Investigating a Science Gateway for an Agent-Based Simulation Application Using REPAST

Adedeji O. Fabiyi, Simon J.E. Taylor, Anastasia Anagnostou
Modelling and Simulation Group
Department of Computer Science
Brunel University London, United Kingdom
Email: Adedeji.Fabiyi@brunel.ac.uk

Mario Torrisi and Roberto Barbera
Department of Physics and Astronomy of the
University of Catania and INFN via S. Sofia 64
Catania, I-95123, Italy

***Abstract*—The benefits of using e-Infrastructure environments, such as cloud, grid, and high performance computing, for performing scientific experiments could be quite significant. In particular, modeling and simulation, which can serve as a key decision making and system analysis tool, could benefit immensely from such environments ranging from issues of how a community of practice could access a simulation to how it could be run quickly. However, the access and use of these e-Infrastructure environments may present a completely different set of challenges, most especially for non-ICT users. Science Gateways (SG), which are digital interfaces to advanced technologies, can be used to overcome the challenges of running many simulations on e-Infrastructures in a reasonable amount of time. In this work, we developed a SG, based on the Liferay portal framework and the Catania grid and cloud engine. We show how an Agent-Based infection simulation, which has been implemented using the Recursive Porous Agent Simulation Toolkit (REPAST) Simphony, can be ported to a Science Gateway and deployed on distributed computing infrastructures. This demonstration illustrates how this technology can be used easily to allow multiple users across the world to access a simulation and to execute their applications in an e-Infrastructures environment.**

Keywords*—science gateways[1]; agent-based modelling and simulation; repast; infection model; liferay portal framework, catania science gateway framework, e-Infrastructures

I. Introduction

To effectively control the transmission of infections a thorough understanding of the determinants and patterns of the spread of such infection is paramount. Scientists have used simulation techniques to develop computer models, ranging from deterministic to stochastic models, in order to model the interactions between individuals within their social networks. In particular, Agent-Based Modelling and Simulation (ABMS) is used by different scientific domains to study the behaviour of adaptive systems and usually complex social networks [1]. However, once a simulation has been developed, there may be challenges in how a community of practice can access the simulation and how it can be run quickly.

Agent-based models consisting of large populations of agents are considered to be too slow to execute and, consequently, analysis with ABMS may then be hampered by poor performance due to the large number of replications that these types of analysis require [2]. Advances in networking and distributed computing techniques where different organisations can combine researches and resources across multiple administrative and organisational domains have changed the way scientists perform their experiments [3]. This is realized by the grid computing paradigm which Foster, Kesselman and Tucke [4] defined as the coordinated resource sharing and problem-solving in dynamic multi-institutional virtual organisations. This has evolved into complex international Information and Communications Technology (ICT) systems referred to, in Europe, as e-Infrastructures. Using distributed computing resources in this way to effectively run simulations can potentially allow authorised scientists to access the simulation and to increase computational power. On the other hand, the deployment and use of these resources can be extremely complex and could be quite a daunting experience which could, in turn, prevent non-ICT experts 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 [5], 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. In view of this, this work has focused on developing a simple but intuitive user interface, known as a Science Gateway (SG), for running and analysing simulation experiments of an ABMS model on different distributed computing infrastructures. We have developed a SG, based on the Liferay portal framework and the Catania grid and cloud engine, by customising an existing open source SG framework known as the Catania Science Gateway Framework (CSGF).

For the purpose of demonstrating how a SG can be used to support world wide access to a simulation model and to execute experiments on e-Infrastructures, we developed an demonstration ABMS infection model, which has been implemented using REPAST Simphony. The aim of the simulation is to show how ABMS can study the behaviour of infections with an annual outbreak by using appropriate input parameters with respect to illustrating how SG can be used in this context, particularly in Euro-African collaborations. We have shown how a SG can be used to allow multiple users, across the world, to access an ABMS running on the distributed computing resources of e-Infrastructures.

The rest of the paper is structured as follows. In Section II, relevant and related approaches for portal frameworks and

[1]This work was part-funded by the H2020 project Energising Scientific Endeavour through Science Gateways and e-Infrastructures in Africa (SciGaIA) (Project Number: 654237).

SG frameworks are briefly discussed. Section III introduces the CSGF approach and discusses its various components. Section IV presents a case study of the Agent-Based Infection Simulation application that we ported on the SG. Section V discusses the implementation of our ABMS on the CSGF SG. The discussion and results of our findings are presented in Section VI. Finally, we conclude our paper in Section VII.

II. Related Work

A. Portal Frameworks

A portal is a single web-based environment that enables the running of applications in a consistent and systematic way [6]. It ensures the rapid development of portlets by enabling a generic portlet repository where a large set of ready-to-use portlets can be found. They help in building portlets from reusable components and aid web developers in creating attractive web portals on the fly. They do not, however, provide back-ends that support Distributed Computing Infrastructure (DCI) access.

There are different types of portal framework which can be used to build portlets, rapidly. The two commonly used are the Gridsphere and Liferay portal frameworks. According to [7], the GridSphere portal framework was developed with a view to improve on the lessons learned from past Grid portal projects such as the Grid Portal Development Kit (GPDK) and the Astrophysics Scientific Collaboratory (ASC) portal. 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 stovepipe web applications that made code re-usability extremely difficult. The Gridsphere architecture takes two forms: the first is associated with the general portal framework used for assisting virtual organizations (such as scientists and project developers) and the other form aids in the development of reusable modular components (portlets).

Liferay is a popular open source framework that enables users to create attractive web portals [8]. It provides a runtime environment for hosting java-based portlets. The web portals that Liferay enables may consist of a wide range of applications such as blogs, wikis, discussion forums, shared calendar, etc. According to [9], the Liferay portal framework, which offers an easy-to-use Web 2.0 interface using Asynchronous JavaScript and XML (AJAX) and other presentation layer technologies, is an award winning portlet container. It has features such as Graphical User Interface (GUI) based personalization, drag-and-drop portlets, dynamic navigation, and an instant-add portlet library.

B. Science Gateways and Science Gateway Frameworks

Until recently, that the concept of SG framework and application-specific SG started to become a common place, early efforts to provide a more user friendly interface to access e-Infrastructures was realised in the GPDK and the Open Grid Computing Environments (GCE/OGCE) as described in the work of Fox et al. [10]. They both make use of Java Commodity Grid (CoG) toolkit in order to provide the necessary functionalities needed to implement the various DCI services [11].

SGs are digital interfaces that provide access to advanced technologies for the support of science and engineering research and education [5]. They are usually implemented as web and/or mobile applications and are used to provide access to community resources such as software, data, and high-performance computing. The concept of SG framework was introduced to mitigate the shortcomings of portal frameworks by facilitating a connection to back-end DCIs such as e-Infrastructures.

Typically, in order to develop a SG, a developer may either build from scratch (i.e. make use of a standard portal framework) or customise an existing SG framework (built on top of a portal framework together with the back-ends that provide access to various DCIs)[12]. SG frameworks connect to DCI back-ends by providing and exposing different APIs and interfaces to support access to a large set of e-Infrastructures. In such ways, instances of SG, which can provide access to many different sets of distributed resources for specific scientific domains, can be quickly developed by customising specific SG frameworks. Examples of SG frameworks include: The CSGF [13], Web Services Parallel Grid Runtime and Developer Environment Framework (WS-PGRADE/GUSE) workflow system [14], the Vine Toolkit [15], GridPort [16] and a Framework for Domain-Specific Science Gateways (InSilicoLab) [17]. These SG frameworks are more generic and are usually not specialised for a certain scientific domain. Scientists from different communities of practice can make use of them with appropriate technical and development support. Instances of a SG, on the other-hand, are more specific to the needs of a given scientific community. They are also known as application specific SGs. SGs belonging to this category include: Diagnostic Enhancement of Confidence by an International Distributed Environment (DECIDE) [18], Grid Initiatives for e-Science virtual communities in Europe and Latin America (GISELA) [9], Molecular Simulation Grid (MoSGrid) Science Gateway [19], e-BioInfra [20] and the VisIVO Science Gateways [21].

III. The Catania Science Gateway Framework Approach

In our work, we adopted the CSGF approach for porting our ABMS application onto the Africa Grid Science Gateway (http://sgw.africa-grid.org/) (as part of an ongoing work on promoting e-Infrastructures in Africa - see www.sci-gaia.eu). At the heart of the CSGF is the Catania Grid and Cloud Engine which consists of three different models, as seen in Fig. 1. These are: The Job engine for job management, the Data engine for moving data across different DCIs, and the Users Tracking and Monitoring which contains functions used to interact with the Users Tracking database [22]. The Job engine is responsible for managing the entire cycle of job execution. It ensures that all jobs being submitted through the SG interface are properly executed. The Data engine ensures a direct transfer of data between the SG and the e-Infrastructures. The Users Tracking and Monitoring is a user tracking and accounting tool used for storing information for each Grid usage. In addition, there are also several interfaces to and from the Catania Grid and Cloud Engine. These include the SG interface which contains connectors to call the functions of the Catania Grid and Cloud Engine and the interface to e-Infrastructures, known as the Simple API for Grid Applications

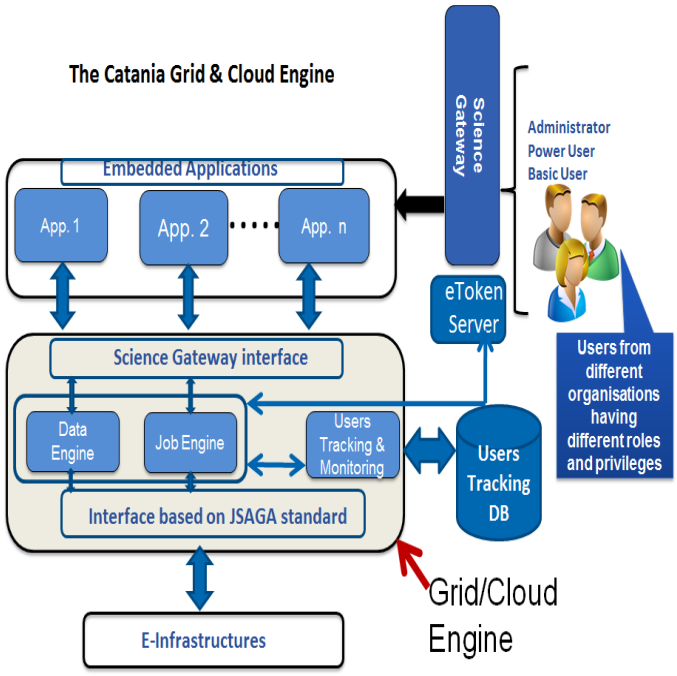

Fig. 1. Catania Grid and Cloud Engine

(SAGA) and its Java implementation of the SAGA (JSAGA) standard [9]. JSAGA enables interoperability between the different infrastructure middleware and jobs can be submitted to the different DCIs using this interface. The embedded application layer contains the different applications, such as our Agent-Based infection portlet, that have been ported on the SG. The e-Infrastructures layer is where different distributed systems (cloud, grid, dedicated high-performance computing facilities) reside.

In a nutshell, the Catania Grid and Cloud Engine is the software layer, based on SAGA, which allows applications to run on different infrastructures using different middleware. The CSGF has implemented different SGs with several applications implemented as portlets. An API/software layer (Grid engine) has connectors through to the portlets, so these portlets can easily call the function of the Grid and Cloud engine, and then the Grid and Cloud engine can call the function of the JSAGA through its interface. The portlets can therefore be executed on different e-Infrastructures in a seamless way [23]. As a consequence, when the functions of the Grid engine are called, it will take care of executing jobs and moving data on the different e-Infrastructures without actually knowing the details of the implementation of the different middleware.

One major advantage of the CSGF is that in order to submit a job to an e-Infrastructure, users do not need to have personal digital certificates or belong to any virtual organization to use the DCI. Proxy certificates which are mandatory for executing actions on e-Infrastructures are created and implemented as robot certificates [24]. These robot certificates, known as e-Tokens, are usually stored on USB form factor smart cards, and they are plugged into e-Token servers [22]. These certificates identify a person responsible for an unattended service or process acting on as client and/or 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 obtaining one. All that is required is VPN access and they will be able to select a certificate from the robot certificate server (e-Token server) to submit jobs, thereby simplifying the job execution process. A developer will, however, need to have registered to Catania's virtual private network (VPN) by requesting an account to be able to access the e-Token server.

In a similar way, when a user wants to execute a job on e-Infrastructures using the SG, users who do not have personal certificates or do not belong to any virtual organisation can still make use of the resources. To execute a job on the e-Infrastructures, the user must log on to the SG by selecting their respective Identity Federation and Identity Provider (a service that authenticates the user to the SG) and submit their federated credentials (containing a combination of username and password). If this is successful, the SG checks a Lightweight Directory Acces Protocol (LDAP), a registry that consists of all user roles and privileges, to see if they belong to any group and what roles and privileges they are entitled to. If this action is successfully completed, the SG will retrieve a proxy certificate (robot certificate) from the e-Token server, on behalf of a user, and jobs can be executed on different e-Infrastructures by using the JSAGA interface.

IV. CASE STUDY

To demonstrate how a SG can be used to execute an application on e-Infrastructures, we developed an infection model which was implemented in REPAST. The aim of the demonstration is to show how scientists can access an ABS model that is used to study the behaviour of infections with an annual outbreak. This, therefore, illustrates how SGs can be used in this context, particularly in Euro-African collaborations. There are three (3) types of agent to represent namely: the infected, susceptible and the recovered population. All agents are randomly located in a Grid environment. Susceptible agents try to avoid contact with infected agents.

When an infected agent approaches a cell with susceptible agent, it infects one, randomly, selected agent. This susceptible agent then becomes infected and can infect other susceptible agents in turn. The infected, susceptible and recovered agents are the input parameters that the user can modify in order to experiment with different initial conditions. The user can also specify the time (in years) that the simulation will run. Therefore, the infection model portlet has four(4) input parameters. These are:

- simulation period (specifies how many years the simulation will run),
- recovered count (specifies the initial recovered population),
- infected count (specifies the initial infected population) and
- susceptible count (specifies the initial susceptible population)

In the implementation of the ABMS infection model portlet that will be explained in the next section, the ABS infection model portlet allows multiple users to modify the input parameters and execute the simulation on DCIs from an interface that hides all the complexity of the underlying technologies.

V. IMPLEMENTATING AN AGENT-BASED SIMULATION APPLICATION ON THE LIFERAY PORTAL FRAMEWORK AND THE CSGF

Liferay contains many built-in applications called portlets [6]. To develop our infection model portlet, we adopted the CSGF which was built on top of the Liferay portal framework. Our infection simulation was deployed in a portlet named myRepast-infection-portlet. In order to manage the different portlet modes and the corresponding views to display, myRepast-infection-portlet made use of Actions enumeration (ACTION_ACTIVATE, ACTION_INPUT, ACTION_SUBMIT and ACTION_PILOT) and its corresponding Views enumeration (VIEW_ACTIVATE, VIEW_INPUT, VIEW_SUBMIT and VIEW_PILOT). myRepast-infection-portlet extends the GenericPortlet class and overrides the class methods. It defines the AppInit class which stores all the values of the portlet preferences. These include the portlet version, grid operation description, grid operation Id, number of infrastructures, etc. Furthermore, the AppInput class which stores all the application input values was also defined. This contains all the input parameters that will be specified by the user via the Infection Model portlet main page such as the simulation period, recovered count, infected count and the susceptible count. A constructor of this class, i.e the AppInput class, was created and each of the input value was set to an empty string. This is illustrated below:

```
class AppInput {
String inputValue; String inputValue2;
String inputValue3; String inputValue4;
String username; String timestamp;
String jobIdentifier;
String inputSandbox_inputFile;
```

Where inputValue, inputValue2, inputValue3, inputValue4 represents the simulation period, recovered count, infected count and the susceptible count, respectively.The constructor of the class was created and the input parameters were set to empty strings.

```
public AppInput() {
inputValue= ""; inputValue2= "";
inputValue3= ""; inputValue4= "";
username= ""; timestamp = "";
jobIdentifier= "";
inputSandbox_inputFile = "";
}
```

In order to retrieve the given application values that is being entered by the user (via the Infection Model portlet main page), a method, getInputForm(), was used to manage all the enumerated types containing all the JSP input parameters of the infection model. This method is of the form:

```
void getInputForm(ActionRequest
request, AppInput appInput) {
appInput.inputValue = (String)
request.getParameter(i)
(Where i is the inputValue,
inputValue2, inputValue3, inputValue4
and the JobIdentifier, respectively).
}
```

The infection model together with its dependencies, such as the REPAST libraries and the java runtime installation, is deployed within a pre-configured Virtual Machine (VM) image. When a user submits a job using the SG, this image will be used to start a VM on a remote cloud site, from the several infrastructures available in the portlet configuration. The pilot_script.sh, consisting of the instructions to start a simulation, resides in the portlet context and it will be uploaded to the remote VM as soon one becomes available. Snapshot image of the VM is created along with the pilot_script.sh that is used to run the infection model. Whenever a simulation job is submitted, the VM image will be started, according to the parameters that were passed to the job, and the executable will be performed using these parameters.

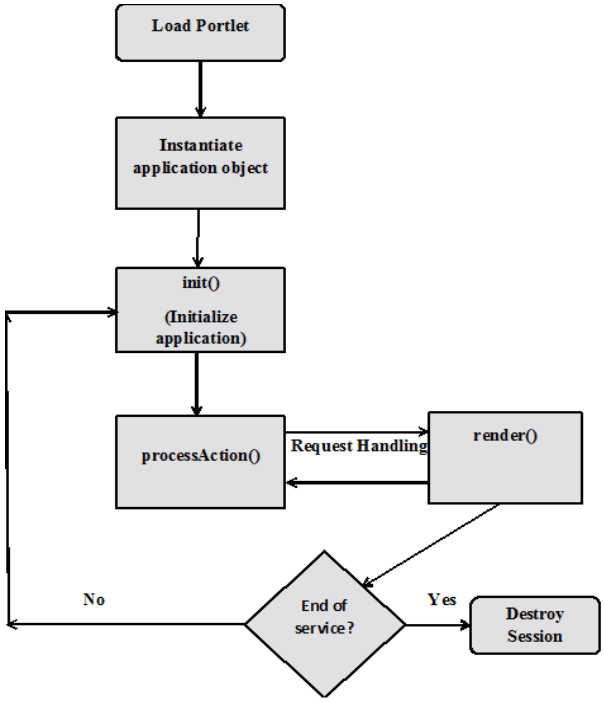

Fig. 2. Portlet Lifecycle

In order to run our infection model, we developed a shell script executable for the execution of our infection model. This script, named pilot_script.sh, performs a simulation each time a user submits a job by providing the necessary input parameters. The script also creates an archive folder containing all the generated output files in a single tar.gz file. When a user submits a job, the portlet can retrieve all the input parameters that the user specified using the getInputForm() method. These jobs can be sent to the Catania Grid and Cloud engine by using the SG interface. The job engine, by making use of the JSAGA, can then manage the whole life cycle of the job from the submission stage to the retrieval of the output files. (i.e it takes care of the job submission, job status and the output retrieval of the ABMS jobs).

Fig. 2 depicts the entire life-cycle of the portlet components. Once a portlet is loaded and instantiated in a portal container, such as the Liferay portal, the init() method is called (which would initialize application preferences) and all the configuration variables of the portlet are set. When a user performs an action using the Infection Model portlet, the processAction() method would be called which would in turn call the render() method. This interaction sees an exchange

of data between both methods (processAction() and render() method) and the java server pages (JSP) responsible for the presentation of the ABMS application. When a portlet reaches the end of its service, it is either relaunched by the init() method, if there are more actions to be performed by the user, or the portal container destroys the service using the destroy() method and the cycle ends.

A. The interaction between the init(), processAction() and the render() methods of the myRepast-infection-portlet main class

The Agent-Based infection application Java code, myRepast-infection-portlet, extends the GenericPortlets class and overrides methods such as: init(), doView(), doEdit(), doHelp() and processAction(), and it uses two main methods to exchange data to and from the JSP pages. This exchange is seen between the processAction() and the render() methods as seen in Fig. 2. These methods make it possible to exchange parameters between the user and a portlet. The processAction() method is responsible for the action being selected by the user in the input forms. The render() method determines the interface that is being shown to the user as a consequence of the actions being performed by the user. Elements of the processAction() and the render() methods include the ActionRequest and the RenderRequest. 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 such as the doView(), doEdit() and doHelp(). The Java code of the Agent-Based Simulation application is illustrated below:

```
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);
}
```

In addition to the java class of the ABMS application, myRepast-infection-portlet, there are other equally important java classes which are used by the main Java class of the ABMS application. These include the Applogger class, the AppPreferences class, and the AppInfrastructureInfo class. The Applogger.java is used to print out the console outputs while the AppPreferences.java class is used for storing the portlet preferences values. The AppInfrastructureinfo.java, on the other hand, stores the required information to submit a job into a given infrastructure.

Furthermore, there are different JSP pages which are used by the Render() methods to present the desired views. These include edit.jsp, help.jsp, input.jsp, submit.jsp and viewPilot.jsp, depending on the views/interfaces that a user might need to access. However, for the purpose of our Infection Model portlet, we have only opted for both the input.jsp and submit.jsp. Each interface is a JSP page of the ABMS application and each JSP page produces an action. This action will be taken by a method of the java source code, known as the processAction(), and will be forwarded to another method known as the doView() which can be used to switch between different interfaces according to the given action.

In order to assign the correct view, based on the different portlet modes (view, edit and help) and thus present the appropriate user interface in proportion to user actions and portlet status, the different JSP pages (input.jsp and submit.jsp) are paramount. These are the different ABMS application JSP pages that will be used by the Generic portlet java class methods (doView, doEdit, and doHelp) to assign the correct portlet view and present the appropriate user interface according to the user action and portlet status.

B. Agent-Based Simulation Portlet Modes

The portlet specification defines three portlet modes: VIEW, EDIT and HELP [25].

- View Mode: This generates a mark-up (i.e the normal user interface of the ABMS),
- Edit Mode: This allows for the customisation of the ABMS application and the setting of preferences.
- Help Mode: This explains the ABMS functionalities, i.e., the usage instructions of the portlet.

Our Infection Model portlet has been developed with the view mode that presents end users with the ABMS user interface (input.jsp and submit.jsp). When the portlet status is view mode, i.e. a user is currently on the view.jsp page and performs 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. The doView() method then performs an action and calls the view JSP page.

However, at the development stage, if the portlet is operating in the Edit mode, the render Request calls the doEdit() method and it sets the configuration variables and calls the necessary edit JSP page. Furthermore, if the portlet is operating in the Help mode, the doHelp() method is initiated and the help JSP page is displayed accordingly. 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 can be called, which bypasses the processAction(), as the doView() method calls the necessary view JSP page.

C. Data Exchange (Interaction) between the myRepast-infection-portlet main class and the ABMS application JSP pages

Interaction takes place between the JSP pages of the ABMS application and the Java code by using form statements to send parameters between the Java code and the JSP pages. During this interaction, there is 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 ABMS application. In order to implement the flow of data from the JSP pages of the application to the java code, all the java input fields are placed in the JSP code web form. i.e. <form action=<portlet: actionURL portlet Mode=view >.

In addition, within the java code (myRepast-infection-portlet class), the input interface values will be obtained with the methods: doView/doHelp/doEdit (RenderRequest request...)

In order to obtain the parameters, we just set the string param_i= request.getParameter(param_name_i); where param_name_i is the portlet status and param_i is the current view. For the flow that sees the exchange of data from the Java code to JSP, the input interface values inside the Java code will be 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); and inside the JSP page, we load the parameter values with <jsp:useBean id=param_name_k class= <variable type k> scope=request>.

VI. DISCUSSION AND RESULTS

To develop a SG, that can support access to community resources, we need to start by identifying specific needs for that scientific domain. The specific needs associated with our ABMS application include mechanisms to input data, perform execution, and interpret results. All these requirements and needs have been identified and captured on the SG. Our demonstration infection model was ported on the Africa Grid Science Gateway by customising an existing framework, known as the CSGF, which is built on top of the Liferay portal framework. This approach was mainly used due to the availability of useful functionalities, portlets and backend with access to various DCIs.

Within the SG, and along with other portlets belonging to different communities of practices, the infection model has been deployed in a portlet named Infection Model (see Fig. 3). This has been developed to enable users to conduct experiments with different input parameters and to obtain results. As well as the results output file, the application has a demonstration graph tool that allows users to see the graphical visualisation of the results. This shows that SGs can be developed to support online complex simulations in an extremely easy to use manner.

To view the Africa grid SG, a user needs to access its main page by using the web link https://sgw.africa-grid.org/. This will take the user to the main page of the SG. In order to access and use any application, on the SG, users need to send the request to, and obtain federated credentials issued by identity providers. If this request is granted, users can either sign on to the SG, by clicking on the sign in tab, or attempt to access any application from the applications menu bar and click on the run icon. As a result, users will be re-directed to a page that is made up of a number of identity federations, where they would be required to select the one which they belong. Within each identity federation, there are a number of identity providers and, similar to the identity federations, users would also be required to select an identity provider from their respective identity federation.

If the process of identity federation and identity provider are successfully completed, the user will be presented with a login page where they can simply use their federated credentials to logon to the SG. When an authorised user successfully log on, they are presented with the application page that they seek to use which, in our own case, is the Infection Model

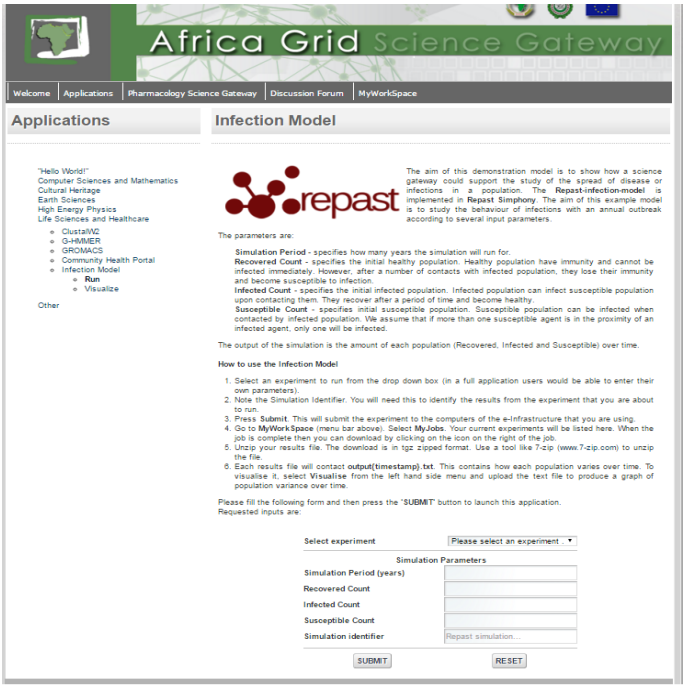

Fig. 3. Infection Simulation application main page

portlet as shown in Fig. 3, where they can specify all the input parameters that were described in section IV. After a user has finished specifying the parameters and clicked on the submit button, the Catania Grid and Cloud engine, using the JSAGA, can submit jobs on different e-Infrastructures without users knowing the implementation of the different middleware. After submitting a job, users would be notified that their jobs have been successfully submitted and then advised to check the MyJobs portlet, a dedicated portlet where the status of all running jobs as shown in Fig. 4, and done jobs can be found. A done job status would be represented by a small folder icon, also shown in Fig. 4, indicating that the job is ready and users can download the output of the infection model for analysis.

The infection model portlet also has a visualize page with a graph tool that allows users to see the graphical visualisation of their results, as shown in Fig. 5. Consequently, when a job is ready and the output is collected, a user can upload the output file, using the infection model visualisation tool on the SG and a graphical view of the job output would be generated.

VII. CONCLUSIONS AND FURTHER WORK

In this paper, we have shown a new approach to executing an ABMS application on e-Infrastructures. In order to capture the requirements of an ABMS application, on a SG, we started by identifying the simulation needs of our ABMS application. These include input data, execution activities, and interpretation of results. All these requirements have been implemented by adopting specific portal and SG framework, known as the Liferay portal framework and the Catania Grid and Cloud engine, respectively. The combination of both technologies is what is known as the CSGF. They both help to provide support for different aspects of the ABMS application and its implementation. The support includes:

Fig. 4. MyJobs portlet showing the view of an Infection Simulation application job in running status and a job that is ready to be downloaded

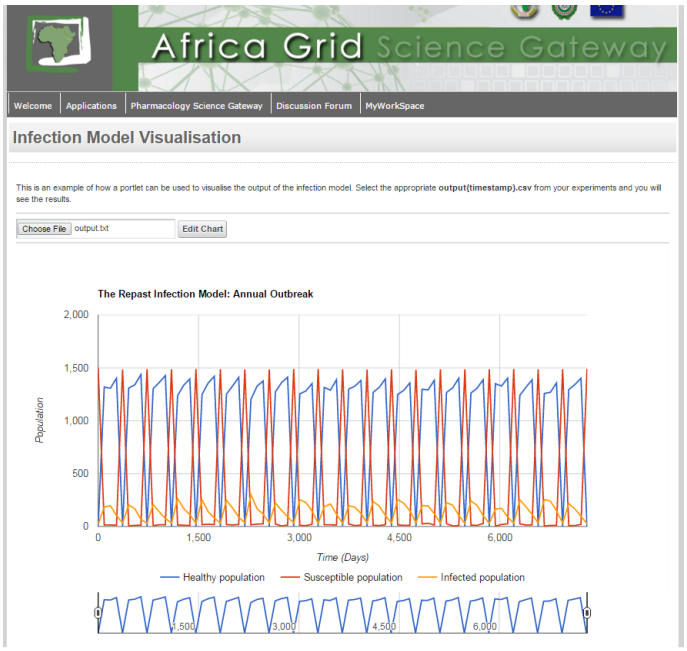

Fig. 5. Infection Simulation application Visualiser

- To help users specify different input parameters by making use of an intuitive and user friendly interface, implemented using the Liferay portal framework.
- The submission and execution of jobs on different e-Infrastructures by making use of the Grid and Cloud engine and its JSAGA functionalities.
- A graphical visualisation of the simulation output.

In order to support the access and execution of our ABMS application, on worldwide infrastructures, we provided an approach, known as SG, which captures and incorporates the different ABMS requirements and needs. For the implementation of these requirements, and the ultimate execution of jobs on the distributed systems, we adopted the CSGF approach that merges the Liferay portal framework, for developing portlet interface, and the Catania Grid and Cloud engine, which handles the execution of jobs on different DCIs. By adopting this approach, portlets can easily and efficiently be developed and used to access and execute jobs on e-Infrastructures.

One limitation of our ABMS portlet is that jobs can only be executed sequentially, i.e. instances of machines with single cores can be used to run jobs in distributed environments. As a consequence, future work in this area would investigate how our ABMS can be used to execute parallel jobs by making use of High Performance Computing (HPC) resources. The aim is to enable faster execution as we do a series of performance testing while running a number of experiments, simultaneously. We also aim to compare our results with the parallel implementation that is being done in some other, well-known, SG frameworks such as the Web Service - Parallel Grid Run-time and Application Development Environment (WS-PGRADE) portal.

ACKNOWLEDGMENT

Special thanks go to the team at the University of Catania for their support and the provision of the infrastructures that enable the execution of our ABMS application jobs. This work was part-funded by the H2020 project Energising Scientific Endeavour through Science Gateways and e-Infrastructures in Africa (Sci-GaIA) (project number 654237).

REFERENCES

[1] S. J. Taylor, A. Anagnostou, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini, "A tutorial on cloud computing for agent-based modeling & simulation with repast," in *Simulation Conference (WSC), 2014 Winter*. IEEE, 2014, pp. 192–206.

[2] N. Collier, J. Ozik, and C. M. Macal, "Large-scale agent-based modeling with repast hpc: A case study in parallelizing an agent-based model," in *Euro-Par 2015: Parallel Processing Workshops*. Springer, 2015, pp. 454–465.

[3] E. Laure and Å. Edlund, "The e-infrastructure ecosystem: Providing local support to global science," *Large-Scale Computing Techniques for Complex System Simulations*, vol. 80, p. 19, 2012.

[4] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *International journal of high performance computing applications*, vol. 15, no. 3, pp. 200–222, 2001.

[5] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, and S. Michael, "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4252–4268, 2015.

[6] R. Sezov, *Liferay in action: the official guide to Liferay portal development*. Manning Shelter Island, NY, 2012.

[7] J. Novotny, M. Russell, and O. Wehrens, "Gridsphere: a portal framework for building collaborations," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 5, pp. 503–513, 2004.

[8] P. Sarang, *Practical liferay: Java-based portal applications development*. Apress, 2009.

[9] E. Ingrà, F. Pistagna, R. Rotondo, R. Bruno, R. Ricceri, M. Fargetta, R. Barbera, V. Ardizzone, A. Calanducci, D. Scardaci *et al.*, "The gisela science gateway," 2014.

[10] G. Fox, D. Gannon, and M. Thomas, "Editorial: A summary of grid computing environments," *CONCURRENCY AND COMPUTATION*, vol. 14, no. 13/15, pp. 1035–1044, 2002.

[11] J. Novotny, "The grid portal development kit," *Concurrency and Computation: Practice and experience*, vol. 14, no. 13-15, pp. 1129–1144, 2002.

[12] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, and I. Marton, "Ws-pgrade/guse generic dci gateway framework for a large variety of user communities," *Journal of Grid Computing*, vol. 10, no. 4, pp. 601–630, 2012.

[13] M. Fargetta, R. Barbera, and R. Rotondo, "A simplified access to grid resources by science gateways," in *International Symposium on Grids and Clouds and the Open Grid Forum*, 2011.

[14] A. Balasko, Z. Farkas, and P. Kacsuk, "Building science gateways by utilizing the generic ws-pgrade/guse workflow system," *Computer Science*, vol. 14, no. 2), pp. 307–325, 2013.

[15] M. Russell, P. Dziubecki, P. Grabowski, M. Krysiński, T. Kuczyński, D. Szjenfeld, D. Tarnawczyk, G. Wolniewicz, and J. Nabrzyski, "The vine toolkit: A java framework for developing grid applications," in *Parallel Processing and Applied Mathematics*. Springer, 2007, pp. 331–340.

[16] M. Dahan and J. R. Boisseau, "The gridport toolkit: A system for building grid portals," in *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 2001, p. 216.

[17] J. Kocot, T. Szepieniec, P. Wójcik, M. Trzeciak, M. Golik, T. Grabarczyk, H. Siejkowski, and M. Sterzel, "A framework for domain-specific science gateways," in *eScience on Distributed Computing Infrastructure*. Springer, 2014, pp. 130–146.

[18] V. Ardizzone, R. Barbera, A. Calanducci, M. Fargetta, E. Ingrà, G. La Rocca, S. Monforte, F. Pistagna, R. Rotondo, and D. Scardaci, "A european framework to build science gateways: architecture and use cases," in *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*. ACM, 2011, p. 43.

[19] S. Gesing, R. Grunzke, J. Kruger, S. Herres-Pawlis, and A. Hoffmann, "Challenges and modifications for creating a mosgrid science gateway for us and european infrastructures," in *Science Gateways (IWSG), 2015 7th International Workshop on*. IEEE, 2015, pp. 73–79.

[20] S. Shahand, A. Benabdelkader, M. M. Jaghoori, M. a. Mourabit, J. Huguet, M. W. Caan, A. H. Kampen, and S. D. Olabarriaga, "A data-centric neuroscience gateway: design, implementation, and experiences," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 489–506, 2015.

[21] E. Sciacca, M. Bandieramonte, U. Becciani, A. Costa, M. Krokos, P. Massimino, C. Petta, C. Pistagna, S. Riggi, and F. Vitello, "Visivo workflow-oriented science gateway for astrophysical visualization," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 2013, pp. 164–171.

[22] R. Bruno, G. Allegri, G. Andronico, R. Barbera, F. Bitelli, A. Budano, A. Calanducci, F. Celli, E. Costantini, M. Fargetta *et al.*, "The aginfra science gateway for agricultural sciences," *POS PROCEEDINGS OF SCIENCE*, pp. 20–pp, 2013.

[23] R. Barbera, R. Bruno, A. Calanducci, A. Messina, M. Pappalardo, and G. Passaro, "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, 2013, p. 5697.

[24] C. Casarino, G. Russo, G. Candiano, G. La Rocca, R. Barbera, G. Borasi, S. Guatelli, C. Messa, G. Passaro, and M. C. Gilardi, "A geant4 web-based application to support intra-operative electron radiotherapy using the european grid infrastructure," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 458–472, 2015.

[25] "Portlet tutorial," accessed: 2015-08-25. [Online]. Available: http://jsr286tutorial.blogspot.co.uk/p/portletmode.html