EUBrazilCC
EU-Brazil Cloud infrastructure Connecting federated resources for Scientific Advancement

D4.2- Programming models - Prototypes

<table>
<thead>
<tr>
<th>Contract number:</th>
<th>FP7-614048 / CNPq 490115/2013-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Date of Project:</td>
<td>1 October 2013</td>
</tr>
<tr>
<td>Duration of Project:</td>
<td>28 months</td>
</tr>
<tr>
<td>Work Package:</td>
<td>WP4 – (Provisioning of a framework for the applications adaptation)</td>
</tr>
<tr>
<td>Due Date:</td>
<td>M22 – 31/07/2015</td>
</tr>
<tr>
<td>Submission Date:</td>
<td>31/07/2015</td>
</tr>
<tr>
<td>Partner Responsible for the Milestone:</td>
<td>BSC</td>
</tr>
<tr>
<td>Dissemination Level:</td>
<td>PU</td>
</tr>
<tr>
<td>Nature:</td>
<td>P</td>
</tr>
<tr>
<td>Author(s):</td>
<td>Daniele Lezzi (BSC), Jacek Cala (UNEW)</td>
</tr>
<tr>
<td>Reviewer(s):</td>
<td>Miguel Caballer (UPV) – José Luis Vivas (UFCG)</td>
</tr>
</tbody>
</table>
Change Log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
<th>Author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.1</td>
<td>23/06/2015</td>
<td>Initial Table of Contents &amp; Draft</td>
<td>Daniele Lezzi (BSC)</td>
</tr>
<tr>
<td>V0.2</td>
<td>06/07/2015</td>
<td>Contents on COMPSs</td>
<td>Daniele Lezzi (BSC)</td>
</tr>
<tr>
<td>V0.3</td>
<td>08/07/2015</td>
<td>Contents on e-Science Central</td>
<td>Jacek Cała (UNEW)</td>
</tr>
<tr>
<td>V0.4</td>
<td>12/07/2015</td>
<td>Final version for review</td>
<td>Daniele Lezzi (BSC)</td>
</tr>
<tr>
<td>V0.5</td>
<td>22/07/2015</td>
<td>Changes according to the reviews</td>
<td>Daniele Lezzi (BSC)</td>
</tr>
<tr>
<td>V0.6</td>
<td>23/07/2015</td>
<td>Changes according to the reviews</td>
<td>Jacek Cała (UNEW)</td>
</tr>
<tr>
<td>V0.7</td>
<td>25/07/2015</td>
<td>Final version with reviews</td>
<td>Daniele Lezzi (BSC)</td>
</tr>
</tbody>
</table>

Document Review

<table>
<thead>
<tr>
<th>Review</th>
<th>Version</th>
<th>Date</th>
<th>Reviewers</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>v0.5</td>
<td>18/07/2015</td>
<td>Miguel Caballer (UPV)</td>
<td>Included in the document</td>
</tr>
<tr>
<td>2</td>
<td>v0.5</td>
<td>15/07/2015</td>
<td>Jose Luis Vivas (UFCG)</td>
<td>Included in the document</td>
</tr>
</tbody>
</table>
Table of contents

Disclaimer ........................................................................................................................................... 5
Executive Summary ................................................................................................................................. 6

1. Introduction .......................................................................................................................................... 7
1.1. Purpose and structure of the document .............................................................................................. 7

2. Detailed prototype components description ....................................................................................... 7
2.1. COMPSs ........................................................................................................................................... 7
   2.1.1. COMPSs programming model .................................................................................................... 8
   2.1.2. COMPSs runtime ....................................................................................................................... 10
   2.1.3. Deployment ................................................................................................................................ 13
   2.1.4. Message sequence diagrams .................................................................................................... 15
2.2. e-Science Central ............................................................................................................................. 15
   2.2.1. Programming Model .................................................................................................................. 16
   2.2.2. System Architecture .................................................................................................................. 17
   2.2.3. Deployment ................................................................................................................................ 19

3. Links to Components and Documentation .......................................................................................... 21
3.1. COMPSs ........................................................................................................................................... 21
3.2. e-Science Central ............................................................................................................................. 21

4. Conclusions ......................................................................................................................................... 21

5. Acronyms and Abbreviations ................................................................................................................ 22

List of figures

Figure 1 - Task dependency graph corresponding to the example script ................................................. 10
Figure 2 - COMPSs Framework architecture .......................................................................................... 11
Figure 3 - The COMPSs IDE for Eclipse .................................................................................................. 11
Figure 4 - PMES architecture .................................................................................................................. 12
Figure 5 - PMES Dashboard .................................................................................................................... 12
Figure 6 - COMPSs execution monior .................................................................................................... 13
Figure 7 - Paraver analysis of COMPSs execution traces ........................................................................ 13
Figure 8 - Deployment of COMPSs and PMES ...................................................................................... 14
Figure 9 - Message Sequence Diagram for the COMPSs Framework ..................................................... 15
Figure 10. An example of a workflow designed for the Leishmania Virtual Laboratory ....................... 16
Figure 11. Basic architecture of e-Science Central .................................................................................. 18
Figure 12. The extended architecture of e-Science Central with three components: Scheduler, Performance Monitor and Infrastructure Manager ................................................................. 19
List of tables

Table 1. Acronyms and abbreviations............................................................................................................. 22
Disclaimer

EUBrazil Cloud Connect “EU-Brazil Cloud infrastructure Connecting federated resources for Scientific Advancement” (2013-2015) (hereinafter “EUBrazilCC”) is a Small or medium-scale focused research project (STREP) funded by the European Commission under the Cooperation Programme, Framework Programme Seven (FP7) Objective FP7-ICT-2013.10.2-EU-Brazil Research and Development cooperation, and the National Council for Scientific and Technological Development of Brazil (CNPq) of the Brazilian Ministry of Science and Technology (MCT) under the corresponding matching Brazilian Call for proposals MCT/CNPq 013/2012.

This document contains information on core activities, findings, and outcomes of EUBrazilCC project, and in some instances, distinguished experts forming part of the project’s External Expert Committee. Any references to content in both website content and documents should clearly indicate the authors, source, organization and date of publication.

The document has been produced with the co-funding of the European Commission and the National Council for Scientific and Technological Development of Brazil. The content of this publication is the sole responsibility of the EUBrazilCC Consortium and its experts and cannot be considered to reflect the views of the European Commission nor the National Council for Scientific and Technological Development of Brazil.
Executive Summary

This document describes the implementation of the programming model prototypes developed as a part of the EUBrazil Cloud Connect platform. The programming models offer the tools to implement user scenarios and execute them on the federated infrastructure. COMPSs and e-Science Central are the two available frameworks for the porting of the scenarios.

Both e-Science Central (e-SC) and COMPSs provide programmatic access through APIs to their workflow management and execution functionality. These APIs are the part of the programming framework layer and are available to the end-user applications.

COMPSs is a programming framework that aims at easing the development of applications (i.e. parallel applications, business or scientific workflows, compositions of services mixing services and code, etc.) and their execution on distributed environments. The framework implements a task-based programming model that allows applications to be written following a sequential paradigm and without need of a specific API, leaving to the runtime the responsibility to execute the code that detects data dependencies between tasks and exploits the inherent parallelism of the sequential code. The availability of a different connector, each implementing the specific cloud provider API, makes it possible the run of computational loads on multi cloud environments without the need of code adaptation, providing scalability and elasticity features and allowing the available resources to adapt to the actual execution needs. In EUBrazilCC COMPSs is used to implement Use Case 2 for the orchestration of parametric executions of the cardiovascular simulations, interoperating with the Fogbow federation middleware to manage resources.

e-Science Central (e-SC) is a cloud-based data analysis and workflow management platform that offers a visual programming model based on workflows. With e-SC the process of application development usually consists of just dragging processing blocks from the palette of available tools and connecting them together into a workflow. The system provides key services for many scientific applications such as secure storage, versioning of data, and audit and provenance logs.

In EUBrazilCC e-Science Central is used to support Use Case 1 – the Leishmaniasis Virtual Laboratory (LVL). It implements a number of molecular pipelines that can be invoked by LVL users on the federated resources provided by the EUBrazilCC computing infrastructure. However, to integrate e-SC with the LVL and federated resources the architecture of the system has been extended with three additional components: the Performance Monitor, Scheduler and Infrastructure Manager.

This report details how the architectural choices described in document D4.1 – Architecture report, and the use case requirements collected in deliverable D5.1 have been translated into developments and extensions of the programming frameworks. It also takes into account interoperability issues and standards implementation plan described in D6.1.

This document is part of the D4.2 deliverable, which also includes the software bundles available through the responsible partners’ repositories.
1. Introduction

1.1. PURPOSE AND STRUCTURE OF THE DOCUMENT

This document describes the Programming Models prototypes that have been developed as part of the EUBrazilCC Platform. With reference to the overall project’s architecture, it explains how the programming frameworks have been adapted in order to provide the required functionalities for the use cases and how they have been extended to interoperate with the federation layer. All software components and interfaces are described; deployment scenarios and sample codes are provided as reference examples.

The next section contains a description of the features of COMPSs and e-Science Central providing details on the development activities around each framework specific to the project’s objectives. The description of the deployment of the components in the EUBrazilCC platform is provided together with references to the software packages, manuals and installation guides. In section 3 the references to software packages and source code are provided.

2. Detailed prototype components description

The programming frameworks enable the implementation of the use cases and their execution on the federated cloud resources available in the project and federated by the Fogbow middleware.

In the next sections we provide a detailed description of the two programming frameworks, COMPSs and eScience Central, highlighting how they have been enhanced in order to be integrated into the infrastructure, and describing the interfaces they offer for developing applications, using the project’s use cases as reference examples.

2.1. COMPSs

COMPSs is a framework, composed of a programming model and a runtime system, which aims to ease the development and deployment of distributed applications and web services. The core of the framework is its programming model, which allows the programmer to write applications in a sequential way and execute them on top of heterogeneous infrastructures exploiting the inherent parallelism of the applications. The COMPSs programming model is task-based, allowing the programmer to select the methods of the sequential application to be executed remotely. This selection is done by means of an annotated interface where all the methods that have to be considered as tasks are defined with annotations describing their data accesses and constraints on the execution of resources. At execution time this information is used by the runtime to build a dependency graph and orchestrate the tasks on the available resources.
2.1.1. COMPSs PROGRAMMING MODEL

The programming model syntax enables the easy development of applications as composite services. A composite, called Orchestration Element (OE), is written as a sequential program from which other services and regular methods, namely Core Elements (CE), are called. Therefore, composites can be hybrid codes that reuse functionalities wrapped in services or methods, adding some value to create a new product that can also be published as a service. Moreover, all the information needed for data-dependency detection and task-based parallelization is contained in a separate annotated Core Element Interface (CEI).

Any COMPSs application can be composed of two different kinds of CE: Method CE and Service CE. Method CEs are regular methods of the application selected to be run remotely. To pick a method CE, the programmer declares the method in the CEI, adding the @Method annotation indicating the implementing class.

On their turn, Service CEs correspond to SOAP Web Service operations described in WSDL documents. To select a SOAP operation as a CE, the developer declares the service operation together with the @Service annotation describing the service details (namespace, service name and service port). The location of the service is not included in the CEI, but instead in the runtime configuration that actually decides which server will run the task; thus, the programming model syntax remains completely unaware of the underlying infrastructure.

The following code provides an example of a COMPSs application in Java. On the left side we have a sample Core Element Interface that includes the alyaadan_run method, designed as a method CE implemented in the aac.webservice.AlyaAdan class. It accesses two pieces of data: a string representing the parameters for the simulation, and a string indicating the path of the output files. On the right side we may see the definition of the Orchestration Element published as a service, which includes a run method needed to execute the web service, and a call to the alyaadan_run CE defined in the interface. The CEs are invoked as regular methods; the COMPSs runtime is in charge of replacing the method invocation with the creation of a new task, thereafter managing its dependencies and execution on top of the available infrastructure.

```java
public interface AlyaAdanItf{
    @Method(declaringClass =
            "aac.webservice.AlyaAdan", isModifier =
            false, isInit = false, priority =
            false)
    public void alyaadan_run(
            @Parameter(direction = Direction.IN)
            String params_in,
            @Parameter(direction = Direction.IN)
            String params_out);
}

@WebService
public class AlyaAdan{
    @WebMethod
    @Orchestration
    public String
    run(@WebParam(name="params_in") String params_in,
        @WebParam(name="params_out") String params_out){
        alyaadan_run(params_in,
                       params_out);
        return "AlyaAdan successfully finished!";
    }
}
```
The next code contains an example of COMPSs application in Python. This script performs some computation for a number of steps (line 5) and then merges the partial results, of type dictionary, into a final dictionary (line 6, variable `result`). Each computation receives a configuration parameter, initialised in line 3. The script can be executed as a sequential Python program, but in order to parallelise it with COMPSs, we define as tasks three functions called by the script: `init_conf`, `compute_step` and `merge`.

1. `result = {}`
2. `num steps = 3`
3. `conf = init_conf ()`
4. `for i in range(num steps):`
5. `step res = compute_step(i, conf)`
6. `merge(result, step res)`
7. `from pycompss.api.api import compss wait on`
8. `result = compss wait on(result)`
9. `print “Result: “, result`

A task definition in PyCOMPSs is done by means of Python decorators, which are part of the standard Python syntax and permit to wrap calls to functions, with some additional behaviour. In particular, the user needs to add, before the definition of the function, a `@task` decorator that describes the task. Continuing with the example, the code below shows the aforementioned functions together with their `@task` decorators. Function `init_conf` returns an object of class `Configuration` (defined in line 1), as stated by its decorator (line 4). Similarly, `compute_step` returns a dictionary (as specified in the decorator in line 7) and receives two parameters: an integer and a `Configuration` object. Finally, `merge` receives two dictionary parameters and merges them into the first one (line 13); in order to declare that the first dictionary will be modified along the task, the decorator defines it as an input-output parameter (line 11).

1. `class Configuration(object):`
2. `...`
3. `from pycompss.api.task import task`
4. `from pycompss.api.parameter import *`
5. `@task(returns = Configuration)`
6. `def init conf():`
7. `return Configuration()`
8. `@task(returns = dict)`
9. `def compute_step(step, conf ):`
10. `res = do some computation(step, conf)`
11. `return res
12. @task(dict1 = INOUT)`
13. `def merge(dict1, dict2):`
14. `dict1.update(dict2)`

Figure 1 depicts the task dependency graph built on the fly by COMPSs for the Python example. The first asynchronous task that is created corresponds to function `init_conf`, and thereafter the main program proceeds immediately to execute the computation loop and merge the tasks. Inside the loop, a total of 3 `compute_step` tasks are generated, depending all on the previous `init_conf` task because they receive the configuration object `conf` as input parameter - if no direction is specified for a parameter, it defaults to `IN`. The loop also generates 3 merge tasks, each depending on their corresponding `compute_step` for the partial result of the iteration (variable `step_res`); moreover, each merge task
depends on the result produced by the previous iteration (stored in result), and are subsequently ordered in a task chain. Once the loop completes, the program reaches lines 7-9, where the final result in variable result is printed. Before printing, though, the script needs to synchronise for the last value of result, produced by the last merge task. In order to do that, PyCOMPSs provides an API function, compss_wait_on, which stalls the main control flow until the last result value is obtained. Hence, the call to compss (line 8) will wait for the last merge task to finish before obtaining and returning the final result, so that it can be printed (line 9).

![Figure 1 - Task dependency graph corresponding to the example script](image)

### 2.1.2. **COMPSs Runtime**

One important feature of the COMPSs runtime is the ability to exploit the cloud elasticity by adjusting the amount of resources to the current workload. When the number of tasks is higher than the available cores, the runtime turns to the cloud looking for a provider offering the type of resources that better meet the requirements of the application and with the lowest economical cost. Analogously, when the runtime detects an excess of resources for the actual workload, it will power off unused instances in a cost-efficient way. Such decisions are based on the information on the type of resources, that contains the details of the software images and instance templates available for every cloud provider. Since each cloud provider offers its own API, COMPSs defines a generic interface to manage resources and to query about details concerning the execution cost of multiple cloud providers during one and the same execution. These, called connectors, are responsible for translating the generic requests to the actual provider’s API.

COMPSs does not provide only a programming model. The framework is complemented with a set of platform tools which facilitates (i) the development of the COMPSs applications by means of an Integrated Development Environment (IDE); (ii) the deployment of applications in distributed infrastructures by means of the Programming Model enactment Service (PMES); and (iii) the monitoring of executions by means of the Monitoring and Tracing tools. Figure 2 shows a diagram with all the tools composing the framework, and their corresponding place in the service lifecycle.
The IDE is implemented as an Eclipse plug-in that extends the Java development tools offered by the Eclipse core with a Service Editor (Figure 3) and a set of wizards and actions to implement and build COMPSs applications in an easy way.

The Service Editor guides developers during the process of defining the application interface, implementing the Orchestration and Core Elements according to the programming model syntax and deploying the application.

The transparent deployment of COMPSs applications on cloud infrastructures is delegated to the PMES PaaS component, whose architecture is depicted in Figure 4. Via a Basic Execution Service (BES)
interface, the PMES exposes the needed operations to the COMPSs IDE dealing with the intricacies of the deployment and contextualization operations, and the installation of the application packages, the required libraries, and the monitoring processes. A dashboard (Figure 5) is also available for the configuration of the user cloud environment.

![PMES architecture](image)

**Figure 4 - PMES architecture**

![PMES Dashboard](image)

**Figure 5 - PMES Dashboard**

The runtime of COMPSs provides some information at execution time so that the user can follow the progress of the application through a web interface that shows real-time information on the tasks being executed and on the usage of the resources (Figure 6).
At the end of each execution or file transfer, the COMPSs runtime also creates usage records. The usage records contain information about the resources involved in the task execution, the source and destination resources in data transfers, and the start and end time of each operation. Once the application completes, all these usage records can be processed by the Tracing tool in order to perform a postmortem reconstruction of the application execution across the different cloud resources. This reconstruction can be visualized by tools such as Paraver in order to detect bottlenecks and unbalanced parts of the application which could be fixed to increase the application performance (Figure 7).

2.1.3. Deployment

Figure 8 depicts the deployment of COMPSs and PMES in the EUBrazilCC infrastructure. A client is used to contact an OGF BES compliant endpoint in order to submit the execution of the application. This request is expressed through a JSDL document containing the application name, the input parameters and data references. When a request is received, a virtual machine is requested to the fogbow manager...
OCCI endpoint in order to deploy the application by downloading its package from a storage location. Once the runtime machine is deployed, the COMPSs application is remotely started. On its turn, the COMPSs runtime will schedule the tasks on a set of machines created on demand. This phase includes the staging of input files from remote locations as specified in the execution request. The PMES service can be also configured to boot a predefined number of VMs on the provider side where the service is deployed; this solution allows serving requests involving the execution of smaller operations within a reasonable amount of time, avoiding thus the overhead of VM creation; if the number of requests exceeds the amount of available resources, the service is still able to dynamically deploy new instances in order to cope with the loadburst.

In order to allow interoperability with the EGI Federated Cloud, the management of virtual machine images (VMIs) is performed through the EGI AppDB. The software developed by the project is indeed published as virtual appliances in the EGI Cloud Marketplace, and each provider in the federation maintains a list of VMs belonging to the eubrazilcloudconnect.eu Virtual Organization (VO). To this aim, the VMCatcher\(^1\) application is used in order to deploy VO-endorsed VMs. The fogbow manager can create new subscriptions in cases where requested images do not exist in the local cloud and are not subscribed by VMCatcher.

The support for the VM management is implemented using the rOCCI client that transparently interoperates with the OCCI fogbow endpoints deployed in the testbed on top of different middlewares as OpenNebula and OpenStack. The connector supports different authentication methods, including

\(^1\) https://github.com/hepix-virtualisation/vmcatcher
X509, with the aid of VOMS proxy certificates. To set up each connector instance, the user indicates which virtual images and instance type the specific provider offers. Thus, when the runtime asks for the creation of a VM, the connector selects the appropriate image and resource template according to the requirements (in terms of cpu, memory, disk, etc) and invokes the rOCCI client through Mixins, extensions of the OCCI Model that allow the instantiation of a VM through templates with additional capabilities (network, storage, etc). The interaction with the deployed resources is done via SSH. To allow the job submission and data transfers between worker VMs, public key exchange is used as authentication method. To this aim, cloud-init is used to contextualize the VM, configuring the SSH keys and deploying software packages such as the binaries used by the tasks, the application code, and the COMPSs worker application. The connector also implements methods for the computation of the economical and temporal costs on each provider, such as the accumulated execution cost, the current cost per hour, and the cost per hour.

### 2.1.4. Message Sequence Diagrams

![Message Sequence Diagram](image)

**Figure 9 - Message Sequence Diagram for the COMPSs Framework**

#### 2.2. E-Science Central

e-Science Central (e-SC) is a cloud-based data analysis and workflow management platform. It offers a visual programming model based on workflows and supports secure storage, versioning of data, audit
and provenance logs. The platform is well suited to many scientific applications that often consist of a set of tasks connected in a directed acyclic graph. With e-Science Central the process of application development usually consists of just dragging processing blocks from the palette of available tools and connecting them together into a workflow. The user can then execute the workflow, and the results are stored within the e-SC file system and can be displayed in the web front-end.

By design, the system is suited to process automatically batches of input data without user interaction, and can scale to problems that require up to TFLOPS of computing power and TB of storage. e-Science Central is also a portable platform. It can run on a public cloud (e.g. in Amazon AWS and Microsoft Azure, in which for improved performance it can use their native scalable storage services), in a cluster of machines, or on a laptop.

e-SC has been used in a number of scientific projects such as spectral data visualisation, medical data capture and analysis, and chemical property prediction. It provides currently about 250 workflow blocks that range from generic utilities to manipulate data matrices and specialist blocks wrapping the Weka data mining tools, to control blocks used for workflow parallelization. The number of blocks grows constantly as the system is exposed to new projects and application domains. For applications that require some specific tools not yet included in e-SC, users (developers) need to build relevant blocks by themselves.

### 2.2.1. Programming Model

e-SC realizes a classic dataflow programming model. A dataflow consists of blocks connected via data dependency links. A block may implement a local function or it may invoke remote service operations. The model has limited support for control primitives (does not support conditionals nor loops), and blocks can only share data through explicit dependency links (Figure 10). This simple dataflow model translates into ease of programming and flexibility. Given the palette of pre-defined workflow blocks, users may create their workflows simply by assembling blocks visually, using a web-based workflow editor provided by the system.

![Figure 10. An example of a workflow designed for the Leishmania Virtual Laboratory.](image)

Importantly, a block may also represent a sub-workflow. This adds hierarchical structure to a workflow design, but it also provides a simple mechanism for parallel execution because sub-workflows are scheduled independently from each other and can be executed concurrently on a cluster of compute nodes. Currently, the pipelines implemented in the project do not use sub-workflows and parallelization
is achieved at the user level – multiple LVL users can submit many pipelines in parallel, yet each pipeline execution is considered as a single, independent task.

Although e-SC is self-contained and can operate on its own offering users a web interface, it also exposes an API to enable external applications to interact with storage and workflow management subsystems. The API allows the system to become a part of larger applications in which e-SC delivers secure and scalable data analysis service. This particular feature has been used to implement the Leishmania Virtual Laboratory – e-SC implements a number of phylogenetic analysis pipelines that are called by the LVL via the API. The LVL web application can monitor the progress of pipeline execution, and collect results after the execution completes.

To facilitate integration with the LVL the e-Science Central system was extended in the following ways:

- **Added ability to run a specific version of a workflow** – previously, workflows could only be started in the latest version, which caused issues with workflow development and integration with the LVL. By adding the ability to run a specific version of a workflow, the LVL can rely on a tested and “released” pipeline version, whereas the pipeline developer has the freedom to update the pipeline without making the LVL submission system to fail.

- **Added ability to run workflows with partially specified public parameters** – an e-SC workflow could be called via API as a remote function call. It required, however, that all exposed block properties had to be provided with the API call. To make the integration with the LVL easier this requirement has been removed and currently a workflow can be invoked with only partially specified parameters (as if all parameters were optional). All unspecified parameters take their default value from the workflow definition.

- **Exposed description of public block properties via API** – the LVL pipelines implemented in e-SC are delivered to users as functions with parameters (public block properties). To make it easier for the LVL users to specify these parameters the description of block properties has been exposed via API. Thus, the LVL can present each workflow parameter with its original description of the underlying block property.

- **Added new block type: shell-wrapper** – the designed LVL pipelines and integration with the PDAS system require using a number of executable tools that are usually invoked from the command line. To make it easier to embed these tools in the system a new shell-wrapper block type has been added. It makes the inclusion of executable tools in workflows as easy as writing a simple shell script that transfers block properties and input/output data between the system and the underlying command.

### 2.2.2. System Architecture

The e-Science Central architecture consists of two basic components (Figure 11): the server with database and file store, and one or more workflow engines. The server offers a number of services such as workflow enactment, versioning, security, and provenance tracking, whereas the engine is responsible for the execution of workflow invocations.
The e-Science Central enactment system is based on a simple FCFS queue. The server receives workflow invocation requests from a user or from another currently running workflow, and submits them to the queue. At the other end of the queue, a pool of e-SC engines fetches invocations from the queue and executes them. Importantly, while running a workflow all workflow blocks are executed on the same engine and communicate through the local file system. This makes data transfer between blocks very efficient because it avoids accessing a remote file store. Only when the data of a workflow need to be transferred to the user or to other workflows are they sent to the e-SC file store.

The simple single-pool model used in e-Science Central, although very effective and scalable, is inadequate for utilising the federated and heterogeneous resources of the EUBrazilCC infrastructure. Workflow invocations need to be dispatched to different resources based on a number of features such as performance of the target execution node, its current load, and data locality. Moreover, the pool of execution engines needs to be adapted to serve the current workload efficiently.

Therefore, the system architecture has been extended with three additional components: the Performance Monitor, the Scheduler and the Infrastructure Manager (Figure 12). The Performance Monitor (PM) gathers vital information about the hardware and software infrastructure of the execution nodes. It collects static information such as system architecture and amount of memory, and dynamic data about current CPU load, thread allocation, and disk load, among others. This information is exposed to other components via the API. The Scheduler (SC) acts as a proxy between the server and engines, and uses information gathered by the PM to decide about the best execution node/pool for the workflow invocations. The Infrastructure Manager (IM) is responsible for managing the resource pools attached to the server. It can increase or decrease the number of resources based on the information from PM such as the length of the invocation queue.
Figure 12. The extended architecture of e-Science Central with three components: Scheduler, Performance Monitor and Infrastructure Manager.

Below is a brief summary of the current implementation of the three components:

- **Improved performance monitoring** – extends the existing implementation, making it suitable for both the Scheduler and Infrastructure Manager components. It is designed as a pluggable component that may not be deployed in the simple single-pool architecture. However, the PM is mandatory for the Scheduler and Infrastructure Manager components.

- **A prototype of the Scheduler component** – designed to be a pluggable component that, if deployed, acts as a proxy between the server and engines. The current implementation can schedule workflow invocations based on the current CPU load of the engines sending invocations to the least loaded node.

- **A prototype of an Infrastructure Manager component** – designed to be a pluggable component that, if deployed, can allocate and deallocate workflow engine nodes. The current implementation uses the queue length information to decide whether the engine pool needs to expand or shrink. This implementation can communicate with a fogbow manager to manage resources in a federated cloud.

### 2.2.3. Deployment

The e-SC system can be deployed in a number of ways: from an all-in-one deployment for testing and development purposes, to large-scale cloud deployments with hundreds of execution nodes and server components distributed in the cloud. The basic deployment options were presented in deliverable D4.1.

To ensure system flexibility the engine can dynamically deploy the blocks and libraries required to execute a workflow. It means that the number of engines in the pool can increase and decrease dynamically depending on the current workload, and also that the engine VMs can be generic. There is no need to preinstall any kind of software dependencies on the VMs except the engine itself; the engine will fetch them from the server when required. This feature fits very well into the cloud-based deployment since the e-SC engine can be delivered as a single one-fits-all VM image.

In the course of the project the most useful deployment scenarios were covered by a set of Ansible roles and playbooks, and integrated with the UPV IM via RADL descriptors. Currently, the e-SC can be automatically deployed for different use cases:

- **As a single node (all-in-one)** – mainly for development and testing purposes. The system is fully functional but runs on a single machine; hence, excessive load of the engines can negatively impact the web user interface.
- Server + engines(s) – the most useful (production) deployment. The system can sustain increased workflow and still effectively respond to the user.
- Server split between two nodes + engine(s) – the high-performance configuration where the web server and database servers are split across two separate nodes. This deployment option is dedicated for high throughput workflows and can sustain a large number of short time running workflows.
- Secure proxy + server + engine(s) – the secured version of the deployment where the main e-SC server can run from behind a HTTPS proxy. This setup is dedicated to the production use by the LVL.
3. Links to Components and Documentation

3.1. COMPSs


3.2. e-Science Central

- e-Science Central source code is available on bitbucket (branch EUBCC):
  - [https://bitbucket.org/digitalinstitute/esciencecentral](https://bitbucket.org/digitalinstitute/esciencecentral)
- Performance Monitor source code is available on bitbucket (branch EUBCC, currently available only on request):
  - [https://bitbucket.org/digitalinstitute/monitorserver](https://bitbucket.org/digitalinstitute/monitorserver)
- Scheduler source code is available on github:
  - [https://github.com/anirudhagarwal/Scheduler](https://github.com/anirudhagarwal/Scheduler)
- Infrastructure Manager source code is available on github:
  - [https://github.com/eubrazilcc/e-SC-fogbow/tree/EUBCC](https://github.com/eubrazilcc/e-SC-fogbow/tree/EUBCC)
- e-Science Central documentation is available on bitbucket:
  - [https://bitbucket.org/digitalinstitute/esciencecentral/src/a10a13ecdac2e7e6f7e57bf7623ceebb5bdd792/documentation/?at=master](https://bitbucket.org/digitalinstitute/esciencecentral/src/a10a13ecdac2e7e6f7e57bf7623ceebb5bdd792/documentation/?at=master)

4. Conclusions

This report provided a description of the components developed for the Programming Models in the EUBrazilCC platform how they fit in the architecture, and how they interact with other components. COMPSs and e-ScienceCentral are the main contributions whose functionalities, exposed through specific and interoperable interfaces, can be used to execute the scientific applications evaluated in the project’s infrastructure.
5. Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppDB</td>
<td>EGI Applications Database</td>
</tr>
<tr>
<td>BES</td>
<td>Basic Execution Service</td>
</tr>
<tr>
<td>CNPq</td>
<td>National Council for Scientific and Technological Development of Brazil</td>
</tr>
<tr>
<td>COMPSs</td>
<td>COMP Superscalar</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>e-SC</td>
<td>eScience Central</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IM</td>
<td>Infrastructure Manager</td>
</tr>
<tr>
<td>JSDL</td>
<td>Job Submission Description Language</td>
</tr>
<tr>
<td>LVL</td>
<td>Leishmania Virtual Laboratory</td>
</tr>
<tr>
<td>MCT</td>
<td>Ministry of Science and Technology</td>
</tr>
<tr>
<td>OCCI</td>
<td>Open Cloud Computing Interface</td>
</tr>
<tr>
<td>OGF</td>
<td>Open Grid Forum</td>
</tr>
<tr>
<td>PDAS</td>
<td>Parallel data analytics service</td>
</tr>
<tr>
<td>PMES</td>
<td>Programming Model Enactment Service</td>
</tr>
<tr>
<td>RADL</td>
<td>RESTful API Description Language</td>
</tr>
<tr>
<td>TB</td>
<td>Terabyte</td>
</tr>
<tr>
<td>UC</td>
<td>Use case</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VMI</td>
<td>Virtual Machine Image</td>
</tr>
<tr>
<td>VO</td>
<td>Virtual Organization</td>
</tr>
<tr>
<td>VOMS</td>
<td>Virtual Organization Membership Service</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
</tbody>
</table>

Table 1. Acronyms and abbreviations