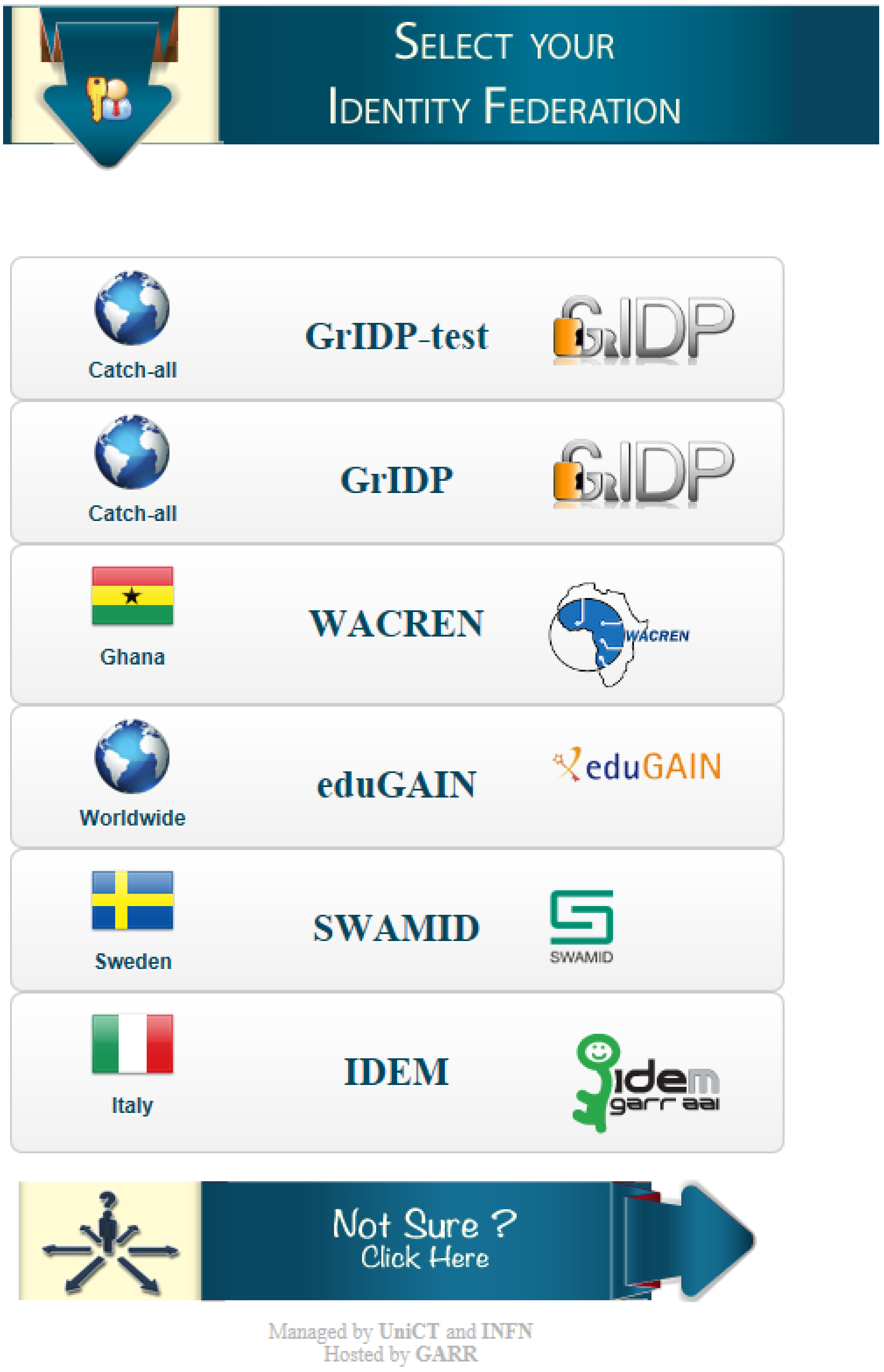
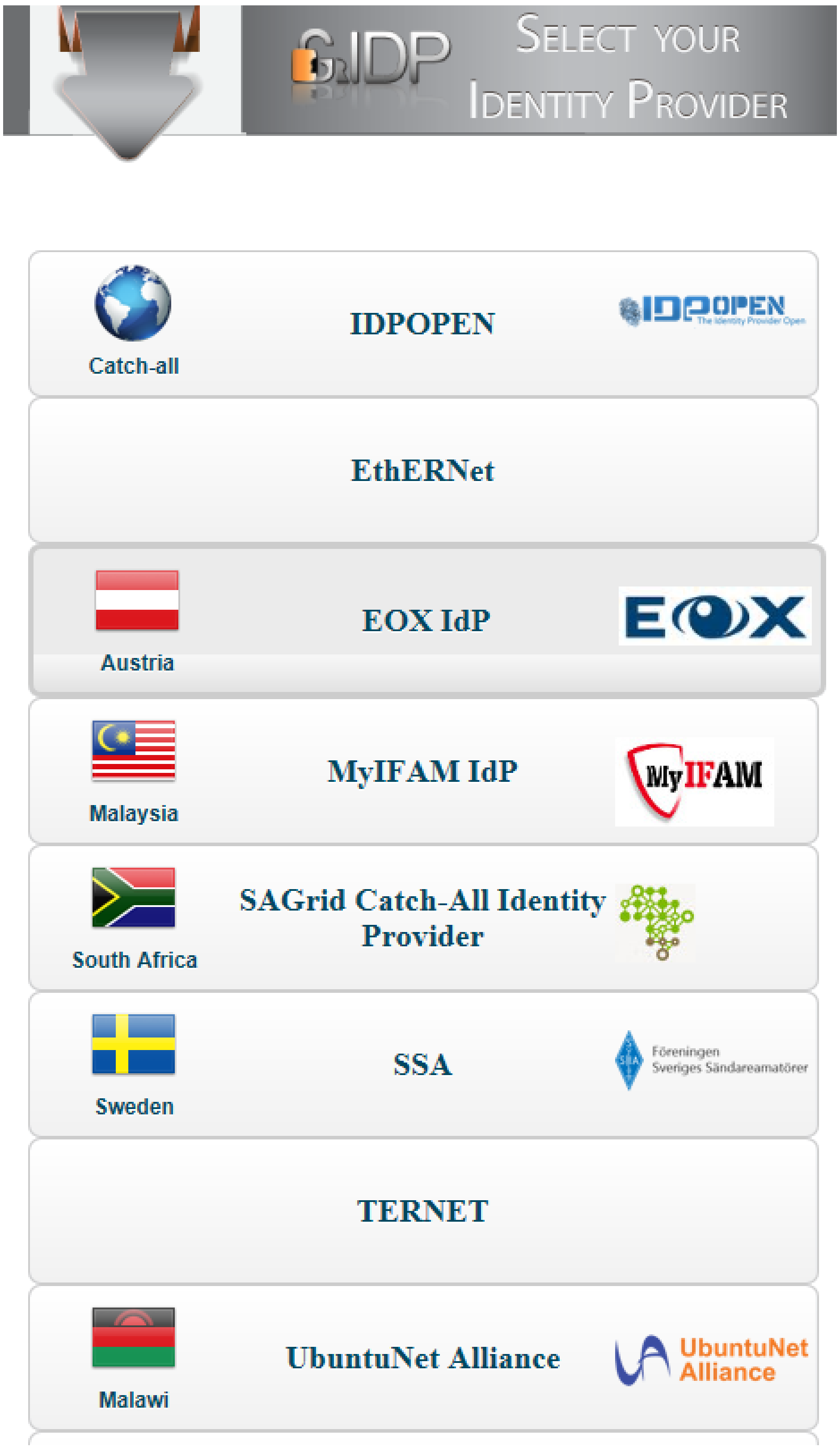
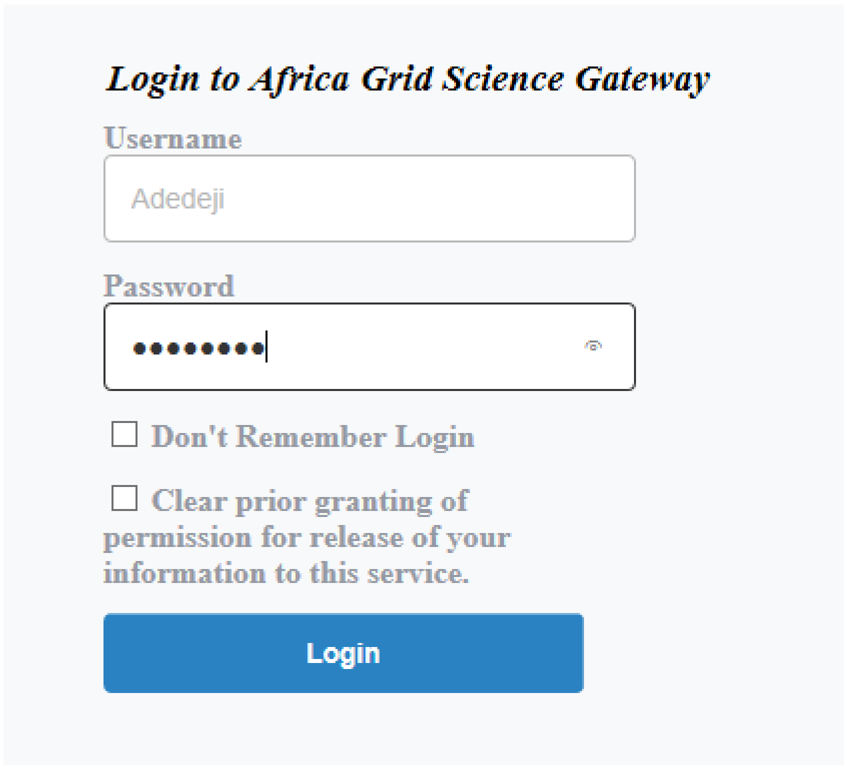


Figure A-3 List of Identity Federations


GIDP Select your Identity Provider

| | | |
|---|---|---|
| Catch-all | IDPOPEN | IDPOPEN The Identity Provider Open |
| | EthERNet | |
| Austria | EOX IdP | EOX |
| Malaysia | MyIFAM IdP | MyIFAM |
| South Africa | SAGrid Catch-All Identity Provider | |
| Sweden | SSA | Föreningen Sveriges Sändareamatörer |
| | TERNET | |
| Malawi | UbuntuNet Alliance | UbuntuNet Alliance |

Figure A-4 List of Identity Providers



*Login to Africa Grid Science Gateway*

Username

Adedeji

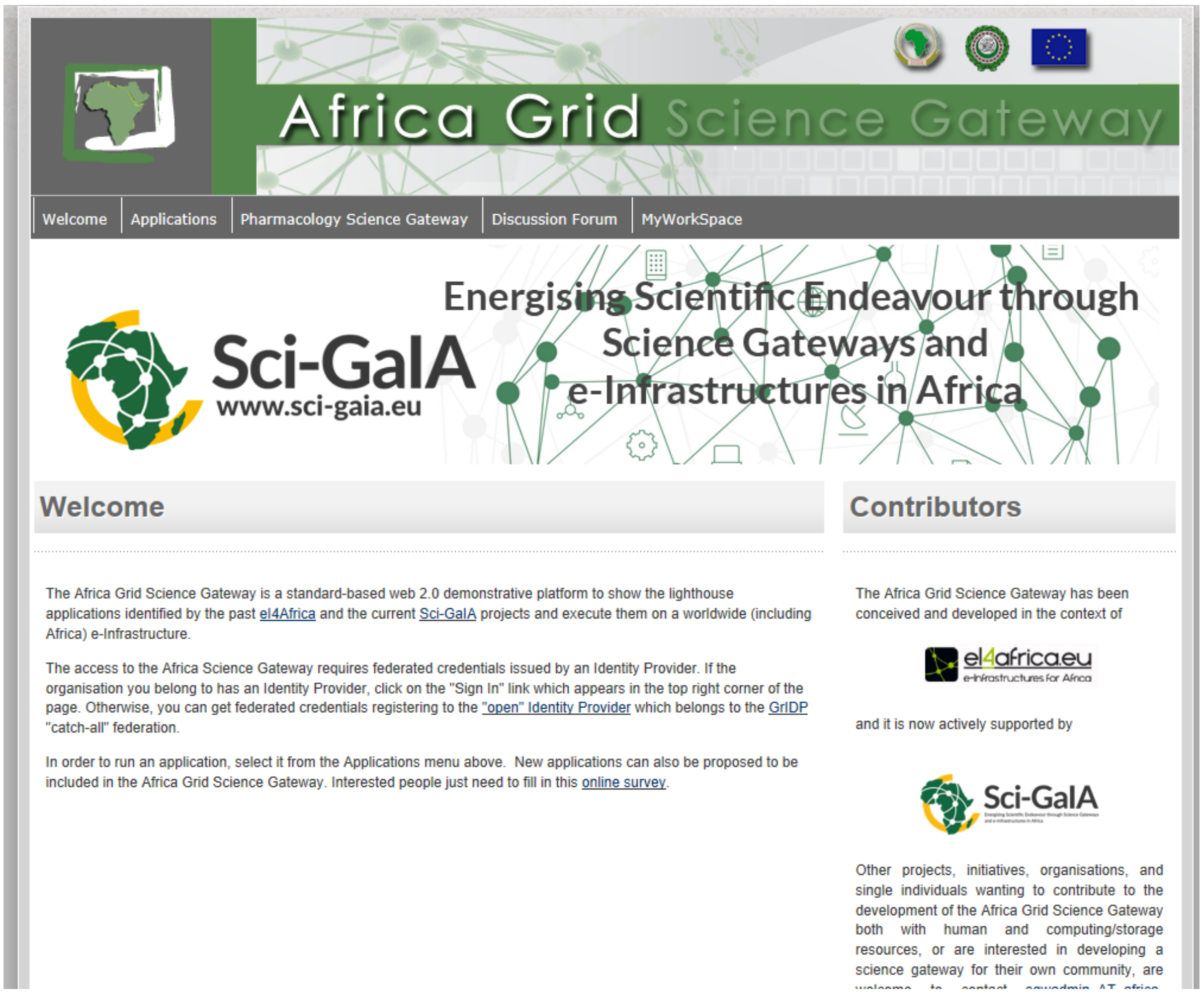
Password

••••••••

☐ Don't Remember Login

☐ Clear prior granting of permission for release of your information to this service.

Figure A-5 Africa Grid Science Gateway Login Page


Africa Grid Science Gateway
Welcome
Applications
Pharmacology Science Gateway
Discussion Forum
MyWorkSpace
Sci-GaIA
www.sci-gaia.eu
Energising Scientific Endeavour through Science Gateways and e-Infrastructures in Africa

Welcome

The Africa Grid Science Gateway is a standard-based web 2.0 demonstrative platform to show the lighthouse applications identified by the past el4Africa and the current Sci-GaIA projects and execute them on a worldwide (including Africa) e-Infrastructure.

The access to the Africa Science Gateway requires federated credentials issued by an Identity Provider. If the organisation you belong to has an Identity Provider, click on the "Sign In" link which appears in the top right corner of the page. Otherwise, you can get federated credentials registering to the "open" Identity Provider which belongs to the GrIDP "catch-all" federation.

In order to run an application, select it from the Applications menu above. New applications can also be proposed to be included in the Africa Grid Science Gateway. Interested people just need to fill in this online survey.

Contributors

The Africa Grid Science Gateway has been conceived and developed in the context of

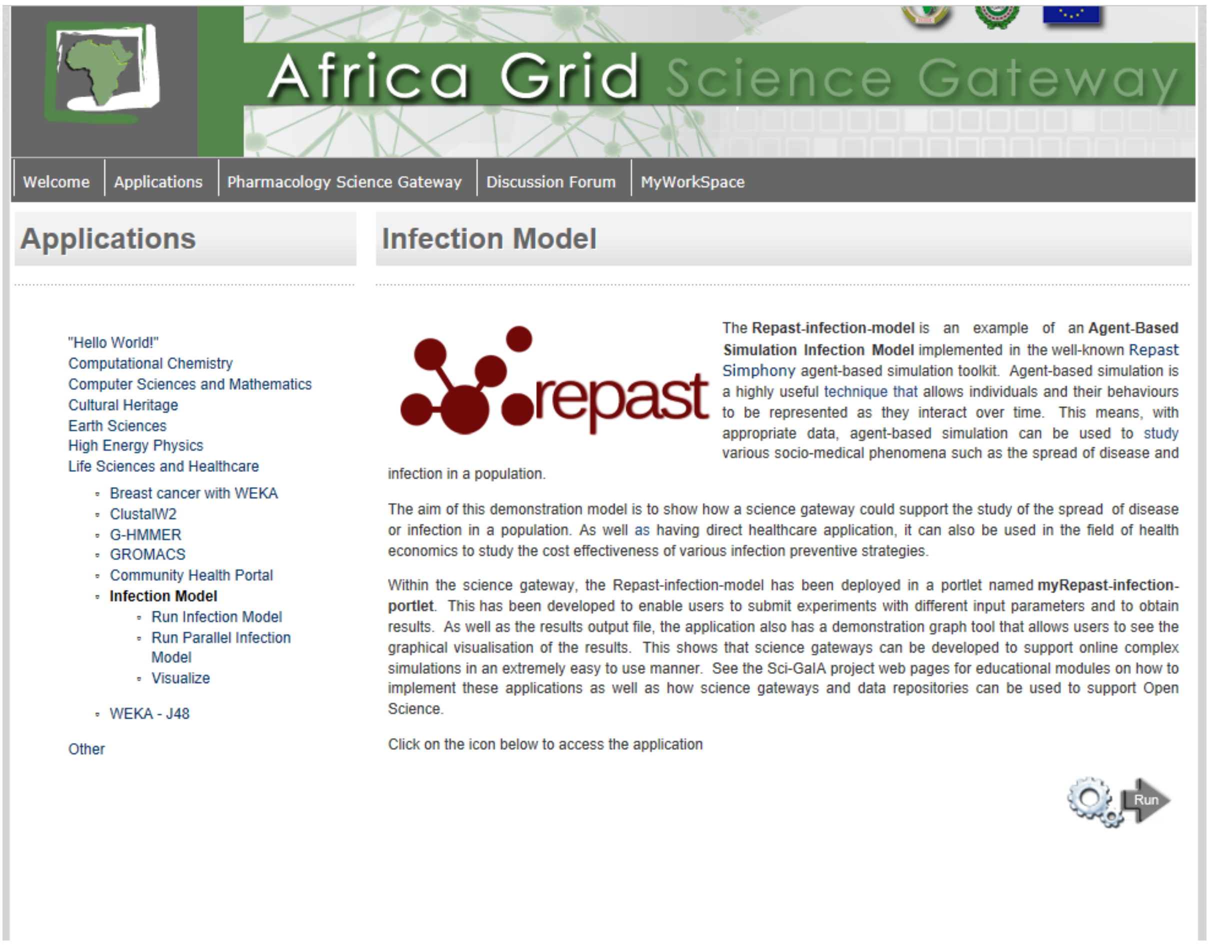
el4africa.eu e-Infrastructures for Africa

and it is now actively supported by

Sci-GaIA

Other projects, initiatives, organisations, and single individuals wanting to contribute to the development of the Africa Grid Science Gateway both with human and computing/storage resources, or are interested in developing a science gateway for their own community, are

Figure A-6 Africa Grid Science Gateway Main Page



Applications

"Hello World!"
Computational Chemistry
Computer Sciences and Mathematics
Cultural Heritage
Earth Sciences
High Energy Physics
Life Sciences and Healthcare

- Breast cancer with WEKA
- ClustalW2
- G-HMMER
- GROMACS
- Community Health Portal
- **Infection Model**
  - Run Infection Model
  - Run Parallel Infection Model
  - Visualize
- WEKA - J48

Other

Infection Model

The **Repast-infection-model** is an example of an **Agent-Based Simulation Infection Model** implemented in the well-known Repast Simphony agent-based simulation toolkit. Agent-based simulation is a highly useful technique that allows individuals and their behaviours to be represented as they interact over time. This means, with appropriate data, agent-based simulation can be used to study various socio-medical phenomena such as the spread of disease and infection in a population.

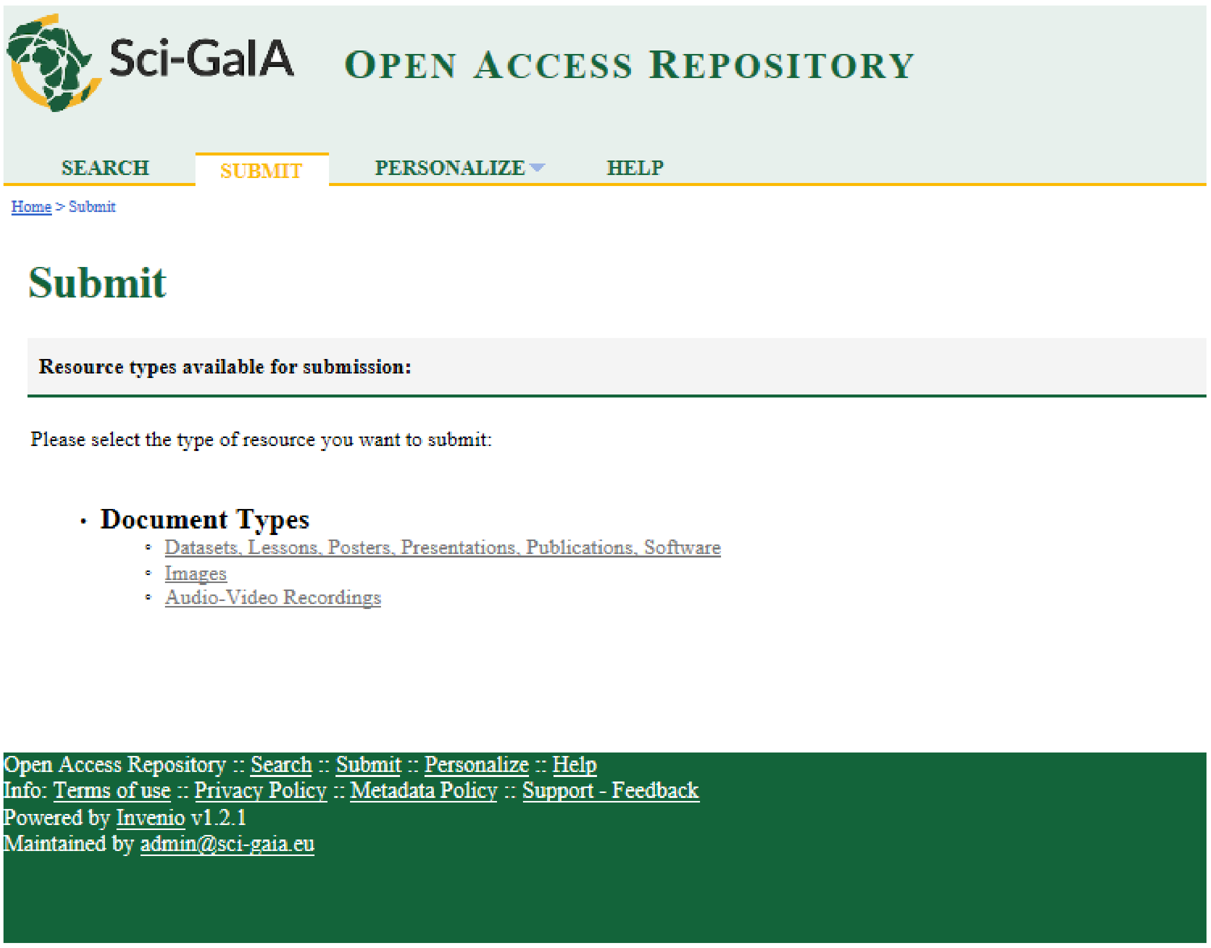
The aim of this demonstration model is to show how a science gateway could support the study of the spread of disease or infection in a population. As well as having direct healthcare application, it can also be used in the field of health economics to study the cost effectiveness of various infection preventive strategies.

Within the science gateway, the Repast-infection-model has been deployed in a portlet named **myRepast-infection-portlet**. This has been developed to enable users to submit experiments with different input parameters and to obtain results. As well as the results output file, the application also has a demonstration graph tool that allows users to see the graphical visualisation of the results. This shows that science gateways can be developed to support online complex simulations in an extremely easy to use manner. See the Sci-GaIA project web pages for educational modules on how to implement these applications as well as how science gateways and data repositories can be used to support Open Science.

Click on the icon below to access the application

Figure A-7 Infection Model Portlet Landing Page

## A5 The Open Access Repository



Submit

Resource types available for submission:

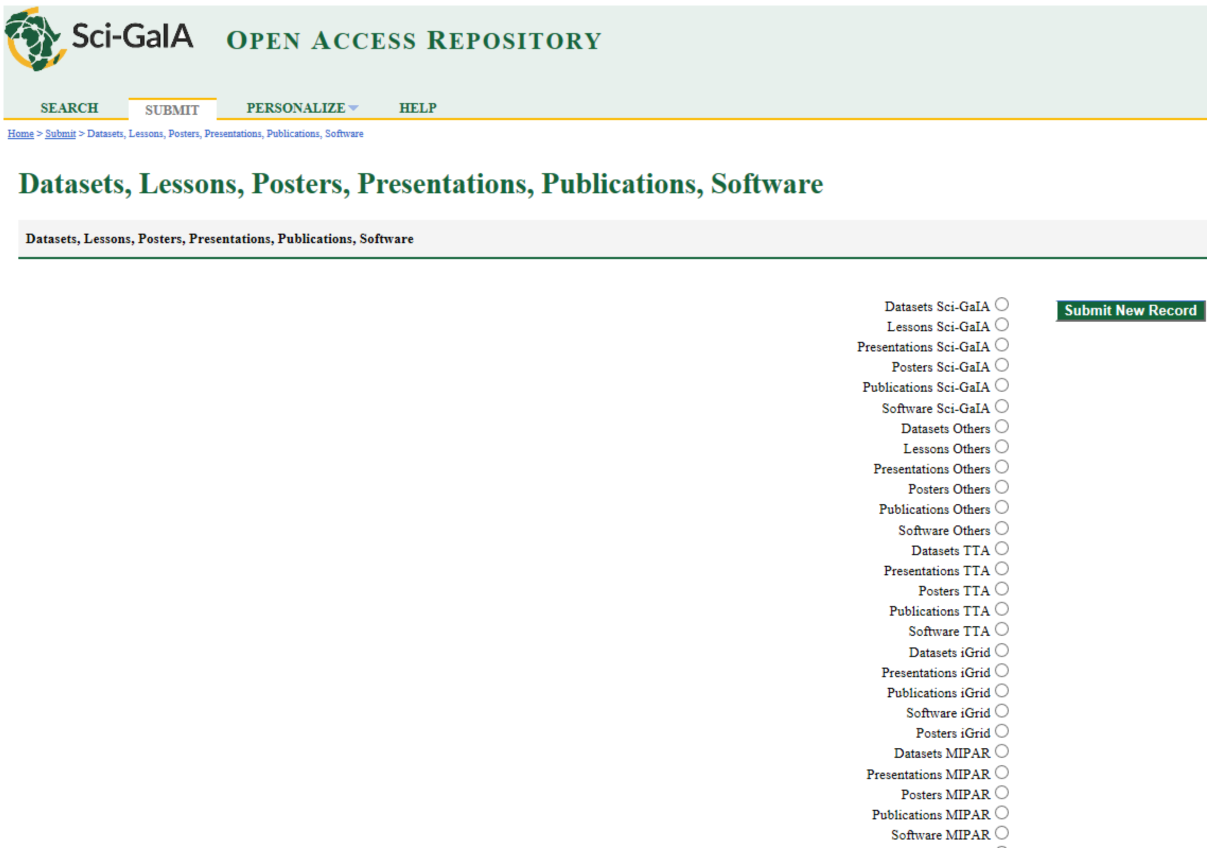
Please select the type of resource you want to submit:

- Document Types
  - Datasets, Lessons, Posters, Presentations, Publications, Software
  - Images
  - Audio-Video Recordings

Open Access Repository :: Search :: Submit :: Personalize :: Help
Info: Terms of use :: Privacy Policy :: Metadata Policy :: Support - Feedback
Powered by Invenio v1.2.1
Maintained by admin@sci-gaia.eu

Figure A-8 Open Science Repository Landing Page

## A6 Submit Research Document of the Open Access Repository



Home > Submit > Datasets, Lessons, Posters, Presentations, Publications, Software

Datasets, Lessons, Posters, Presentations, Publications, Software

Datasets, Lessons, Posters, Presentations, Publications, Software

Datasets Sci-GaIA ○
Lessons Sci-GaIA ○
Presentations Sci-GaIA ○
Posters Sci-GaIA ○
Publications Sci-GaIA ○
Software Sci-GaIA ○
Datasets Others ○
Lessons Others ○
Presentations Others ○
Posters Others ○
Publications Others ○
Software Others ○
Datasets TTA ○
Presentations TTA ○
Posters TTA ○
Publications TTA ○
Software TTA ○
Datasets iGrid ○
Presentations iGrid ○
Publications iGrid ○
Software iGrid ○
Posters iGrid ○
Datasets MIPAR ○
Presentations MIPAR ○
Posters MIPAR ○
Publications MIPAR ○
Software MIPAR ○

Figure A-9 Submit Research Documents using an Open Science Repository

# Appendix B

## B1 Sample Infection Model Main Class

```
import java.io.*;
import java.net.*;
import java.util.Calendar;
import java.text.SimpleDateFormat;
// Importing portlet libraries
import javax.portlet.*;
// Importing liferay libraries
import com.liferay.portal.kernel.util.WebKeys;
import com.liferay.portal.theme.ThemeDisplay;
import com.liferay.portal.model.User;
// Importing Apache libraries
// Importing GridEngine Job libraries
import it.infn.ct.GridEngine.Job.*;
import it.infn.ct.GridEngine.JobResubmission.GEJobDescription;
import it.infn.ct.GridEngine.Job.MultiInfrastructureJobSubmission;
/**
* This is the class that overrides the GenericPortlet class methods
* You can create your own application just customizing this code skeleton
* This code provides mainly a full working example on:
* 1) How to manage user interaction managing the Actions/Viewa combination
* 2) How to manage portlet preferences and help pane
* 3) How to print application information using the Log object
* 4) How to execute a distributed application with GridEngine
*
* @author <a href="adedeji.fabiyi@brunel.ac.uk">Adedeji Fabiyi</a>(COMETA)
*/
public class myRepast_infection_portlet extends GenericPortlet {
// Instantiate the logger object
AppLogger _log = new AppLogger(myRepast_infection_portlet.class);
```

```
// This portlet uses Aciont/Views enumerations in order to
// manage the different portlet modes and the corresponding
// view to display
// You may override the current values with your own business
// logic best identifiers and manage them through: jsp pages and
// this java code
// The jsp parameter PortletStatus will be the responsible of
// portlet mode switching. This parameter will be read by
// the processAction method (actionRequest) who will select
// then the proper view mode. The doView method will read this
// value (renderResponSe) assigning the correct view mode.
//
// At first boot the application will be in ACTIVATE status
// that means the application still requires to be registered
// into the GridEngine' UsersTrackingDB' GridOperations table
// Once registered the defaul view mode will be the VIEW_INPUT
/**
* Actions enumeration contains the possible action status mode
* managed by the application. Action modes are stored into the 'PortletStatus'
* parameter inside the actionRequest object
*/
private enum Actions {
ACTION_ACTIVATE // User (Admin) activated the portlet
,ACTION_INPUT // User asked to submit a job
,ACTION_SUBMIT// User asked to return to the input form
,ACTION_PILOT // The user did something in the edit pilot screen pane
}
/**
* Views enumeration contains the possible view mondes managed b
* the application. View modes are stored into the parameter 'PortletStatus'
* inside the renderResponse object
*/
```

```
private enum Views {
VIEW_ACTIVATE // Show acrivation pane (called 1st time only)
,VIEW_INPUT // View containing application input fields
,VIEW_SUBMIT // View reporting input value // View reporting the job submission
,VIEW_PILOT // Shows the pilot script and makes it editable
}
/**
* Instanciate the AppPreferences object that stores the Application preferences
*
* @see AppPreferences
*/
AppPreferences appPreferences = new AppPreferences(_log);
AppPreferences appInitPreferences = new AppPreferences(_log);
/**
* This class contains all the necessary data to submit a job inside
* a distributed infrastructure. Each submission will instantiate this
* object
*/
class AppInit {
String default_prefvalue;
public AppInit() {
default_prefvalue = "";
}
} // App_Init
// Instanciate the App_Init object
//public App_Init appInit = new App_Init();
//
// Application input values
//
class AppInput {
String inputValue;
String inputValue2;
String inputValue3;//Text for application input field
String inputValue4;
String inputValue5;
String username;
String timestamp;
String jobIdentifier;
String inputSandbox_inputFile;
public AppInput() {
inputValue = "";
inputValue2= "";
inputValue3= "";
inputValue4= "";
inputValue5= "";
username = "";
timestamp = "";
jobIdentifier = "";
inputSandbox_inputFile = "";
}
} // App_Input
// Liferay portal data
// Classes below are used by this portlet code to get information
// about the current user
public String portalName="localhost"; // Name of the hosting portal
public String appServerPath; // This variable stores the absolute path of the Web
 applications
// Other misc valuse
// (!) Pay attention that altough the use of the LS variable
// the replaceAll("\n","") has to be used
public static final String LS = System.getProperty("line.separator");
// Users must have separated inputSandbox files
// these file will be generated into /tmp directory
// and prefixed with the format <timestamp>_<user>_*
// The timestamp format is:
public static final String tsFormat = "yyyyMMddHHmmss";
// This variable holds the GridEngine' GridOperation identifier
// associated to this application
int gridOperationId=-1;
//---------------------------
// Portlet Overriding Methods
//---------------------------
/**
* The init method will be called when installing the portlet for the first time
* or when restarting the portal server.
* This is the right time to get default values from WEBINF/portlet.xml file
* Those values will be assigned into the application preferences as default values
* If preference values already exists for this application the default settings will
* be overwritten
*
* @see AppInfrastructureInfo
* @see AppPreferences
*
* @throws PortletException
*/
@Override
public void init()
throws PortletException
{
// Load default values from WEBINF/portlet.xml
appInitPreferences.setGridOperationDesc (""+getInitParameter( "gridOperationDesc"));
appInitPreferences.setPortletVersion (""+getInitParameter( "portletVersion"));
appInitPreferences.setLogLevel (""+getInitParameter( "logLevel"));
appInitPreferences.setNumInfrastructures(""+getInitParameter("numInfrastructures"));
appInitPreferences.setGridOperationId (""+getInitParameter( "gridOperationId"));
// Get the number of infrastructures to load
```

```
int numInfra=appInitPreferences.getNumInfrastructures();
_log.info("Number of infrastructures: '"+numInfra+"'");
// Load infrastructure settings
for(int i=0; i<numInfra; i++) {
int j=i+1;
appInitPreferences.setInfrastructure(
i
, ""+getInitParameter(j+ "_enableInfrastructure")
, ""+getInitParameter(j+ "_nameInfrastructure")
, ""+getInitParameter(j+"_acronymInfrastructure")
, ""+getInitParameter(j+ "_bdiiHost")
, ""+getInitParameter(j+ "_wmsHosts")
, ""+getInitParameter(j+ "_pxServerHost")
, ""+getInitParameter(j+ "_pxServerPort")
, ""+getInitParameter(j+ "_pxServerSecure")
, ""+getInitParameter(j+ "_pxRobotId")
, ""+getInitParameter(j+ "_pxRobotVO")
, ""+getInitParameter(j+ "_pxRobotRole")
, ""+getInitParameter(j+ "_pxRobotRenewalFlag")
, ""+getInitParameter(j+ "_pxUserProxy")
, ""+getInitParameter(j+ "_softwareTags")
);
} // Load infrastructure settings
appInitPreferences.setSciGwyUserTrackingDB_Hostname(""+getInitParameter
("sciGwyUserTrackingDB_Hostname"));
appInitPreferences.setSciGwyUserTrackingDB_Username(""+getInitParameter
("sciGwyUserTrackingDB_Username"));
appInitPreferences.setSciGwyUserTrackingDB_Password(""+getInitParameter
("sciGwyUserTrackingDB_Password"));
appInitPreferences.setSciGwyUserTrackingDB_Database(""+getInitParameter
("sciGwyUserTrackingDB_Database"));
appInitPreferences.setJobRequirements (""+getInitParameter( "jobRequirements"));
appInitPreferences.setPilotScript (""+getInitParameter( "pilotScript"));
// Assigns the log level
_log.setLogLevel(appInitPreferences.getLogLevel());
// Show loaded values into log
_log.info(appInitPreferences.dump());
} // init
/**
* This method allows the portlet to process an action request; this method is normally
* called upon each user interaction (i.e. A submit button inside a jsp' <form statement)
* This method determines the current application mode through the actionRequest value:
* 'PortletStatus' and then determines the correct view mode to assign through the
* ActionResponse 'PortletStatus' variable that will be read by the doView
* This method will also takes care about the std JSR168/286: EDIT and HELP portlet modes.
*
* @param request ActionRequest object instance
* @param response ActionResponse object instance
*
* @throws PortletException
* @throws IOException
*/
@Override
public void processAction(ActionRequest request, ActionResponse response)
throws PortletException, IOException
{
_log.info("calling processAction ...");
// Determine the username
ThemeDisplay themeDisplay = (ThemeDisplay)request.getAttribute
(WebKeys.THEME_DISPLAY);
User user = themeDisplay.getUser();
String username = user.getScreenName();
// Determine the application pathname
PortletSession portletSession = request.getPortletSession();
```

```
                    PortletContext portletContext = portletSession.getPortletContext();
                    appServerPath = portletContext.getRealPath("/");
                    // Show info
                    _log.info("appUserName : '"+username +"'"
                    +LS+"appServerPath : '"+appServerPath+"'"
                    );
                    // Determine the current portlet mode and forward this state to the response
                    // Accordingly to JSRs168/286 the standard portlet modes are:
                    // VIEW, EDIT, HELP
                    PortletMode mode = request.getPortletMode();
                    response.setPortletMode(mode);
                    // Switch among different portlet modes: VIEW, EDIT, HELP
                    // any custom modes are not covered by this template
//----------
                    // VIEW Mode
                    //
                    // The actionStatus value will be taken from the calling jsp file
                    // through the 'PortletStatus' parameter; the corresponding
                    // VIEW mode will be stored registering the portlet status
                    // as render parameter. See the call to setRenderParameter
                    // If the actionStatus parameter is null or empty the default
                    // action will be the ACTION_INPUT (input form)
                    // This happens the first time the portlet is shown
                    // The PortletStatus variable is managed by jsp and this java code
                    //----------
                    if (mode.equals(PortletMode.VIEW)) {
                    // The VIEW mode is the normal portlet mode where normal portlet
                    // content will be shown to the user
                    _log.info("Portlet mode: VIEW");
                    String actionStatus=request.getParameter("PortletStatus");
                    // Assigns the default ACTION mode
                    if( null==actionStatus
                    || actionStatus.equals(""))
                    actionStatus=""+Actions.ACTION_INPUT;
                    // Different actions will be performed accordingly to the
                    // different possible statuses
                    AppInput appInput = null;
                    switch(Actions.valueOf(actionStatus)) {
                    case ACTION_ACTIVATE:
                    _log.info("Got action: 'ACTION_ACTIVATE'");
                    // Called when activating the portlet for the first time
                    // it will be used to save the gridOperationId value
                    // into the application preferences
                    gridOperationId=Integer.parseInt(request.getParameter("gridOperationId"));
                    _log.info("Received gridOperationId: '"+gridOperationId+"'");
                    // If the application is registered go to the VIEW_INPUT
                    // and the application will no longer go to the ACTIVATE pane
                    if(gridOperationId > 0) {
                    storePreferences(request);
                    response.setRenderParameter("PortletStatus",""+Views.VIEW_INPUT);
                    }
                    break;
                    case ACTION_INPUT:
                    _log.info("Got action: 'ACTION_INPUT'");
                    // Assign the correct view
                    response.setRenderParameter("PortletStatus",""+Views.VIEW_INPUT);
                    break;
                    case ACTION_PILOT:
                    _log.info("Got action: 'ACTION_PILOT'");
                    // Stores the new pilot script
                    String pilotScript=request.getParameter("pilotScript");
                    pilotScript.replaceAll("\r", "");
                    storeString(appServerPath+"WEB-INF/job/"+appPreferences.getPilotScript(),pilotScript);
                    // Assign the correct view
```

```
response.setPortletMode(PortletMode.EDIT);
break;
case ACTION_SUBMIT:
_log.info("Got action: 'ACTION_SUBMIT'");
// Get current preference values
getPreferences(request,null);
// Create the appInput object
appInput = new AppInput();
// Stores the user submitting the job
appInput.username=username;
// Determine the submissionTimeStamp
SimpleDateFormat dateFormat = new SimpleDateFormat(tsFormat);
String timestamp = dateFormat.format(Calendar.getInstance().getTime());
appInput.timestamp=timestamp;
// Process input fields and files to upload
getInputForm(request,appInput);
// Following files have to be updated with
// values taken from textareas or from uploaded files:
// input_file.txt
//updateFiles(appInput);
// Submit the job
submitJob(appInput);
// Send the jobIdentifier and assign the correct view
response.setRenderParameter("PortletStatus",""+Views.VIEW_SUBMIT);
response.setRenderParameter("inputValue", "" + appInput.inputValue);
break;
default:
_log.info("Unhandled action: '"+actionStatus+"'");
response.setRenderParameter("PortletStatus",""+Views.VIEW_INPUT);
} // switch actionStatus
} // VIEW
//----------
```

```
// HELP Mode
//
// The HELP mode used to give portlet usage HELP to the user
// This code will be called after the call to doHelp method
//----------
else if(mode.equals(PortletMode.HELP)) {
_log.info("Portlet mode: HELP");
}
//----------
// EDIT Mode
//
// The EDIT mode is used to view/setup portlet preferences
// This code will be called after the user sends the actionURL
// generated by the doEdit method
// The code below just stores new preference values or
// reacts to the preference settings changes
//----------
else if(mode.equals(PortletMode.EDIT)) {
_log.info("Portlet mode: EDIT");
// Retrieve the current ifnrstructure in preference
int numInfrastructures=appPreferences.getNumInfrastructures ();
int currInfra =appPreferences.getCurrPaneInfrastructure();
_log.info(
LS+"Number of infrastructures: '"+numInfrastructures+"'"
+LS+"currentInfrastructure: '"+currInfra +"'"
+LS);
// Take care of the preference action (Infrastructure preferences)
// <,>,+,- buttons
String pref_action=""+request.getParameter("pref_action");
_log.info("pref_action: '"+pref_action+"'");
// Reacts to the current infrastructure change and
// determine the next view mode (return to the input pane)
```

```
if(pref_action.equalsIgnoreCase("next")) {
appPreferences.switchNextInfrastructure();
_log.info("Got next infrastructure action; switching to: '"+
appPreferences.getCurrPaneInfrastructure()+"'");
}
else if(pref_action.equalsIgnoreCase("previous")) {
appPreferences.switchPreviousInfrastructure();
_log.info("Got prev infrastructure action; switching to: '"+
appPreferences.getCurrPaneInfrastructure()+"'");
}
else if(pref_action.equalsIgnoreCase("add")) {
appPreferences.addNewInfrastructure();
_log.info("Got add infrastructure action; current infrastrucure is now: '"+
appPreferences.getCurrPaneInfrastructure()+"'");
}
else if(pref_action.equalsIgnoreCase("remove")) {
appPreferences.delCurrInfrastructure();
_log.info("Got remove infrastructure action; current infrastrucure is now: '"+
appPreferences.getCurrPaneInfrastructure()+"' and infrastructures are now: '"+
appPreferences.getNumInfrastructures()+"'");
}
else if(pref_action.equalsIgnoreCase("done")) {
// None of the above actions selected; return to the VIEW mode
response.setPortletMode(PortletMode.VIEW);
response.setRenderParameter("PortletStatus", ""+Views.VIEW_INPUT);
}
else if(pref_action.equalsIgnoreCase("viewPilot")) {
// None of the above actions selected; return to the VIEW mode
response.setPortletMode(PortletMode.VIEW);
response.setRenderParameter("PortletStatus",""+Views.VIEW_PILOT);
response.setRenderParameter("pilotScript"
,updateString(appServerPath+"WEB-INF/job/"+appPreferences.getPilotScript()));
```

```
}
else {
// No other special actions to do ...
}
// Number of infrastructures and Currentinfrastructure values
// may be changed by add/delete,<,> actions
int newCurrInfra =appPreferences.getCurrPaneInfrastructure();
int newNumInfrastructures=appPreferences.getNumInfrastructures ();
// Store infrastructure changes
String infrastructuresInformations="";
// Preference settings (logLevel has been taken above)
String newpref_logLevel = ""+request.getParameter( "pref_logLevel");
String newpref_gridOperationId = ""+request.getParameter("pref_gridOperationId");
String newpref_jobRequirements = ""+request.getParameter("pref_jobRequirements");
String newpref_pilotScript = ""+request.getParameter( "pref_pilotScript");
//LIC
String newpref_sciGwyUserTrackingDB_Hostname = ""+request.getParameter
("pref_sciGwyUserTrackingDB_Hostname");
String newpref_sciGwyUserTrackingDB_Username = ""+request.getParameter
("pref_sciGwyUserTrackingDB_Username");
String newpref_sciGwyUserTrackingDB_Password = ""+request.getParameter
("pref_sciGwyUserTrackingDB_Password");
String newpref_sciGwyUserTrackingDB_Database = ""+request.getParameter
("pref_sciGwyUserTrackingDB_Database");
// Store infrastructure changes only if the user did not select the delete button
if(newNumInfrastructures >= numInfrastructures) {
// Current infrastructure preference settings
```

// Assign the
new variable to
the preference
object

```
appPreferences.setLogLevel ( newpref_logLevel);
appPreferences.setGridOperationId(newpref_gridOperationId);
appPreferences.setJobRequirements(newpref_jobRequirements);
appPreferences.setPilotScript ( newpref_pilotScript);
//LIC
appPreferences.setSciGwyUserTrackingDB_Hostname
(newpref_sciGwyUserTrackingDB_Hostname);
appPreferences.setSciGwyUserTrackingDB_Username(newpref_sciGwyUser
TrackingDB_Username);
appPreferences.setSciGwyUserTrackingDB_Password(newpref_sciGwyUser
TrackingDB_Password);
appPreferences.setSciGwyUserTrackingDB_Database(newpref_sciGwyUser
TrackingDB_Database);
// Store new preferences
storePreferences(request);
} // EDIT Mode
//----------
// EDIT Mode
//
// Any custom portlet mode should be placed here below
//----------
else {
// Unsupported portlet modes come here
_log.warn("Custom portlet mode: '"+mode.toString()+"'");
} // CUSTOM Mode
} // processAction
/**
* This method is responsible to assign the correct Application view
* the view mode is taken from the renderRequest instance by the PortletStatus patameter
* or automatically assigned accordingly to the Application status/default view mode
*
* @param request RenderRequest instance normally sent by the processAction
* @param response RenderResponse used to send values to the jsp page
```

```
*
* @throws PortletException
* @throws IOException
*/
@Override
protected void doView(RenderRequest request, RenderResponse response)
throws PortletException, IOException
{
_log.info("calling doView ...");
response.setContentType("text/html");
// Get current preference values
getPreferences(null,request);
gridOperationId=Integer.parseInt(appPreferences.getGridOperationId());
_log.info("GridOperationId: '"+gridOperationId+"'");
// currentView comes from the processAction; unless such method
// is not called before (example: page shown with no user action)
// In case the application is not yet register (gridOperationId<0)
// the VIEW_INITIALIZE pane will be enforced otherwise the
// VIEW_INPUT will be selected as default view
String currentView=request.getParameter("PortletStatus");
if(currentView==null) currentView="VIEW_INPUT";
if(gridOperationId<0) currentView="VIEW_ACTIVATE";
// Different actions will be performed accordingly to the
// different possible view modes
switch(Views.valueOf(currentView)) {
// The following code is responsible to call the proper jsp file
// that will provide the correct portlet interface
case VIEW_ACTIVATE: {
_log.info("VIEW_ACTIVATE Selected ...");
/*
* Following statements requires a patch on the GridEngine not yet included and related
* to the application auto registration feature (see code appAutoRegistration.java)
```

```
*
// Portlet uses the couple (portalName,GridOperationDesc)
// to be identified by the GridEngine UserTrackingDB
// VIEW_INITIALIZE checks if the Application is already
// registered or not
try {
UsersTrackingDBInterface utDB = new UsersTrackingDBInterface();
Company company = PortalUtil.getCompany(request);
portalName = company.getName();
String operationDesc = appPreferences.getGridOperationDesc();
int utId = utDB.registerOperation(portalName, operationDesc);
PortletPreferences portletPreferences= request.getPreferences();
// Show values ...
_log.info(LS+"Check configuration"
+LS+"--------------------"
+LS+"utId : '"+utId+"'"
+LS+"portalName : '"+portalName+"'"
+LS+"operationDesc : '"+appPreferences.getGridOperationDesc()+"'"
); // _log
// Show the registration page
request.setAttribute("gridOperationDesc", operationDesc);
request.setAttribute( "gridOperationId", utId);
request.setAttribute( "portal", portalName);
request.setAttribute( "appPreferences",appPreferences);
PortletRequestDispatcher dispatcher=getPortletContext().getRequestDispatcher
("/activate.jsp");
dispatcher.include(request, response);
}
catch (PortalException ex) {
_log.error("Got exception: '"+ex.toString()+"'");
}
catch (SystemException ex) {
```

```
_log.error("Got exception: '"+ex.toString()+"'");
}
* Application autoregistration feature
*/
}
break;
case VIEW_INPUT: {
_log.info("VIEW_INPUT Selected ...");
PortletRequestDispatcher dispatcher=getPortletContext().getRequestDispatcher
("/input.jsp");
dispatcher.include(request, response);
}
break;
case VIEW_PILOT: {
_log.info("VIEW_PILOT Selected ...");
String pilotScript = request.getParameter("pilotScript");
request.setAttribute("pilotScript", pilotScript);
PortletRequestDispatcher dispatcher=getPortletContext().getRequestDispatcher
("/viewPilot.jsp");
dispatcher.include(request, response);
}
break;
case VIEW_SUBMIT: {
_log.info("VIEW_SUBMIT Selected ...");
String inputValue = request.getParameter("inputValue");
request.setAttribute("inputValue", inputValue);
PortletRequestDispatcher dispatcher = getPortletContext().getRequestDispatcher
("/submit.jsp");
dispatcher.include(request, response);
}
break;
default:
```

```
_log.info("Unknown view mode: "+currentView.toString());
} // switch
} // doView
/**
* This method is responsible to retrieve the current Application preference settings
* and then show the edit.jsp page where the user can edit the Application preferences
* This methods prepares an actionURL that will be used by edit.jsp file into a <input ...>
* As soon the user press the action button the processAction will be called going in EDIT
* This method is equivalent to the doView method
*
* @param request Render request object instance
* @param response Render response object isntance
*
* @throws PortletException
* @throws IOException
*
*/
@Override
public void doEdit(RenderRequest request,RenderResponse response)
throws PortletException,IOException {
response.setContentType("text/html");
_log.info("Calling doEdit ...");
// Get current preference values
getPreferences(null,request);
// Get the current infrastructure and the number of infrastructure
int currInfra =appPreferences.getCurrPaneInfrastructure();
int numInfrastructures=appPreferences.getNumInfrastructures();
// ActionURL and the current preference value will be passed to the edit.jsp
PortletURL pref_actionURL = response.createActionURL();
request.setAttribute("pref_actionURL",pref_actionURL.toString());
// Send preference values
request.setAttribute("pref_logLevel" ,""+appPreferences.getLogLevel ());
```

```
                    request.setAttribute("pref_numInfrastructures",""+appPreferences.getNumInfrastructures ());
                    request.setAttribute("pref_currInfrastructure",""+appPreferences.getCurrPaneInfrastructure());
                    request.setAttribute("pref_gridOperationId" ,""+appPreferences.getGridOperationId ());
                    request.setAttribute("pref_gridOperationDesc" ,""+appPreferences.getGridOperationDesc ());
 /**
          * This method just calls the jsp responsible to show the portlet information
          * This method is equivalent to the doView method
          *
          * @param request Render request object instance
          * @param response Render response object isntance
          *
          * @throws PortletException
          * @throws IOException
          */
          @Override
          public void doHelp(RenderRequest request, RenderResponse response)
          throws PortletException,IOException {
          _log.info("Calling doHelp ...");
          response.setContentType("text/html");
          request.setAttribute("portletVersion",appPreferences.getPortletVersion());
          PortletRequestDispatcher dispatcher=getPortletContext().getRequestDispatcher("/help.jsp");
          dispatcher.include(request, response);
          } // doHelp
          //---------------------------
          // Portlet Standard Methods
          //---------------------------
          /**
          * This method is used to retrieve from the Application preferences the
          * GridEngine' GridOperations identifier associated to this application
          * Such index is automatically created when registering the application
          * with the couple (portalName,applicationDesc)
          * The portal name is automatically extracted from the Application
```

```
* The portal description is defined in the default parameters (portlet.xml)
* This method can be called by processAction or doViewe evakuating one of the
* corresponding actionRequest or renderRequest object instances
*
* @param actionRequest an ActionRequest instance or,
* @param renderRequest a RenderRequest instance
* @return The GridOperationId associated to this application or -1 if the application is not yet
registered
*
* @see AppPreferences
*
private String getPrefGridOperationId(ActionRequest actionRequest
, RenderRequest renderRequest) {
PortletPreferences portletPreferences;
String prefOperationId="";
if(null != actionRequest) {
portletPreferences= actionRequest.getPreferences();
prefOperationId = portletPreferences.getValue("pref_gridOperationId","-1");
}
if(null != renderRequest) {
portletPreferences= renderRequest.getPreferences();
prefOperationId = portletPreferences.getValue("pref_gridOperationId","-1");
}
return prefOperationId;
}
*/
/**
* This method Uses the AppPreference object settings to store Application preferences
*
* @param request ActinRequest instance (called by the processAction)
*
* @throws PortletException
* @throws IOException
*/
void storePreferences(ActionRequest request)
throws PortletException, IOException{
_log.info("Calling storePreferences ...");
// Stored preference content
String storedPrefs="Stored preferences:"
+LS+"-------------------"
+LS;
// The code below stores all the portlet preference values
PortletPreferences prefs = request.getPreferences();
if(prefs!=null) {
String logLevel =appPreferences.getLogLevel ();
String gridOperationId =appPreferences.getGridOperationId ();
int numInfrastructures =appPreferences.getNumInfrastructures ();
int currPaneInfrastructure=appPreferences.getCurrPaneInfrastructure();
String gridOperationDesc =appPreferences.getGridOperationDesc ();
prefs.setValue("pref_logLevel" , ""+logLevel );
prefs.setValue("pref_gridOperationId" , ""+gridOperationId );
prefs.setValue("pref_gridOperationDesc" , ""+gridOperationDesc );
prefs.setValue("pref_numInfrastructures", ""+numInfrastructures );
prefs.setValue("pref_currInfrastructure", ""+currPaneInfrastructure);
storedPrefs+="pref_logLevel : '"+logLevel +"'"
+LS+"pref_gridOperationId : '"+gridOperationId +"'"
+LS+"pref_gridOperationDesc : '"+gridOperationDesc +"'"
+LS+"pref_numInfrastructures : '"+numInfrastructures +"'"
+LS+"pref_currInfrastructure : '"+currPaneInfrastructure+"'"
+LS;
// For each preference infrastructure
for(int i=0; i<numInfrastructures; i++) {
int j=i+1;
storedPrefs=LS+"Infrastructure #"+j
```

```
            +LS+"--------------------"
            +LS;
            String enableInfrastructure =appPreferences.getEnableInfrastructure (i);
            String nameInfrastructure =appPreferences.getNameInfrastructure (i);
            String acronymInfrastructure=appPreferences.getAcronymInfrastructure(i);
            String bdiiHost =appPreferences.getBdiiHost (i);
            String wmsHost =appPreferences.getWmsHosts (i);
            String pxServerHost =appPreferences.getPxServerHost (i);
            String pxServerPort =appPreferences.getPxServerPort (i);
            String pxServerSecure =appPreferences.getPxServerSecure (i);
            String pxRobotId =appPreferences.getPxRobotId (i);
            String pxRobotVO =appPreferences.getPxRobotVO (i);
            String pxRobotRole =appPreferences.getPxRobotRole (i);
            String pxRobotRenewalFlag =appPreferences.getPxRobotRenewalFlag (i);
            String pxUserProxy =appPreferences.getPxUserProxy (i);
            String softwareTags =appPreferences.getSoftwareTags (i);
            // Set preference values
            prefs.setValue("pref_"+j+"_enableInfrastructure" ,enableInfrastructure );
            prefs.setValue("pref_"+j+"_nameInfrastructure" ,nameInfrastructure );
            prefs.setValue("pref_"+j+"_acronymInfrastructure",acronymInfrastructure);
            prefs.setValue("pref_"+j+"_bdiiHost" ,bdiiHost );
            prefs.setValue("pref_"+j+"_wmsHosts" ,wmsHost );
            prefs.setValue("pref_"+j+"_pxServerHost" ,pxServerHost );
            prefs.setValue("pref_"+j+"_pxServerPort" ,pxServerPort );
            prefs.setValue("pref_"+j+"_pxServerSecure" ,pxServerSecure );
            prefs.setValue("pref_"+j+"_pxRobotId" ,pxRobotId );
            prefs.setValue("pref_"+j+"_pxRobotVO" ,pxRobotVO );
            prefs.setValue("pref_"+j+"_pxRobotRole" ,pxRobotRole );
            prefs.setValue("pref_"+j+"_pxRobotRenewalFlag" ,pxRobotRenewalFlag );
            prefs.setValue("pref_"+j+"_pxUserProxy" ,pxUserProxy );
            prefs.setValue("pref_"+j+"_softwareTags" ,softwareTags );
 // Dumps the
```

infrastructure

preferences

```
storedPrefs+= " pref_"+j+"_enableInfrastructure : '"+enableInfrastructure +"'"
+LS+" pref_"+j+"_nameInfrastructure : '"+nameInfrastructure +"'"
+LS+" pref_"+j+"_acronymInfrastructure: '"+acronymInfrastructure+"'"
+LS+" pref_"+j+"_bdiiHost : '"+bdiiHost +"'"
+LS+" pref_"+j+"_wmsHosts : '"+wmsHost +"'"
+LS+" pref_"+j+"_pxServerHost : '"+pxServerHost +"'"
+LS+" pref_"+j+"_pxServerPort : '"+pxServerPort +"'"
+LS+" pref_"+j+"_pxServerSecure : '"+pxServerSecure +"'"
+LS+" pref_"+j+"_pxRobotId : '"+pxRobotId +"'"
+LS+" pref_"+j+"_pxRobotVO : '"+pxRobotVO +"'"
+LS+" pref_"+j+"_pxRobotRole : '"+pxRobotRole +"'"
+LS+" pref_"+j+"_pxRobotRenewalFlag : '"+pxRobotRenewalFlag +"'"
+LS+" pref_"+j+"_pxUserProxy : '"+pxUserProxy +"'"
+LS+" pref_"+j+"_softwareTags : '"+softwareTags +"'"
+LS;
} // for each preference infrastructure
String jobRequirements=appInitPreferences.getJobRequirements();
String pilotScript =appInitPreferences.getPilotScript ();
prefs.setValue("pref_jobRequirements", jobRequirements);
prefs.setValue("pref_pilotScript" , pilotScript );
//LIC
String sciGwyUserTrackingDB_Hostname=appPreferences.getSciGwy
UserTrackingDB_Hostname();
String sciGwyUserTrackingDB_Username=appPreferences.getSciGwyUserTrackingDB_
Username();
String sciGwyUserTrackingDB_Password=appPreferences.getSciGwy
UserTrackingDB_Password();
String sciGwyUserTrackingDB_Database=appPreferences.getSciGwy
UserTrackingDB_Database();
prefs.setValue("pref_sciGwyUserTrackingDB_Hostname",sciGwy
UserTrackingDB_Hostname);
prefs.setValue("pref_sciGwyUserTrackingDB_Username",sciGwy
UserTrackingDB_Username);
prefs.setValue("pref_sciGwyUserTrackingDB_Password",sciGwy
UserTrackingDB_Password);
prefs.setValue("pref_sciGwyUserTrackingDB_Database",sciGwy
UserTrackingDB_Database);
storedPrefs+= "pref_jobRequirements : '"+jobRequirements +"'"
+LS+"pref_pilotScript : '"+pilotScript +"'"
+LS+"pref_sciGwyUserTrackingDB_Hostname: '"+sciGwyUserTrackingDB_Hostname+"'"
+LS+"pref_sciGwyUserTrackingDB_Username: '"+sciGwyUserTrackingDB_Username+"'"
+LS+"pref_sciGwyUserTrackingDB_Password: '"+sciGwyUserTrackingDB_Password+"'"
+LS+"pref_sciGwyUserTrackingDB_Database: '"+sciGwyUserTrackingDB_Database+"'"
+LS;
// Store preferences
prefs.store();
} // pref !=null
// Show saved preferences
_log.info("Stored preferences"
+LS+"------------------"
+storedPrefs
+LS);
} // storePreferences
/**
* This method fills the appPreferences values retrieving them from the
* portlet preference object.
* This method can be called by both processAction or doView methods
* in case no preference values are yet defined the default settings loaded
* by the init method will be used
*
* @param actionRequest an ActionRequest instance or,
* @param renderRequest a RenderRequest instance
```

```
                    *
                    */
                    private void getPreferences( ActionRequest actionRequest
                    ,RenderRequest renderRequest) {
                    _log.info("Calling: getPreferences ...");
                    PortletPreferences prefs=null;
                    if(null!=actionRequest)
                    prefs = actionRequest.getPreferences();
                    else if(null != renderRequest)
                    prefs = renderRequest.getPreferences();
                    else _log.warn("Both render request and action request are null");
                    if (null != prefs) {
                    appPreferences.updateValue( "logLevel",""+prefs.getValue( "pref_logLevel",
                    appInitPreferences.getLogLevel ()));
                    appPreferences.updateValue( "gridOperationId",""+prefs.getValue
                    ( "pref_gridOperationId",
                    appInitPreferences.getGridOperationId ()));
                    appPreferences.updateValue( "gridOperationDesc",""+prefs.getValue
                    ( "pref_gridOperationDesc", appInitPreferences.getGridOperationDesc ()));
                    appPreferences.updateValue("numInfrastructures",""+prefs.getValue
                    ("pref_numInfrastructures",""+appInitPreferences.getNumInfrastructures()));
                    // Now retrieves the infrastructures information
                    int numInfras=appPreferences.getNumInfrastructures();
                    _log.info("getpref: num infra="+numInfras);
                    // For each infrastructure ...
                    // The preference name is indexed with the infrastructure number: 1,2,...
                    String infrastructuresInfrormations="";
 /**
      * This method takes as input a filename and will transfer its content inside a String variable
      *
      * @param file A complete path to a given file
      * @return File content into a String
```

```
* @throws IOException
*/
private String updateString(String file) throws IOException {
String line;
StringBuilder stringBuilder = new StringBuilder();
BufferedReader reader = new BufferedReader( new FileReader (file));
while((line = reader.readLine()) != null ) {
stringBuilder.append(line);
stringBuilder.append(LS);
}
return stringBuilder.toString();
}
/**
* This method will transfer the content of a given String into a given filename
*
* @param fileName A complete path to a file to write
* @param fileContent The string content of the file to write
* @throws IOException
*/
private void storeString(String fileName,String fileContent) throws IOException {
BufferedWriter writer = new BufferedWriter(new FileWriter(fileName));
writer.write(fileContent);
writer.close();
}
/**
* This enumerated type contains all JSP input items to be managed
* by the getInputForm method
*
* @see getInputForm
*/
void getInputForm(ActionRequest request, AppInput appInput) {
// Retrieve from the input form the given application values
appInput.inputValue = (String) request.getParameter("inputValue");
appInput.inputValue2 = (String) request.getParameter("inputValue2");
appInput.inputValue3 = (String) request.getParameter("inputValue3");
appInput.inputValue4 = (String) request.getParameter("inputValue4");
appInput.jobIdentifier = (String) request.getParameter("JobIdentifier");
// Show into the log the taken inputs
System.out.println(
LS + "Taken input parameters:"
+ LS + "-----------------------"
+ LS + "myValue: '" + appInput.inputValue + "'"
+ LS + "myValue2: '" + appInput.inputValue2 + "'"
+ LS + "myValue3: '" + appInput.inputValue3 + "'"
+ LS + "myValue4: '" + appInput.inputValue4 + "'"
+ LS + "JobIdentifier: '" + appInput.jobIdentifier + "'"
+ LS);
} // getInputForm
/**private enum inputControlsIds {
* file_inputFile // Input file textarea
* ,inputFile // Input file input file
* ,JobIdentifier // User defined Job identifier
};
*/
/**
* This method manages the user input fields managing two cases
* distinguished by the type of the input <form ... statement
* The use of upload file controls needs the use of "multipart/form-data"
* while the else condition of the isMultipartContent check manages the
* standard input case. The multipart content needs a manual processing of
* all <form items
* All form' input items are identified by the 'name' input property
* inside the jsp file
*
```

```
* @param request ActionRequest instance (processAction)
* @param appInput AppInput instance storing the jobSubmission data
*/
/**void getInputForm(ActionRequest request,AppInput appInput) {
*if (PortletFileUpload.isMultipartContent(request))
* try {
* FileItemFactory factory = new DiskFileItemFactory();
* PortletFileUpload upload = new PortletFileUpload( factory );
* List items = upload.parseRequest(request);
* File repositoryPath = new File("/tmp");
* DiskFileItemFactory diskFileItemFactory = new DiskFileItemFactory();
* diskFileItemFactory.setRepository(repositoryPath);
* Iterator iter = items.iterator();
* String logstring="";
* while (iter.hasNext()) {
* FileItem item = (FileItem)iter.next();
* String fieldName =item.getFieldName();
* String fileName =item.getName();
* String contentType=item.getContentType();
* boolean isInMemory =item.isInMemory();
* long sizeInBytes=item.getSize();
* // Prepare a log string with field list
* logstring+=LS+"field name: '"+fieldName+"' - '"+item.getString()+"'";
* switch(inputControlsIds.valueOf(fieldName)) {
* case file_inputFile:
* appInput.inputFileName=item.getString();
* processInputFile(item,appInput);
* break;
* case inputFile:
* appInput.inputFileText=item.getString();
* break;
* case JobIdentifier:
```

```
* appInput.jobIdentifier=item.getString();
* break;
* default:
* _log.warn("Unhandled input field: '"+fieldName+"' - '"+item.getString()+"'");
* } // switch fieldName
* } // while iter.hasNext()
* _log.info(
* LS+"Reporting"
* +LS+"---------"
* +LS+logstring
* +LS);
* } // try
* catch (Exception e) {
* _log.info("Caught exception while processing files to upload: '"+e.toString()+"'");
* }
* // The input form do not use the "multipart/form-data"
* else {
* // Retrieve from the input form the given application values
* appInput.inputFileName=(String)request.getParameter("file_inputFile");
* appInput.inputFileText=(String)request.getParameter("inputFile");
* appInput.jobIdentifier=(String)request.getParameter("JobIdentifier");
* } // ! isMultipartContent
* // Show into the log the taken inputs
* _log.info(
* LS+"Taken input parameters:"
* +LS+"-----------------------"
* +LS+"inputFileName: '"+appInput.inputFileName+"'"
* +LS+"inputFileText: '"+appInput.inputFileText+"'"
* +LS+"jobIdentifier: '"+appInput.jobIdentifier+"'"
* +LS);
} // getInputForm
*/
```

```
/**
* This method is called when the user specifies a input file to upload
* The file will be saved first into /tmp directory and then its content
* stored into the corresponding String variable
* Before to submit the job the String value will be stored in the
* proper job inputSandbox file
*
* @param item
* @param appInput AppInput instance storing the jobSubmission data
*/
/**void processInputFile(FileItem item,AppInput appInput) {
* // Determine the filename
* String fileName = item.getName();
* if(!fileName.equals("")) {
* // Determine the fieldName
* String fieldName = item.getFieldName();
*
* // Create a filename for the uploaded file
* String theNewFileName = "/tmp/"
* +appInput.timestamp
* +"_"
* +appInput.username
* +"_"
* +fileName;
* File uploadedFile = new File(theNewFileName);
* _log.info("Uploading file: '"+fileName+"' into '"+theNewFileName+"'");
* try {
* item.write(uploadedFile);
* }
* catch (Exception e) {
* _log.error("Caught exception while uploading file: 'file_inputFile'");
* }
* // File content has to be inserted into a String variables:
* // inputFileName -> inputFileText
* try {
* if(fieldName.equals("file_inputFile"))
* appInput.inputFileText=updateString(theNewFileName);
* // Other params can be added as below ...
* //else if(fieldName.equals("..."))
* // ...=updateString(theNewFileName);
* else { // Never happens
* }
* }
* catch (Exception e) {
* _log.error("Caught exception while processing strings: '"+e.toString()+"'");
* }
* } // if
} // processInputFile
*/
/**
* Before to submit the job this method creates the inputSandbox files
* starting from users' input fields (textareas or uploaded files)
*
* @param appInput AppInput instance storing the jobSubmission data
*/
/**void updateFiles(AppInput appInput) {
* // First of all remove all possible ^Ms from Strings
* appInput.inputFileText=appInput.inputFileText.replaceAll("\r","");
* // Now save string content into files
* // This must be done for each input sandbox file
* try {
* appInput.inputSandbox_inputFile="/tmp/"
* +appInput.timestamp
* +"_"
* +appInput.username
* +"_input_file.txt"
* ;
* FileWriter fwInput=new FileWriter(appInput.inputSandbox_inputFile);
* BufferedWriter bwInput = new BufferedWriter(fwInput);
* bwInput.write(appInput.inputFileText);
* bwInput.close();
* }
* catch (Exception e) {
* _log.error("Caught exception while creating inputSandbox files");
* }
} // updateFiles
*/
/**
* -- WARNING ------------------------------------------------------
* (DEPRECATED) This method will be left only for some future commits
* ----------------------------------------------------------------
* This method sends the job into the distributed infrastructure using
* the GridEngine methods
*
* @param appInput AppInput instance storing the jobSubmission data
*/
void __submitJob(AppInput appInput) {
// GridEngine' MultiInfrastructure job submission object
MultiInfrastructureJobSubmission miJobSubmission=null;
//
// Initialize the GridEngine Multi Infrastructure Job Submission object
//
// GridEngine uses two different kind of constructors. The constructor
// taking void type as argument is used for production environments, while
// the constructor taking SciGwyUserTrackingDB parameters is normally used
// for development purposes. In order to switch-on the production constructor
// just set to empty strings the following portlet init parameters:
// sciGwyUserTrackingDB_Hostname
// sciGwyUserTrackingDB_Username
// sciGwyUserTrackingDB_Password
// sciGwyUserTrackingDB_Database
//
if(null != appPreferences.getSciGwyUserTrackingDB_Hostname()
&& !appPreferences.getSciGwyUserTrackingDB_Hostname().equals("")
&& null != appPreferences.getSciGwyUserTrackingDB_Username()
&& !appPreferences.getSciGwyUserTrackingDB_Username().equals("")
&& null != appPreferences.getSciGwyUserTrackingDB_Password()
&& !appPreferences.getSciGwyUserTrackingDB_Password().equals("")
&& null != appPreferences.getSciGwyUserTrackingDB_Database()
&& !appPreferences.getSciGwyUserTrackingDB_Database().equals("")
) {
String arg1="jdbc:mysql://" + appPreferences.getSciGwyUserTrackingDB_Hostname() +
"/" + appPreferences.getSciGwyUserTrackingDB_Database();
String arg2=appPreferences.getSciGwyUserTrackingDB_Username();
String arg3=appPreferences.getSciGwyUserTrackingDB_Password();
miJobSubmission = new MultiInfrastructureJobSubmission(arg1,arg2,arg3);
_log.info("MultiInfrastructureJobSubmission [DEVEL]\n"
+LS+" Arg1: '" + arg1 + "'"
+LS+" Arg2: '" + arg2 + "'"
+LS+" Arg3: '" + arg3 + "'"
);
}
else {
miJobSubmission = new MultiInfrastructureJobSubmission();
_log.info("MultiInfrastructureJobSubmission [PROD]");
}
// Assigns all enabled infrastructures
InfrastructureInfo[] infrastructuresInfo=appPreferences.getEnabledInfrastructures();
```

```
        for(int i=0; i<infrastructuresInfo.length; i++) {
        _log.info("Adding infrastructure #"+(i+1)
        +" - Name: '"+infrastructuresInfo[i].getName()+"'"+LS);
        miJobSubmission.addInfrastructure(infrastructuresInfo[i]);
        }
        // Check the enabled infrastructures
        if(infrastructuresInfo.length > 0) {
        // Application Id
        int applicationId=Integer.parseInt(appPreferences.getGridOperationId());
        // Grid Engine' UserTraking needs the portal IP address
        String portalIPAddress="";
        try {
        InetAddress addr = InetAddress.getLocalHost();
        byte[] ipAddr=addr.getAddress();
        portalIPAddress= ""+(short)(ipAddr[0]&0xff)
        +":"+(short)(ipAddr[1]&0xff)
        +":"+(short)(ipAddr[2]&0xff)
        +":"+(short)(ipAddr[3]&0xff);
        }
        catch(Exception e) {
        _log.error("Unable to get the portal IP address");
  }
      // Job details
      String executable="/bin/sh"; // Application executable
      String arguments =appPreferences.getPilotScript(); // executable' arguments
      String outputPath="/tmp/"; // Output Path
      String outputFile="myRepast-infection-Output.txt"; // Distributed application standard output
      String errorFile ="myRepast-infection-Error.txt"; // Distrubuted application standard error
      String appFile ="myRepast-infection-Files.tar.gz"; // Hostname output files (created by the pilot script)
      // InputSandbox (string with comma separated list of file names)
      /**String inputSandbox=appServerPath+"WEB-INF/job/" //
      * +appPreferences.getPilotScript() // pilot script
```

```
* +","+appInput.inputSandbox_inputFile // input file
*
*;
*/
// OutputSandbox (string with comma separated list of file names)
String outputSandbox=appFile; // Output file
// Take care of job requirements
// More requirements can be specified in the preference value 'jobRequirements'
// separating each requirement by the ';' character
String jdlRequirements[] = appPreferences.getJobRequirements().split(";");
int numRequirements=0;
for(int i=0; i<jdlRequirements.length; i++) {
if(!jdlRequirements[i].equals("")) {
jdlRequirements[numRequirements] = "JDLRequirements=("+jdlRequirements[i]+")";
numRequirements++;
_log.info("Requirement["+i+"]='"+jdlRequirements[i]+"'");
}
} // for each jobRequirement
// Other job initialization settings
miJobSubmission.setExecutable ( executable); // Specify the executeable
miJobSubmission.setArguments ( arguments); // Specify the application' arguments
miJobSubmission.setOutputPath ( outputPath); // Specify the output directory
miJobSubmission.setOutputFiles(outputSandbox); // Setup output files (OutputSandbox)
miJobSubmission.setJobOutput ( outputFile); // Specify the std-outputr file
miJobSubmission.setJobError ( errorFile); // Specify the std-error file
/*if( null != inputSandbox // Setup input files (InputSandbox) avoiding empty inputSandboxes
* && inputSandbox.length() > 0)
* miJobSubmission.setInputFiles(inputSandbox);
*if(numRequirements>0) // Setup the JDL requirements
* miJobSubmission.setJDLRequirements(jdlRequirements);
*/
// Submit Job
```

```
miJobSubmission.submitJobAsync(appInput.username, portalIPAddress, applicationId,
appInput.jobIdentifier);
// Show log
// View jobSubmission details in the log
_log.info(
LS+"JobSent"
+LS+"-------"
+LS+"Portal address: '"+portalIPAddress+"'"
+LS+"Executable : '"+executable +"'"
+LS+"Arguments : '"+arguments +"'"
+LS+"Output path : '"+outputPath +"'"
+LS+"Output sandbox: '"+outputSandbox +"'"
+LS+"Ouput file : '"+outputFile +"'"
+LS+"Error file : '"+errorFile +"'"
//+LS+"Input sandbox : '"+inputSandbox +"'"
+LS); // _log.info
} // numInfra > 0
else {
_log.warn(
LS+"There are no enough enabled infrastructures!"
+LS+"It is impossible to send any job"
+LS+"Configure the application preferences in order to setup"
+LS+"or enable at least one infrastructure."
+LS);
} // numInfra == 0
} // __submitJob
/**
* -- WARNING -----------------------------------------------------
* (DEPRECATED) This method will be left only for some future commits
* ---------------------------------------------------------------
* This method sends the job into the distributed infrastructure using
* the GridEngine methods
```

```
*
* @param appInput AppInput instance storing the jobSubmission data
*/
void submitJob(AppInput appInput) {
// Job details
String executable="/bin/sh"; // Executable
String arguments =appPreferences.getPilotScript() + // Script
" " + appInput.inputValue + // Arguments
" " + appInput.inputValue2 +
" " + appInput.inputValue3 +
" " + appInput.inputValue4 ;
String outputPath="/tmp/"; // Output Path
String outputFile="myRepast-infection-Output.txt"; // Distributed application standard output
String errorFile ="myRepast-infection-Error.txt"; // Distrubuted application standard error
String appFile ="myRepast-infection-Files.tar.gz"; // Hostname output files (created by the pilot script)
// InputSandbox (string with comma separated list of file names)
String inputSandbox=appServerPath+"WEB-INF/job/pilot_script.sh"; //
// +appPreferences.getPilotScript(); // pilot script
// +","+appInput.inputSandbox_inputFile; // input file;
// OutputSandbox (string with comma separated list of file names)
String outputSandbox=appFile; // Output file
// Take care of job requirements
// More requirements can be specified in the preference value 'jobRequirements'
// separating each requirement by the ';' character
// The loop prepares a string array with GridEngine/JSAGA compliant requirements
String jdlRequirements[] = appPreferences.getJobRequirements().split(";");
int numRequirements=0;
for(int i=0; i<jdlRequirements.length; i++) {
if(!jdlRequirements[i].equals("")) {
jdlRequirements[numRequirements] = "JDLRequirements=("+jdlRequirements[i]+")";
numRequirements++;
_log.info("Requirement["+i+"]='"+jdlRequirements[i]+"'");
}
} // for each jobRequirement
// Prepare the GridEngine job description
GEJobDescription jobDesc = new GEJobDescription();
jobDesc.setExecutable ( executable); // Specify the executeable
jobDesc.setArguments ( arguments); // Specify the application' arguments
jobDesc.setOutputPath ( outputPath); // Specify the output directory
jobDesc.setOutput ( outputFile); // Specify the std-output file
jobDesc.setError ( errorFile); // Specify the std-error file
jobDesc.setOutputFiles(outputSandbox); // Setup output files (OutputSandbox) (*)
jobDesc.setInputFiles ( inputSandbox); // Setut input files (InputSandbox)
// GridEngine' MultiInfrastructure job submission object
MultiInfrastructureJobSubmission miJobSubmission=null;
//
// Initialize the GridEngine Multi Infrastructure Job Submission object
//
// GridEngine uses two different kind of constructors. The constructor
// taking no database arguments is used for production environments, while
// the constructor taking SciGwyUserTrackingDB parameters is normally used
// for development purposes. In order to switch-on the production constructor
// just set to empty strings the following portlet init parameters or form
// the portlet preferences:
// sciGwyUserTrackingDB_Hostname
// sciGwyUserTrackingDB_Username
// sciGwyUserTrackingDB_Password
// sciGwyUserTrackingDB_Database
//
if(null != appPreferences.getSciGwyUserTrackingDB_Hostname()
&& !appPreferences.getSciGwyUserTrackingDB_Hostname().equals("")
&& null != appPreferences.getSciGwyUserTrackingDB_Username()
&& !appPreferences.getSciGwyUserTrackingDB_Username().equals("")
&& null != appPreferences.getSciGwyUserTrackingDB_Password()
&& !appPreferences.getSciGwyUserTrackingDB_Password().equals("")
&& null != appPreferences.getSciGwyUserTrackingDB_Database()
&& !appPreferences.getSciGwyUserTrackingDB_Database().equals("")
) {
String DBNM="jdbc:mysql://" + appPreferences.getSciGwyUserTrackingDB_Hostname() +
"/" + appPreferences.getSciGwyUserTrackingDB_Database();
String DBUS=appPreferences.getSciGwyUserTrackingDB_Username();
String DBPW=appPreferences.getSciGwyUserTrackingDB_Password();
miJobSubmission = new MultiInfrastructureJobSubmission(DBNM,DBUS,DBPW,jobDesc);
_log.info("MultiInfrastructureJobSubmission [DEVEL]\n"
+LS+" DBNM: '" + DBNM + "'"
+LS+" DBUS: '" + DBUS + "'"
+LS+" DBPW: '" + DBPW + "'"
);
}
else {
miJobSubmission = new MultiInfrastructureJobSubmission(jobDesc);
_log.info("MultiInfrastructureJobSubmission [PROD]");
}
// Assigns now all enabled infrastructures
InfrastructureInfo[] infrastructuresInfo=appPreferences.getEnabledInfrastructures();
for(int i=0; i<infrastructuresInfo.length; i++) {
_log.info("Adding infrastructure #"+(i+1)
+" - Name: '"+infrastructuresInfo[i].getName()+"'"+LS);
miJobSubmission.addInfrastructure(infrastructuresInfo[i]);
}
// Check the enabled infrastructures
if(infrastructuresInfo.length > 0) {
// GridOperations' Application Id
int applicationId=Integer.parseInt(appPreferences.getGridOperationId());
// Grid Engine' UserTraking needs the portal IP address
String portalIPAddress="";
```

```
try {
InetAddress addr = InetAddress.getLocalHost();
byte[] ipAddr=addr.getAddress();
portalIPAddress= ""+(short)(ipAddr[0]&0xff)
+":"+(short)(ipAddr[1]&0xff)
+":"+(short)(ipAddr[2]&0xff)
+":"+(short)(ipAddr[3]&0xff);
}
catch(Exception e) {
_log.error("Unable to get the portal IP address");
}
// Setup job requirements
if(numRequirements>0)
miJobSubmission.setJDLRequirements(jdlRequirements);
// Ready now to submit the Job
miJobSubmission.submitJobAsync(miJobSubmission.getInfrastructure(), appInput.username
,portalIPAddress
,applicationId
,appInput.jobIdentifier
);
// Show log
// View jobSubmission details in the log
_log.info(
LS+"JobSent"
+LS+"-------"
+LS+"Portal address: '"+portalIPAddress+"'"
+LS+"Executable : '"+executable +"'"
+LS+"Arguments : '"+arguments +"'"
+LS+"Output path : '"+outputPath +"'"
+LS+"Output sandbox: '"+outputSandbox +"'"
+LS+"Ouput file : '"+outputFile +"'"
+LS+"Error file : '"+errorFile +"'"
// +LS+"Input sandbox : '"+inputSandbox +"'"
+LS); // _log.info
} // numInfra > 0
else {
_log.warn(
LS+"There are no enough enabled infrastructures!"
+LS+"It is impossible to send any job"
+LS+"Configure the application preferences in order to setup"
+LS+"or enable at least one infrastructure."
+LS);
} // numInfra == 0
} // submitJob
} // myRepast_infection_portletcic
```

## B2 Sample Infection Model Portlet JSP Pages

```
<portlet:defineObj
ects />
                <%
                // Gets the current timestamp
                java.util.Date date = new java.util.Date();
                %>
                <%
                // Below the descriptive area of the REPAST simulation web form
                %>
                <table>
                <tr>
                <td valign="top">
                <img align="left" style="padding:10px 10px;"
                src="<%=renderRequest.getContextPath()%>/images/Repast_logo_100h.p
                ng" />
                </td>
                <td align="justify">
```

```
The aim of this demonstration model is to show how a science gateway
could support the study of the spread of disease or infections in a
population.
The <b>Repast-infection-model</b> is implemented in <b>Repast
Simphony</b>. The aim of this example model is to study the behaviour of
infections
with an annual outbreak according to several input parameters.
</td>
</tr>
<tr>
<td colspan="2" align="justify">
The parameters are:
<ul>
<li><b>Simulation Period</b> - specifies how many years the simulation
will run for.</li>
<li><b>Recovered Count</b> - specifies the initial healthy population.
Healthy population have immunity and cannot be infected immediately.
However, after a number of contacts with infected population, they lose
their immunity and become susceptible to infection.</li>
<li><b>Infected Count</b> - specifies the initial infected population.
Infected population can infect susceptible population upon contacting
them.
They recover after a period of time and become healthy.</li>
<li><b>Susceptible Count</b> - specifies initial susceptible population.
Susceptible population can be infected when contacted by infected
population. We assume that if more than one susceptible agent is in the
proximity of an infected agent, only one will be infected.</li>
</ul>
The output of the simulation is the amount of each population (Recovered,
Infected and Susceptible) over time.
</td>
</tr>
```

```
<tr>
<td colspan="2" align="justify"><br/>
<b>How to use the Infection Model</b>
<ol>
<li>Select an experiment to run from the drop down box (in a full
application users would be able to enter their own parameters).</li>
<li>Note the Simulation Identifier. You will need this to identify the results
from the experiment that you are about to run.</li>
<li> Press <b>Submit</b>. This will submit the experiment to the
computers of the e-Infrastructure that you are using.</li>
<li>Go to <b>MyWorkSpace</b> (menu bar above). Select
<b>MyJobs</b>. Your current experiments will be listed here.
When the job is complete then you can download by clicking on the icon
on the right of the job. </li>
<li>Unzip your results file. The download is in tgz zipped format. Use a tool
like 7-zip (<a href="http://www.7-zip.com">www.7-zip.com</a>) to unzip
the file.</li>
<li>Each results file will contact <b>output{timestamp}.csv</b>. This
contains how each population varies over time.
To visualise it, select <b>Visualise</b> from the left hand side menu and
upload the text file to produce a graph of population variance over
time.</li>
</ol>
</td>
</tr>
<tr>
<td colspan="2" align="justify">
Please fill the following form and then press the <b>'SUBMIT'</b> button
to launch this application.<br>
Requested inputs are:
</td>
</tr>
```

```
                        </table>
                        <%
                        // Below the application submission web form
                        //
                        // The <form> tag contains a portlet parameter value called 'PortletStatus'
                        the value of this item
                        // will be read by the processAction portlet method which then assigns a
                        proper view mode before
                        // the call to the doView method.
                        // PortletStatus values can range accordingly to values defined into Enum
                        type: Actions
                        // The processAction method will assign a view mode accordingly to the
                        values defined into
                        // the Enum type: Views. This value will be assigned calling the function:
                        setRenderParameter
                        //
                        %>
                        <br />
                        <br />
                        <center>
                        <form action="<portlet:actionURL portletMode="view"><portlet:param
                        name="PortletStatus" value="ACTION_SUBMIT"/></portlet:actionURL>"
                        method="post">
                        <table>
    <tr
    >
        <td style="width: 40%"><b>Select
        experiment</b></td>
        <td >
        <select id="experiment"
        onchange="setSimulationValues()" style="width:
        95%;float: right;" >
```

```
<option value="-1">Please select an experiment
...</option>
<option value="0">Experiment 1</option>
<option value="1">Experiment 2</option>
<option value="2">Experiment 3</option>
<option value="3">Experiment 4</option>
<option value="4">Experiment 5</option>
</select>
</td>
</tr>
<tr>
<td colspan="2"><hr/></td>
<tr>
<th colspan="2"><center>Simulation
Parameters</center></th>
</tr>
<tr>
<td style="width: 40%"><b>Simulation Period
(years)</b></td>
<td id="simulationPeriodId" style="width: 60%">
<input type="text" id="inputValueId"
name="inputValue" readonly="true" style="width:
95%;float: right;"/>
</td>
</tr>
<tr>
<td style="width: 40%"><b>Recovered
Count</b></td>
<td id="recoveredCountId" style="width: 60%">
<input type="text" id="inputValueId2"
name="inputValue2" readonly="true" style="width:
95%;float: right;"/>
```

```
</td>
</tr>
<tr>
<td style="width: 40%"><b>Infected Count</b></td>
<td id="infectedCountId" style="width: 60%">
<input type="text" id="inputValueId3"
name="inputValue3" readonly="true" style="width:
95%;float: right;"/>
</td>
</tr>
<tr>
<td style="width: 40%"><b>Susceptible
Count</b></td>
<td id="susceptibleCountId" style="width: 60%">
<input type="text" id="inputValueId4"
name="inputValue4" readonly="true" style="width:
95%;float: right;"/>
</td>
</tr>
<tr>
<td style="width: 40%"><b>Simulation
identifier</b></td>
<td><input type="text" id="jobIdentifierId"
name="JobIdentifier" placeholder="Repast simulation..."
style="width: 95%;float: right;" readonly="true"/></td>
</tr>
<tr>
<td colspan="3"><hr/></td>
</tr>
<tr>
<td style="width: 50%">
<center>
```

```
<input type="button" value="SUBMIT"
onClick="preSubmit()" />
</center>
</td>
<td colspan="2">
<center>
<input type="reset" value="RESET"/>
</center>
</td>
</tr>
</table>
</form>
</center>
<script type="text/javascript">
var experiments = [
{"simulationPeriod": 20, "recoveredCount": 0,
"infectedCount": 20, "susceptibleCount": 1500},
{"simulationPeriod": 20, "recoveredCount": 0,
"infectedCount": 20, "susceptibleCount": 2000},
{"simulationPeriod": 20, "recoveredCount": 0,
"infectedCount": 10, "susceptibleCount": 1500},
{"simulationPeriod": 20, "recoveredCount": 0,
"infectedCount": 10, "susceptibleCount": 2000},
{"simulationPeriod": 20, "recoveredCount": 0,
"infectedCount": 10, "susceptibleCount": 3000}
];
function preSubmit() {
var jobIdentifier =
document.getElementById('jobIdentifierId');
var state_jobIdentifier = false;
var inputValue =
document.getElementById('inputValueId');
```

```
var inputValue2 =
document.getElementById('inputValueId2');
var inputValue3 =
document.getElementById('inputValueId3');
var inputValue4 =
document.getElementById('inputValueId4');
var missingFields = "";
if (inputValue.value === "")
missingFields += "\nSimulation Period";
if (inputValue2.value === "")
missingFields += "\nRecovered Count";
if (inputValue3.value === "")
missingFields += "\nInfected Count";
if (inputValue4.value === "")
missingFields += "\nSusceptible Count";
if (jobIdentifier.value === "")
state_jobIdentifier = true;
if (state_jobIdentifier)
missingFields += "\nJob identifier";
if (missingFields === "") {
// alert("Ready to submit");
document.forms[0].submit();
} else {
alert("You cannot send an inconsistent job
submission!\n\nPlease select an experiment.");
}
}
function setSimulationValues() {
var x = document.getElementById("experiment").value;
var experimentDropDown =
document.getElementById("experiment");
//
```

```
document.getElementById("simulationPeriodId").innerHT
ML = experiments[x].simulationPeriod;
if (x !== "-1") {
document.getElementById("inputValueId").value =
experiments[x].simulationPeriod;
document.getElementById("inputValueId2").value =
experiments[x].recoveredCount;
document.getElementById("inputValueId3").value =
experiments[x].infectedCount;
document.getElementById("inputValueId4").value =
experiments[x].susceptibleCount;
var currentTime = new Date();
var jobIdentifier =
document.getElementById('jobIdentifierId');
jobIdentifier.value =
experimentDropDown.options[experimentDropDown.sel
ectedIndex].innerHTML;
jobIdentifier.value += " " + currentTime.getDate() + "/"
+ (currentTime.getMonth() + 1) + "/" +
currentTime.getFullYear() + " - " +
currentTime.getHours() + ":" + currentTime.getMinutes()
+ ":" + currentTime.getSeconds();
} else {
alert("Please select a valid experiment.");
}
}
function setSimulationPeriod() {
var x =
document.getElementById("simulationPeriod").value;
document.getElementById("inputValueId").value = x;
}
function setRecoveredCount() {
```

```
            var x =
            document.getElementById("recoveredCount").value;
            document.getElementById("inputValueId2").value = x;
            }
            function setInfectedCount() {
            var x =
            document.getElementById("infectedCount").value;
            document.getElementById("inputValueId3").value = x;
            }
            function setSusceptibleCount() {
            var x =
            document.getElementById("susceptibleCount").value;
            document.getElementById("inputValueId4").value = x;
            }
            </script>

   <%@ taglib
   uri="http://java.sun.com/
   portlet_2_0"
   prefix="portlet" %>
                              <portlet:defineObjects />
                              <%//
                              // Application Submission page
                              //
                              //
                              // The portlet code assigns the jobIdentifier as
                              input parameter for this jsp file
                              //
                              %>
                              <jsp:useBean id="jobIdentifier"
                              class="java.lang.String" scope="request"/>
                              <%
```

```
// Below the submission web form
//
// It only have a button that will show the input
form again for a new job submission
%>
<table>
<tr>
<td valign="top"><img align="left"
style="padding:10px 10px;"
src="<%=renderRequest.getContextPath()%>/ima
ges/Repast_logo_100h.png" /></td>
<td>
Your job has been <b>successfully</b>
submitted; you may get reference to it with
identifier:<br>
<b><%= jobIdentifier %></b><br>
Have a look on <a href="/my-jobs">MyJobs</a>
area to get more information about all your
submitted jobs.
</td>
</tr>
<tr>
<td align="center"><form
action="<portlet:actionURL
portletMode="view"><portlet:param
name="PortletStatus"
value="ACTION_INPUT"/></portlet:actionURL>"
method="post">
<input type="submit" value="Run a new
application"></form></td>
<td>Press the <b>Run a new application</b>
button to start another job submission</td>
```

```
</tr>
</table>
```

# Appendix C

## Code Fragments of the Main Java Classes

### C1. Applogger Class

```
private static final int TRACE_LEVEL=6;
private static final int DEBUG_LEVEL=5;
private static final int INFO_LEVEL=4;
private static final int WARN_LEVEL=3;
private static final int ERROR_LEVEL=2;
private static final int FATAL_LEVEL=1;
private static final int UNKNOWN_LEVEL=0;
```

### C2.Initialisation of the Application Preferences

```
        init()
        throws PortletException
        {
        appInitPreferences.setGridOperationDesc(""+getInitParameter(
        "gridOperationDesc"));
        appInitPreferences.setPortletVersion(""+getInitParameter("portletVers
        ion"));
        appInitPreferences.setLogLevel(""+getInitParameter("logLevel"));
        appInitPreferences.setNumInfrastructures(""+getInitParameter("numInfr
        astructures");
        appInitPreferences.setGridOperationId(""+getInitParameter("gridOperat
        ionId"));
```

### C3. Java code of the Agent-Based Simulation application

```
Class myRepast-infection-portlet extends GenericPortlets {
            Init (PortletConfig);
            processAction (ActionRequest, ActionResponse);
            Render (RenderRequest, RenderResponse);
            Destroy ();
            Do View (Request, Response);
            Do Edit (Request, Response);
            Do Help (Request, Response);
      }
```

### C4. The GridEngine Method

```
submitJob(AppInput appInput){
            MultiInfrastructureJobSubmission miJobSubmission=null;
            }
```

where the appInput is the AppInput instance that stores the jobSubmission data and the miJobSubmission is the GridEngine Multi Infrastructure Job Submission object

## C5. Portlet Parameters in Production Stage

```
sciGwyUserTrackingDB_Hostname
sciGwyUserTrackingDB_Username
sciGwyUserTrackingDB_Password
sciGwyUserTrackingDB_Database
```

## C6. Portlet Parameters in Development Stage

```
String arg1="jdbc:mysql://"
appPreferences.getSciGwyUserTrackingDB_Hostnsame()    +
appPreferences.getSciGwyUserTrackingDB_Database();
String arg2=appPreferences.getSciGwyUserTrackingDB_Username();
String arg3=appPreferences.getSciGwyUserTrackingDB_Password();
```

## C7. GridEngine Job Description

```
GEJobDescription jobDesc = new GEJobDescription();
            jobDesc.setExecutable (executable);
            jobDesc.setArguments (arguments);
            jobDesc.setOutputPath(outputPath);
            jobDesc.setOutput(outputFile);
            jobDesc.setError(errorFile);
            jobDesc.setOutputFiles(outputSandbox);
            jobDesc.setInputFiles(inputSandbox);
```

## C8. GridEngine Multi-infrastructure job Submission

```
            miJobSubmission.submitJobAsync
            (miJobSubmission.getInfrastructure(),
            appInput.username, portalIPAddress, applicationId,
            appInput.jobIdentifier);
```

## C9. Comand to send job to futureGatewayAPI

```
curl  -X  POST  -H  "Content-Type:  application/json"  -H  "Cache-Control:  no-
cache" -H "Postman-Token: db918854-b421-ed25-04ac-1de48ede6d10" -d '{

    "application":"8",

    "description":"WEKA Application",

    "input_files": [{

     "name":"weather.data"

    }],

    "arguments": [

     "-f weka.filters.unsupervised.attribute.ReplaceMissingValues",

      "weather.data",
      "weka.classifiers.decisiontree.J48"
    ],
    "output_files": [{
        "name":"weka-output.tar.gz"
    }]
```

### C10. FutureGateway Output

```
{
  "status": "WAITING",
  "application": "8",
  "date": "2016-11-03T18:40:59Z",
  "description": "WEKA Application",
  "output_files": [
    {
      "url": "file?path=&name=weka-output.tar.gz",
      "name": "weka-output.tar.gz"
    },
    {
      "url": "file?path=&name=weka-output.tar.gz",
      "name": "weka-output.tar.gz"
    },
    {
      "url": "file?path=&name=stdout",
      "name": "stdout"
    },
    {
      "url": "file?path=&name=stderr",
      "name": "stderr"
    }
  ],
  "_links": [
    {
      "href": "/v1.0/tasks/129",
      "rel": "self"
    },
    {
      "href": "/v1.0/tasks/129/input",
      "rel": "input"
    }
  ],
  "user": "adedeji.fabiyi@brunel.ac.uk",
  "input_files": [
    {
      "status": "READY",
      "name": "pilot_script.sh"
    },
    {
      "status": "NEEDED",
      "name": "weather.arff"
    }
  ],
  "id": "129",
  "arguments": [
    "-f weka.filters.unsupervised.attribute.ReplaceMissingValues",
    "weather.arff",
    "weka.classifiers.decisiontree.J48"
  ]
}
```

**C11. Submit job collection Object**

```
submitJobCollection(AppPreferences preferences, AppInput appInput,
InfrastructureInfo[] enabledInfrastructures){
ArrayList<GEJobDescription> descriptions = new
ArrayList<GEJobDescription>();
for (int i = 0; i < appInput.getTaskNumber(); i++) {
GEJobDescription description = new GEJobDescription();
description.setExecutable("/bin/sh");
description.setInputFiles(pilotScript);
description.setArguments(args);
description.setOutputPath("/tmp");
description.setOutput("output-" + i + ".txt");
description.setError("error-" + i + ".txt");
description.setOutputFiles("repast-infection-Files.tar.gz");
descriptions.add(description);
}}
```

### C12. GridEngine JobCollectionSubmission job submission object

```
JobCollectionSubmission tmpJobCollectionSubmission = null;
if (!preferences.isProductionEnviroment()) {
String DBNM = "jdbc:mysql://"
+ preferences.getSciGwyUserTrackingDB_Hostname() + "/"
+ preferences.getSciGwyUserTrackingDB_Database();
String DBUS = preferences.getSciGwyUserTrackingDB_Username();
String DBPW = preferences.getSciGwyUserTrackingDB_Password();
tmpJobCollectionSubmission = new JobCollectionSubmission(DBNM,
DBUS, DBPW, collection);
} else {
tmpJobCollectionSubmission = new JobCollectionSubmission(collection);
}
```

# Appendix D

## Agent State Diagram

### D1 Chain of Transmission

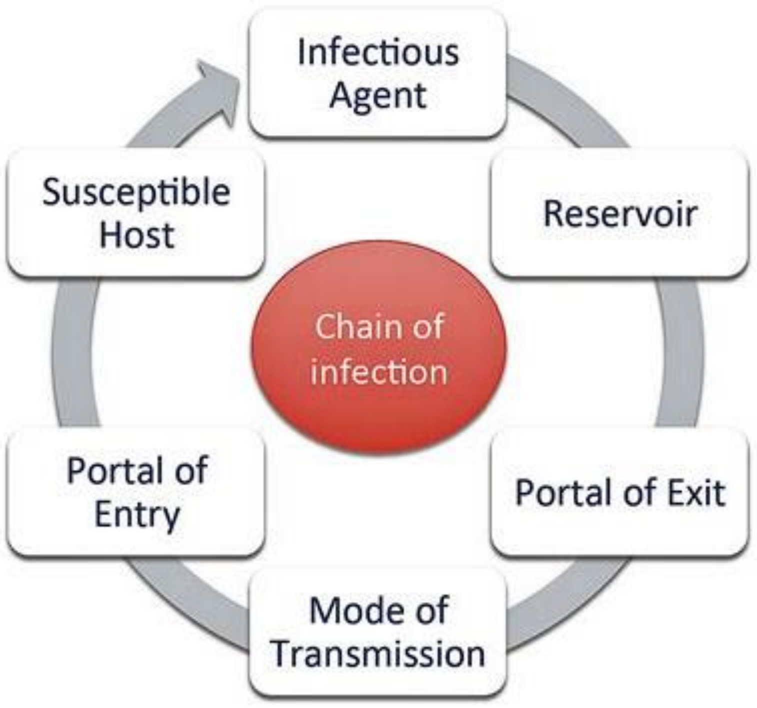


Figure D-1 Infection Model Chain of Transmission

**D2 Chain of Infection**

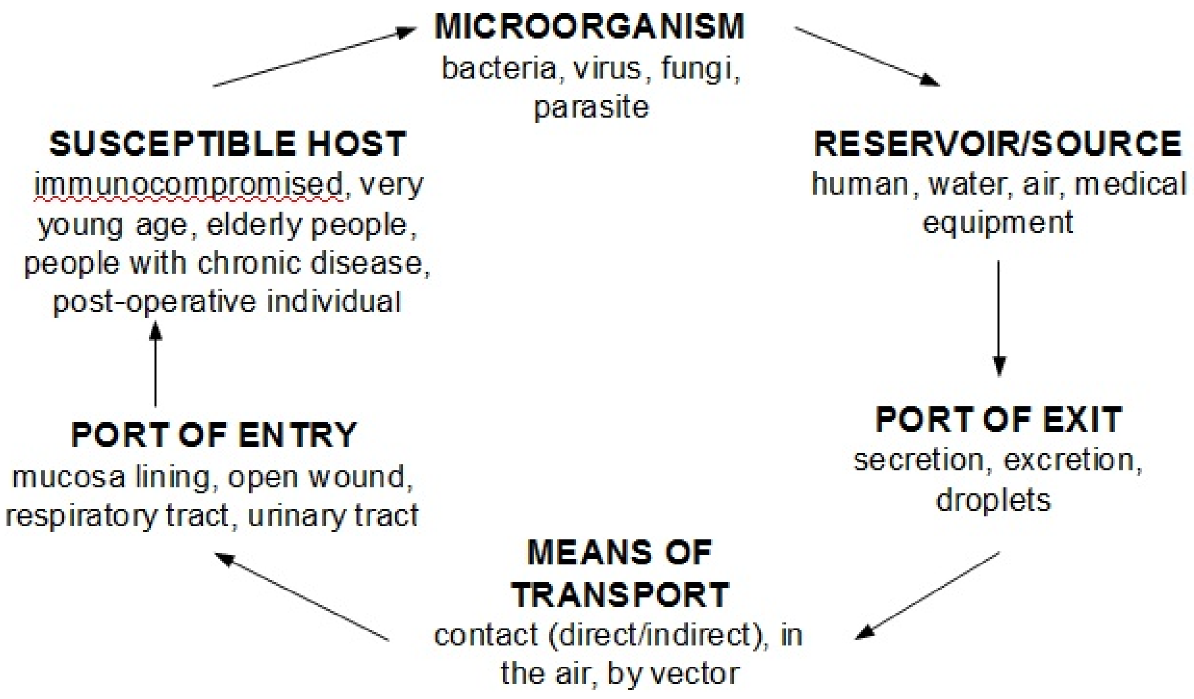

MICROORGANISM
bacteria, virus, fungi, parasite
RESERVOIR/SOURCE
human, water, air, medical equipment
PORT OF EXIT
secretion, excretion, droplets
MEANS OF TRANSPORT
contact (direct/indirect), in the air, by vector
PORT OF ENTRY
mucosa lining, open wound, respiratory tract, urinary tract
SUSCEPTIBLE HOST
immunocompromised, very young age, elderly people, people with chronic disease, post-operative individual

Figure D-2 Infection Model Chain of Infection 1

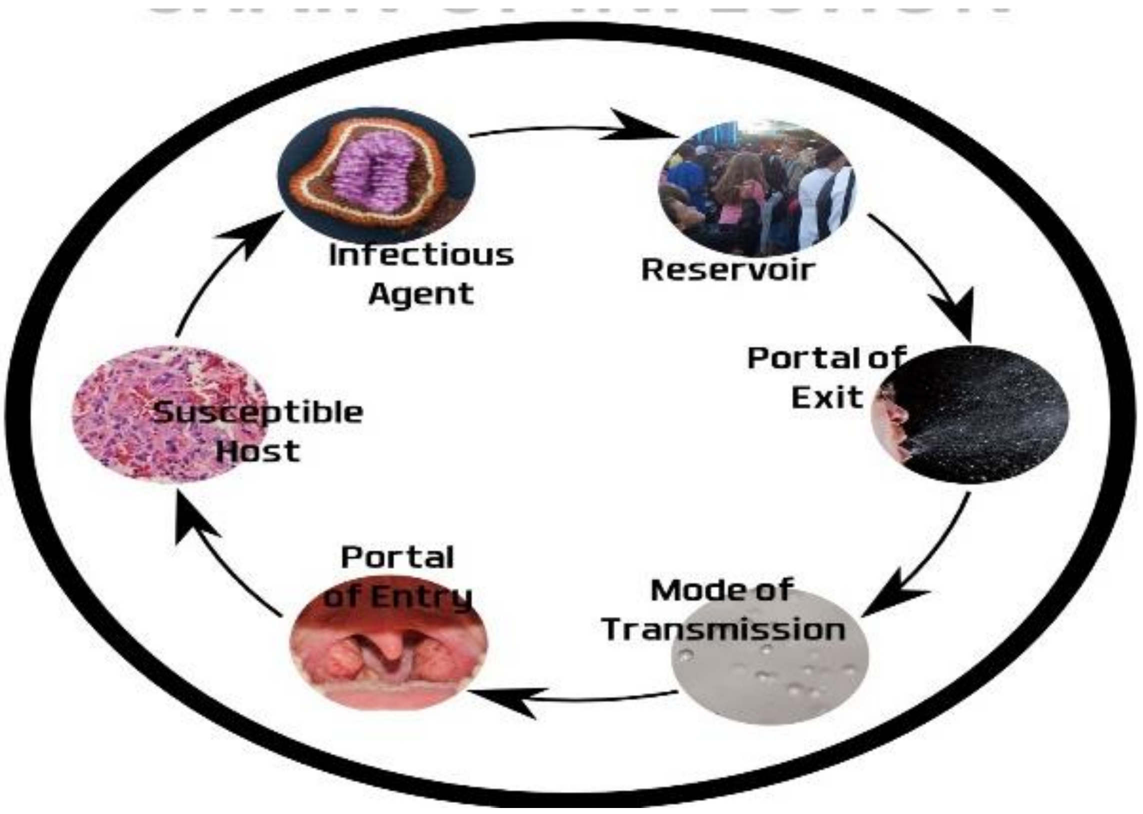
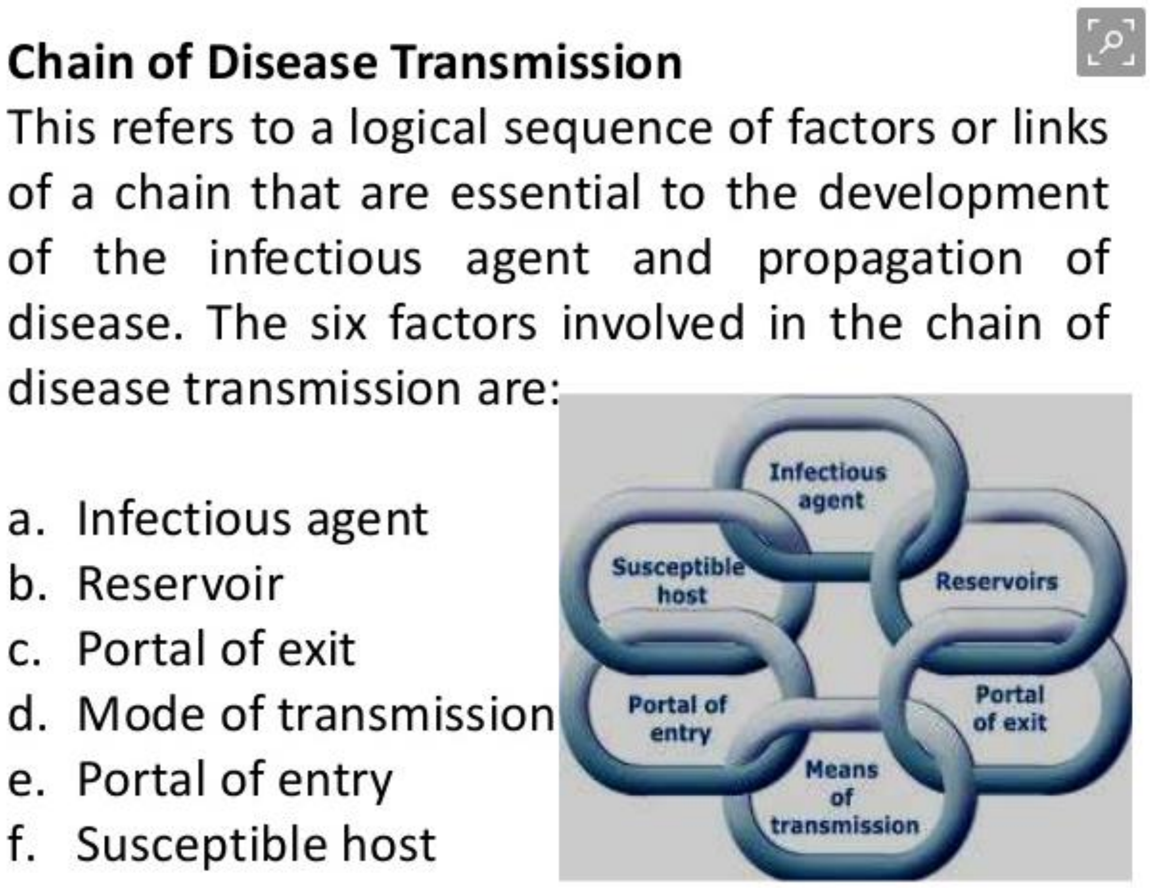


Figure D-3 Infection Model Chain of Infection 2

**D3. Chain of Disease Transmission**



**Chain of Disease Transmission**

This refers to a logical sequence of factors or links of a chain that are essential to the development of the infectious agent and propagation of disease. The six factors involved in the chain of disease transmission are:

a. Infectious agent
b. Reservoir
c. Portal of exit
d. Mode of transmission
e. Portal of entry
f. Susceptible host

Infectious agent
Reservoirs
Portal of exit
Means of transmission
Portal of entry
Susceptible host

Figure D-4 Infection Model Chain of Disease Transmission

**D4. Modes of Transmission**

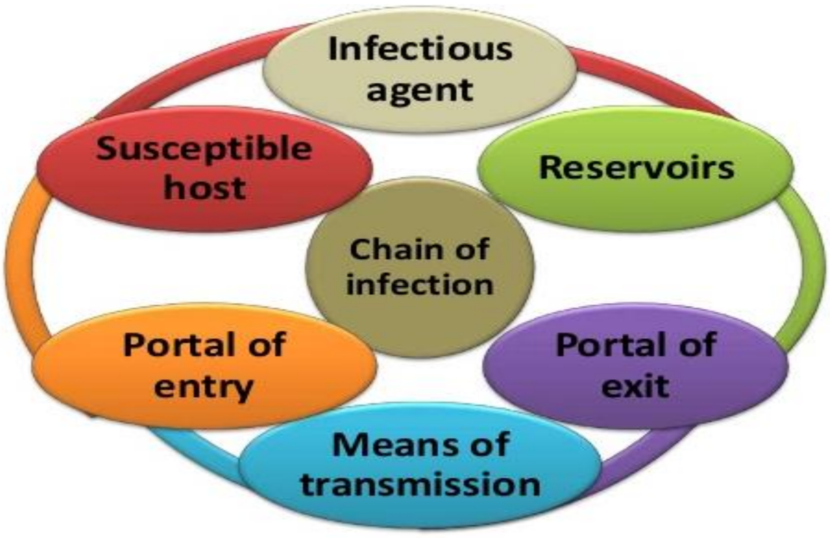


Figure D-5 Infection Model Modes of Transmission

# Appendix E

## Diagrams to show interaction between classes and methods

### E1 Process Action, Render and Destroy methods

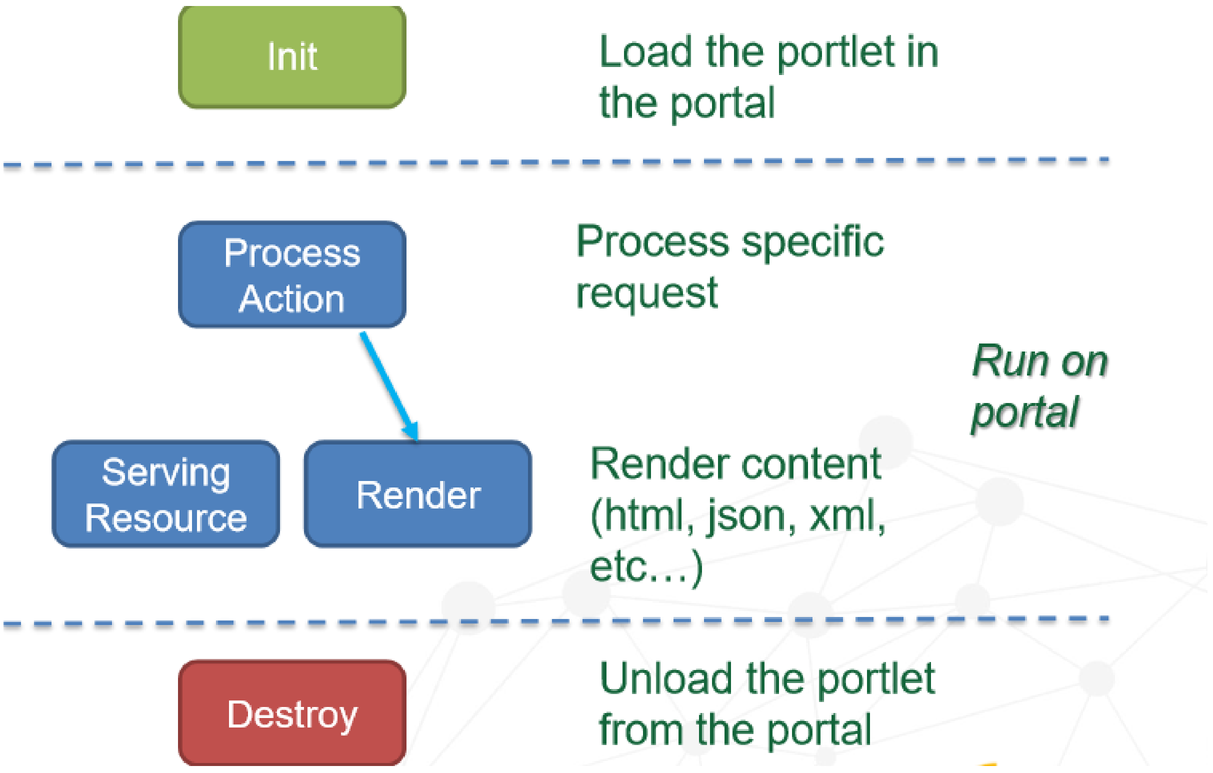

Init
Load the portlet in the portal
Process Action
Process specific request
Run on portal
Serving Resource
Render
Render content (html, json, xml, etc…)
Destroy
Unload the portlet from the portal

Figure E-1 Interaction between classes and Methods

**E2.Phases: Load a Page**

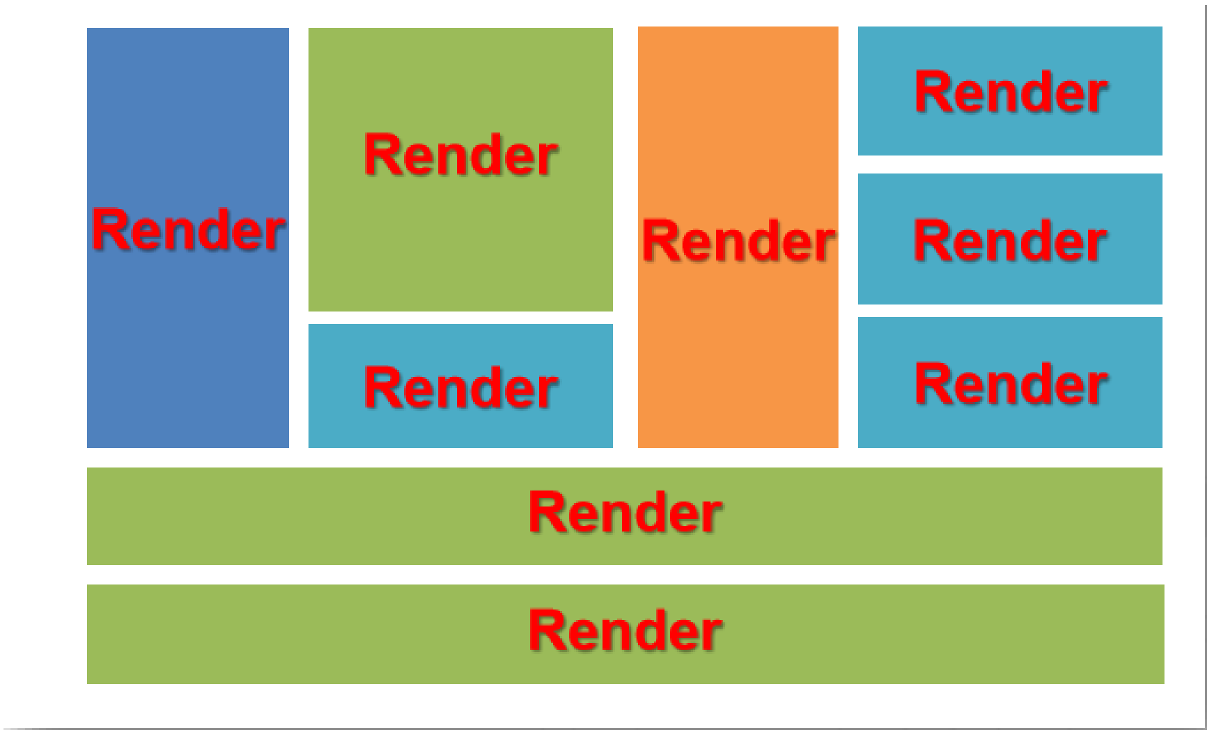


Figure E-2 Load a Portlet Page

**E3 Submit a form**

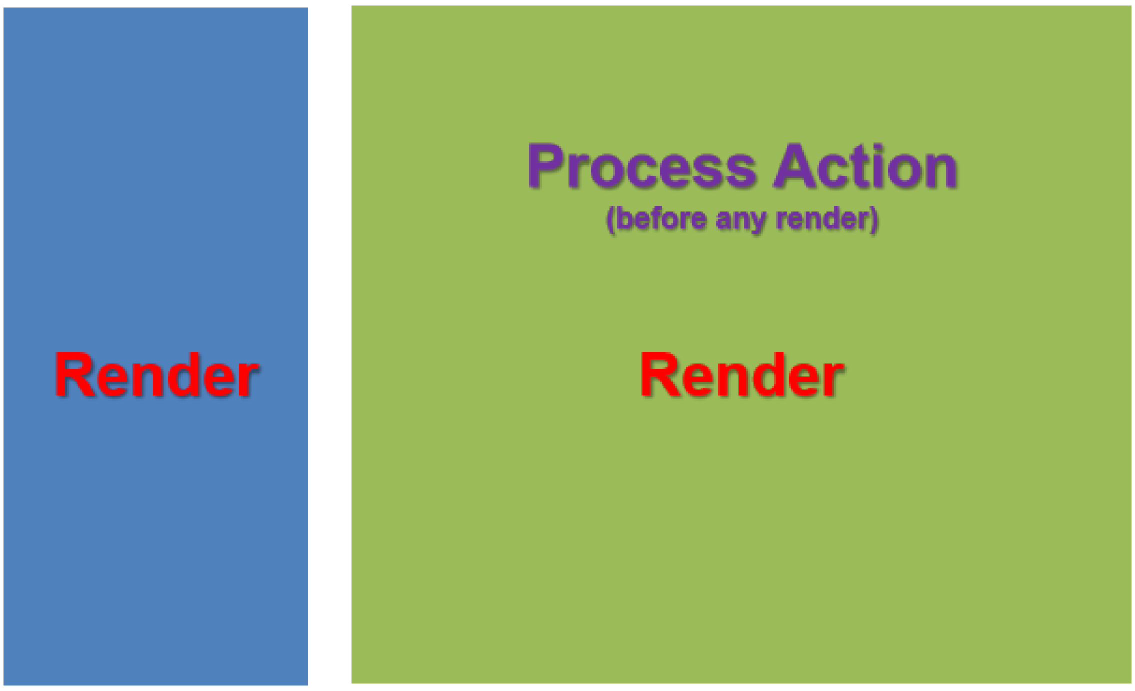


Figure E-3 Form Submission Action

**E4 Generic Portlet Class**

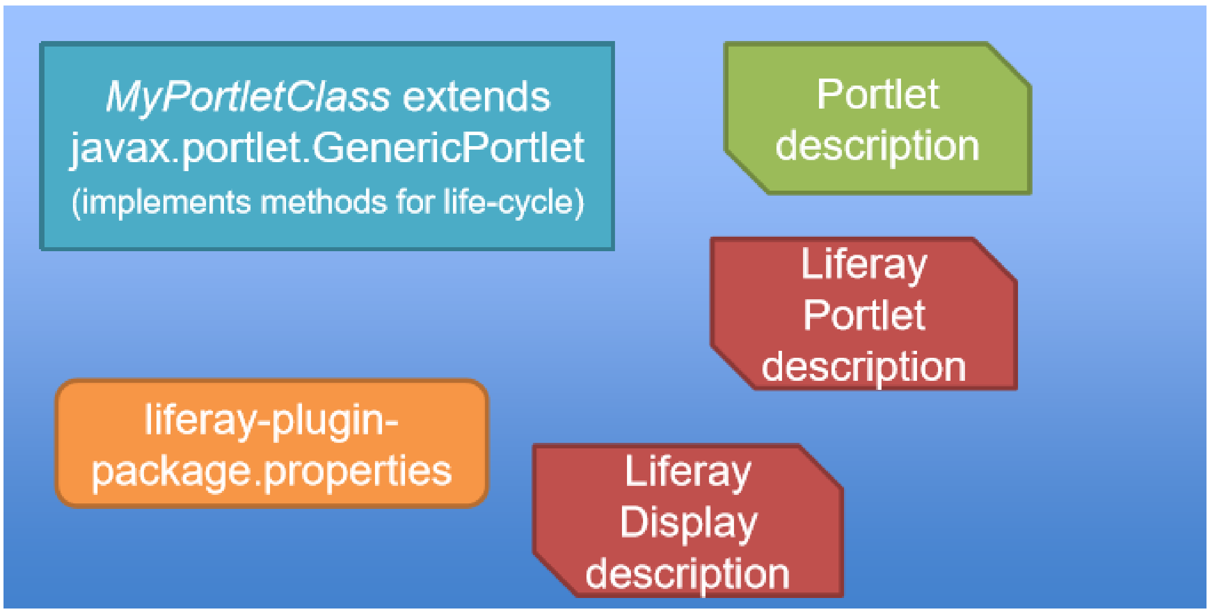

*MyPortletClass* extends
javax.portlet.GenericPortlet
(implements methods for life-cycle)
Portlet description
Liferay Portlet description
liferay-plugin-package.properties
Liferay Display description

Figure E-4 Infection Model Generic Portlet Class

**E5 Liferay MVC**

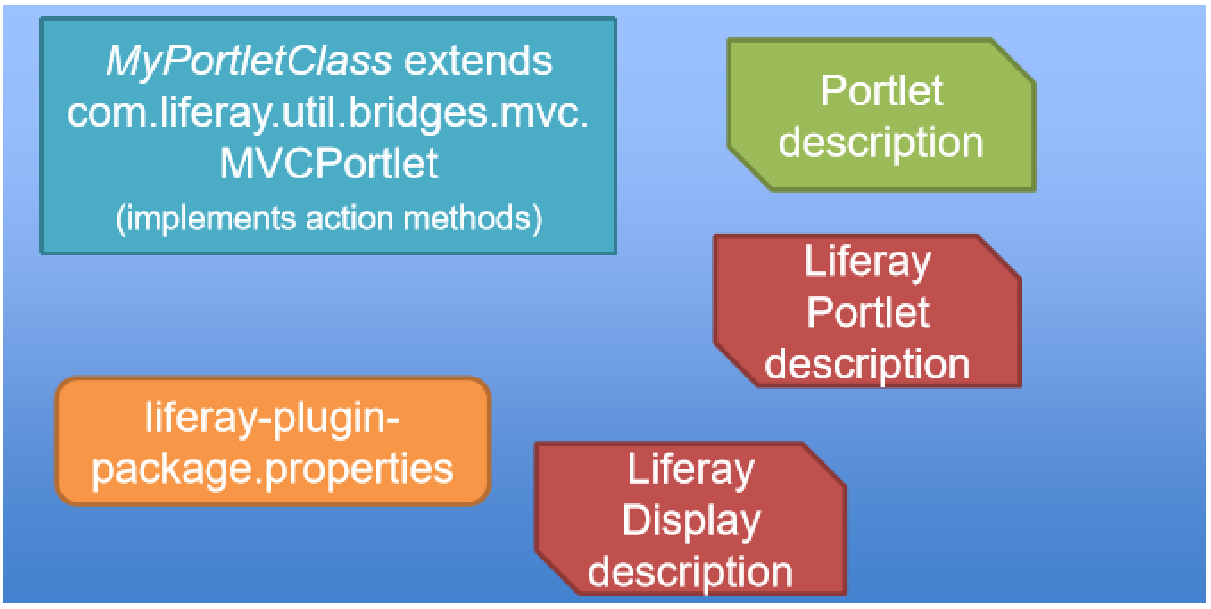

*MyPortletClass* extends
com.liferay.util.bridges.mvc.
MVCPortlet
(implements action methods)
Portlet description
Liferay Portlet description
liferay-plugin-package.properties
Liferay Display description

Figure E-5 Liferay MVC

# Appendix F

## CSGF and Future Gateway API Implementation Layers

### F1 The FutureGateway API Server

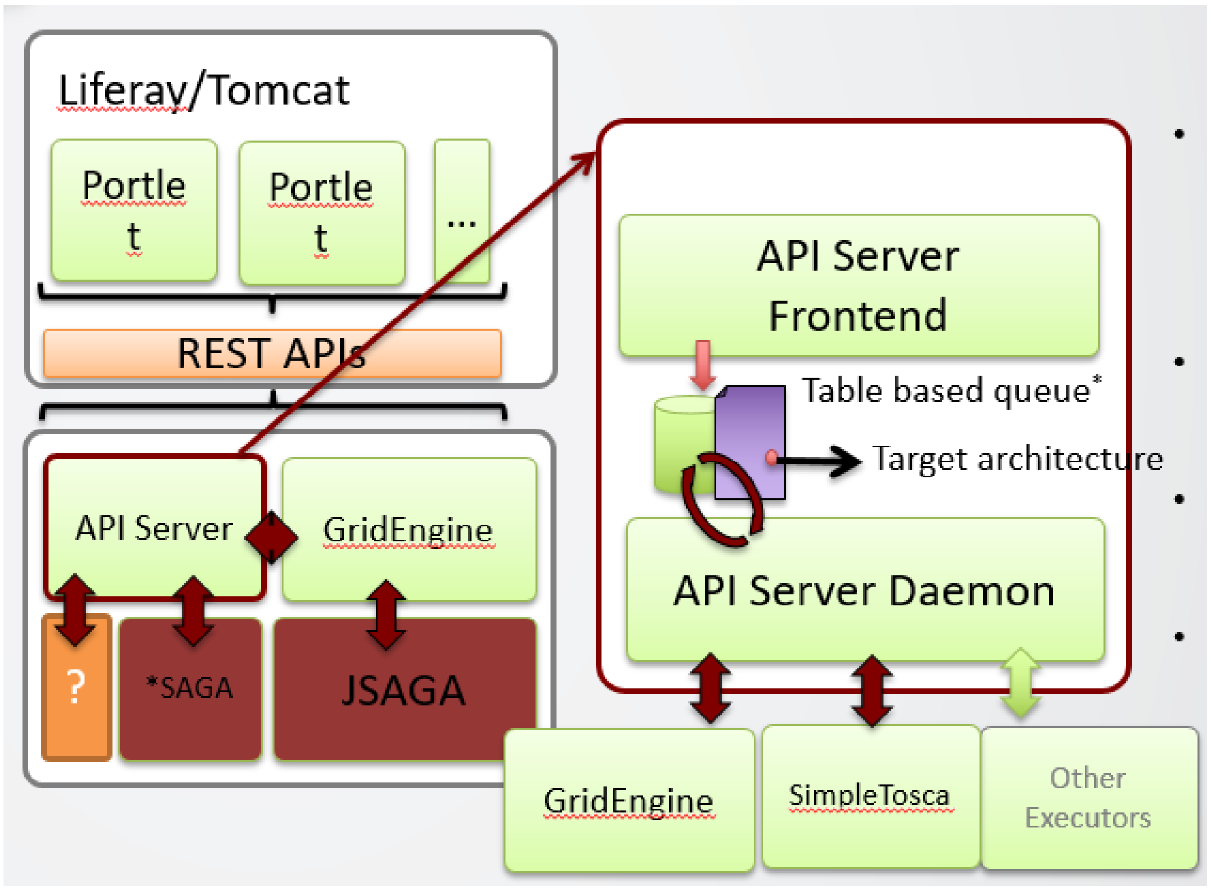
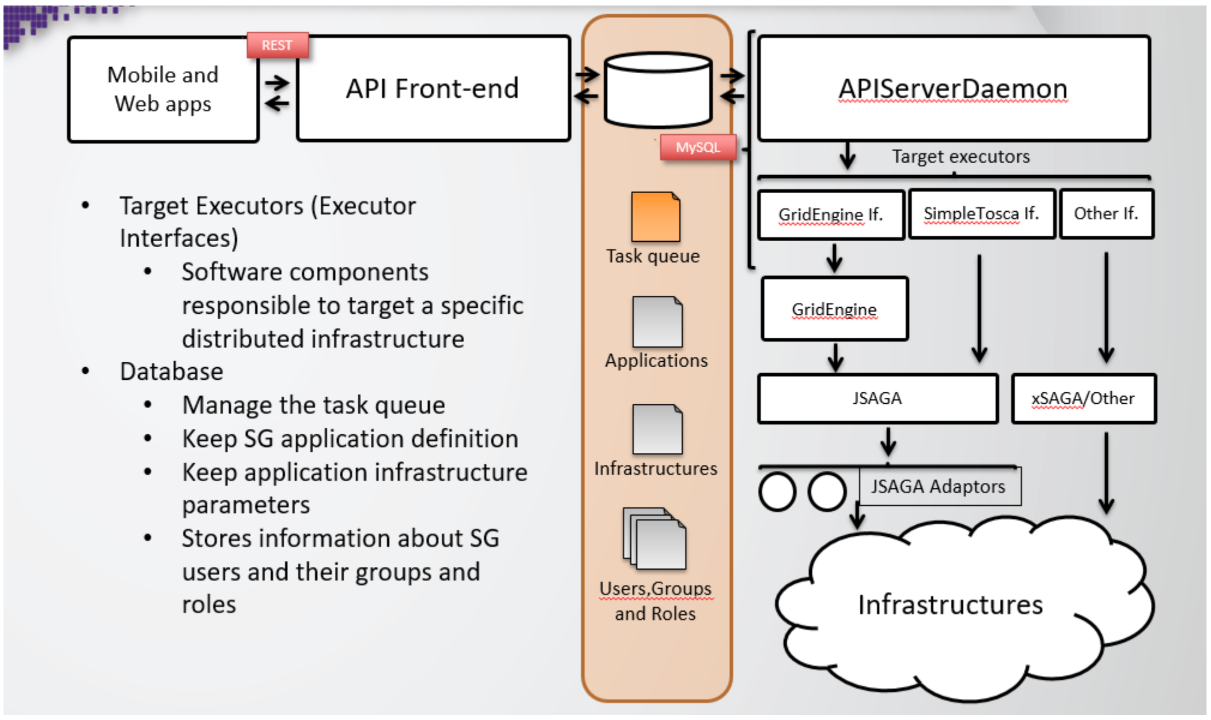


Figure F-1 The FutureGateway API Server

## F2. Future Gateway API Architecture



- Target Executors (Executor Interfaces)
  - Software components responsible to target a specific distributed infrastructure
- Database
  - Manage the task queue
  - Keep SG application definition
  - Keep application infrastructure parameters
  - Stores information about SG users and their groups and roles

Figure F-2 The FutureGateway API Architecture

## F3. Typical Portal Usage

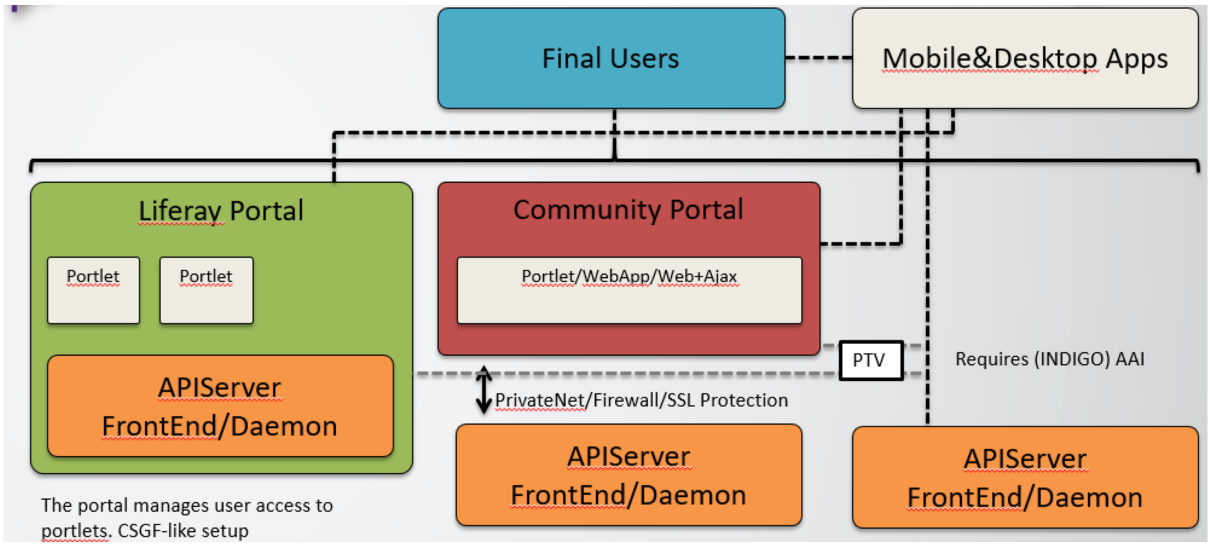
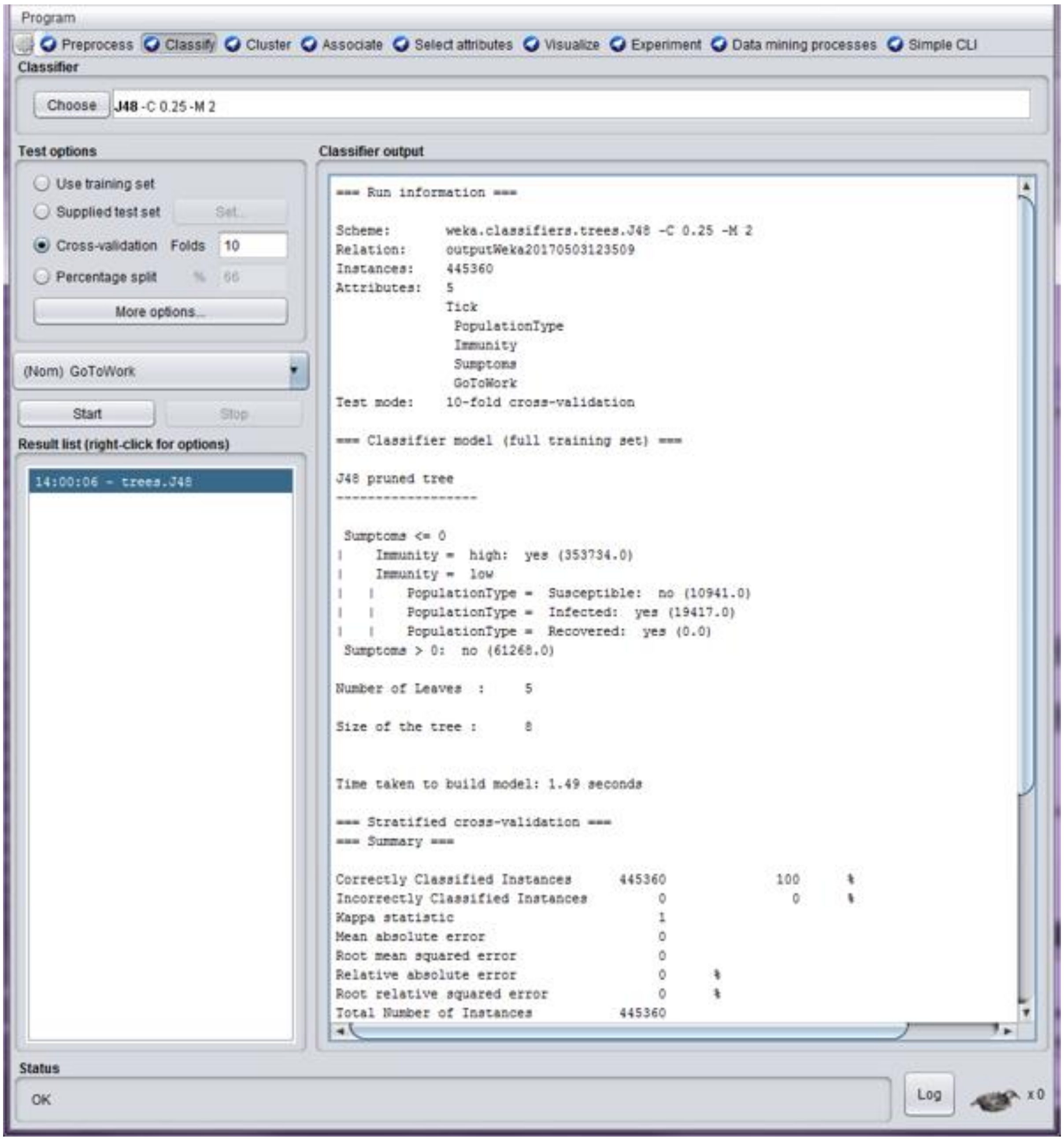

Final Users
Mobile&Desktop Apps
Liferay Portal
Portlet
Portlet
APIServer FrontEnd/Daemon
Community Portal
Portlet/WebApp/Web+Ajax
PTV
Requires (INDIGO) AAI
PrivateNet/Firewall/SSL Protection
APIServer FrontEnd/Daemon
APIServer FrontEnd/Daemon
The portal manages user access to portlets. CSGF-like setup

Figure F-3 Portal Usage Scenario

# Appendix G

## Analysis of the Simulation Output Result Using WEKA – J48 Portlet

### G1 Classifier Output



Classifier

J48 -C 0.25 -M 2

Test options

Use training set
Supplied test set
(●) Cross-validation Folds 10
Percentage split % 66

(Nom) GoToWork

Result list (right-click for options)

14:00:06 - trees.J48

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     outputWeka20170503123509
Instances:    445360
Attributes:   5
              Tick
               PopulationType
               Immunity
               Sumptoms
               GoToWork
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

 Sumptoms <= 0
|    Immunity =  high:  yes (353734.0)
|    Immunity =  low
|   |    PopulationType =  Susceptible:  no (10941.0)
|   |    PopulationType =  Infected:  yes (19417.0)
|   |    PopulationType =  Recovered:  yes (0.0)
 Sumptoms > 0:  no (61268.0)

Number of Leaves  :     5

Size of the tree :      8


Time taken to build model: 1.49 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      445360              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0
Root mean squared error                  0
Relative absolute error                  0      %
Root relative squared error              0      %
Total Number of Instances           445360
```

Status

OK

Figure G-1 Classifier Output

**G2 Weka – J48 Decision Tree**

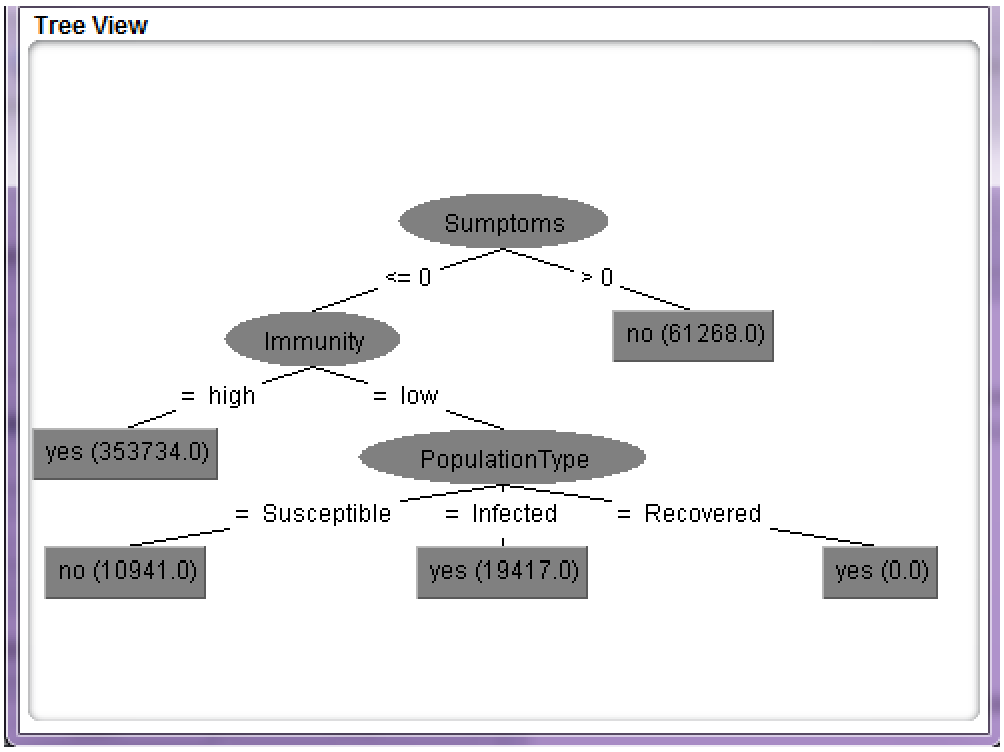


Figure G-2 WEKA - J48 Decision Tree

The Infection Model is slightly modified in order to perform analysis on the simulation output result using the WEKA – J48 portlet on the Science Gateway. The model is used to determine whether or not a person should go to work in the event of an infection outbreak. This event is carried out to obtain information from a set of instances in order to predict future occurences. In this experiment, there are 445360 instances and 5 attributes in total. The different attributes of the model include: Population type, Immunity, Symptoms and GotoWork. From the above Figure G-1, the classifier output shows the result of the classifier model on the full training set. The first split is on the Symptom attribute, the second split is on the Immunity attribute while the third split is on the PopulationType. The number of leaves in the tree structure is five while the number of nodes in the tree is eight. The number in front of each attribute represents the total number of instances that reach a leaf. According to the percentage split, 445360 instances were used for evaluation and all of them were correctly classified. Also, there were no incorrectly classified instances. This indicates the results obtained from the training set are optimistic. Subsequently, the J48 pruned tree (in Figure G-2) shows that; if the symptom level is more than 0, then a person should not go to work. However, if the symptom level is less than or equal to 0 and the level of immunity is high, then the individual can go to work. Also, if the level of immunity is low and the PopulationType is either Infected or Recovered, then an individual may choose to go to work. Finally, a PopulationType that is Susceptible shows that an individual may not go to work.