From lesson learned to the refactoring of the DRIHM science gateway for hydro-meteorological research

Daniele D'Agostino, Emauele Danovaro, Andrea Clematis,
Luca Roverelli, Gabriele Zereik, Antonella Galizia

IMATI-CNR

Preprint of the paper accepted on the Journal of grid computing

March 2016

Abstract

A full hydro-meteorological (HM) simulation, from rainfall to impact on urban areas, is a multidisciplinary activity which consists in the execution of a workflow composed by complex and heterogeneous model engines. Moreover an extensive set of configuration parameters have to be selected consistently among the models, otherwise the simulation can fail or produce unreliable results. The DRIHM portal is a Web-based science gateway aiming to support HM researchers in designing, executing and managing HM simulations. The first version of the portal was developed during the DRIHM project using the gUSE science gateway toolkit. The lesson we learned is guiding a refactoring process that, together with a review of the most relevant technologies for the development of a science gateway, represent the focus of this paper. Beside the technological aspects, the need of a strong interplay between ICT and other domain-specific communities clearly emerged, together with coherent policies in the management of data, computational resources and software components that represent the ecosystem of a science gateways.

1 Introduction

Severe hydro-meteorological (HM) events like storms and floods are highly impacting on human society and economical activities. This is the reason why Hydro-Meteorology Research (HMR) can be considered one of the key research topics in this century, in particular considering the incoming climate change effects [1].

A full HM simulation is composed by meteorological, hydrological and hydraulic models. Many attempts have been made in different countries to build up sound model workflows [2, 3]. However in most cases the models in the chains are clumsily stitched together so that only one meteorological model i and one hydrological model j and one hydraulic model k fit together via handcrafted scripts. Adding another data set or replacing model j by model j_2 can involve considerable re-engineering and analysis and thus hampers progress.

The Distributed Research Infrastructure for Hydro-Meteorology[1] project (DRIHM) makes now possible to work in a modular environment, the science gateway named DRIHM portal[2]. The result is an enhancement of the modelling and data processing capabilities of the HMR community through the adaptation, optimization and integration of dedicated HMR services over the DRIHM e-Infrastructure. The science gateway is an open platform accessible to all interested scientists.

The Information and Communications Technology (ICT) aspects of the portal, implemented during the DRIHM project using the grid and cloud User Support Environment (gUSE) [4] science gateway toolkit and the European e-Infrastructure ecosystem, has been analyzed in a companion paper [5]. In particular we presented technological insights, with an analysis of the main features and issues of the adopted technologies and e-Infrastructures. The conclusion was that science gateway toolkits are powerful tools that allow non-ICT users to quickly deploy applications by providing a set of enabling technologies, plus front-end and back-end services ready to be used.

However in some scenarios more effective solutions can require the exploitation of different technologies: this is the case of the HM experiments at the basis of the DRIHM project. But this approach requires a strong interplay between the ICT and the domain-specific communities. The present paper discusses these aspects and presents how we are refactoring the portal services on the basis of the lesson learned. Our goals are to develop a more open environment with respect to the current one and to trade some of the

[1]The DRIHM website, http://www.drihm.eu

[2]The DRIHM portal, https://portal.drihm.eu

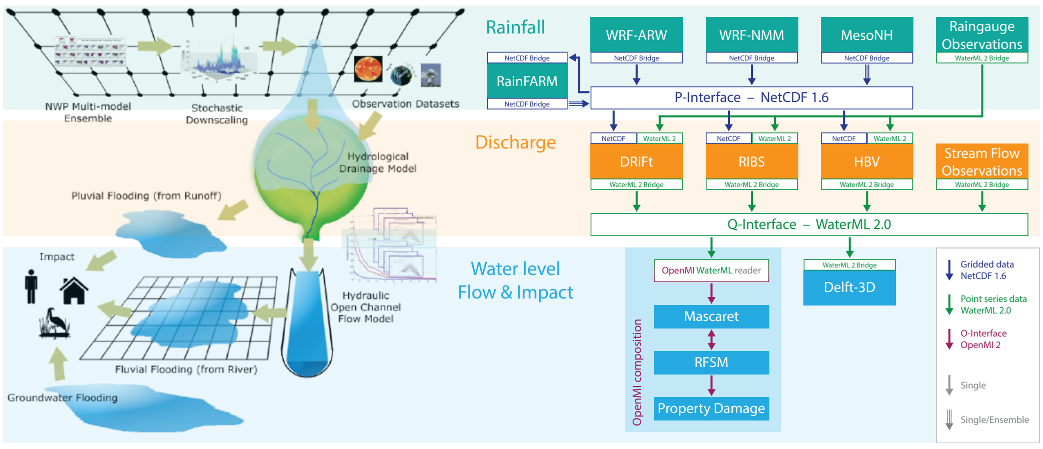


Figure 1: Hydro-meteorological simulations performed by DRIHM platform (left) and corresponding models available on the DRIHM platform, communicating trough standardized interfaces.

flexibility in workflow management (not required by the HM use case) for a lighter software architecture with a cleaner decoupling between the server and the user interfaces.

The paper is structured as follows: Section 2 summarizes the requirements of the HM experiments, the most important features of the DRIHM portal and the limits of the adopted technologies. Section 3 provides a brief overview of the available technologies science gateway developers can exploit. Section 4 discusses the refactoring process on the basis of the lesson learned, while the last Section concludes.

2 The DRIHM science gateway

The main purpose of the DRIHM science gateway is to support users in HM experiments configuration and execution.

An experiment can produce a rainfall simulation on a selected domain, a river discharge simulation, a water level and impact simulation in case of flood or a combination of them. Left side of Figure 1 highlights the three activities involved in a full simulation chain, while right side of Figure 1 represents the set of numerical models currently available in the science gateway[3]. We can simulate rainfall in two complementary ways: by executing a deterministic *meteorological model* (i.e. WRF-ARW, WRF-NMM and MesoNH) or with

[3]Details on software and data licences are provided in Deliverable 4.3, http://www.drihm.eu/index.php/project/deliverables

stochastic downsampling (i.e. producing a set of meteorological predictions with a stochastic approach) as in RainFARM. Discharge simulation is generated by *hydrological models* (i.e. DRiFt, RIBS and HBV), which simulate the flow in the river channel. In case of flood, water level and impact are simulated by *hydraulic models* (i.e. Delft3D or the OpenMI composition of Mascaret, RFSM and Property damage).

The adoption of standardized interfaces and proper data conversion tools developed in the project resulted in the possibility to freely couple these models. These interfaces are depicted in the right part of Figure 1, which can be interpreted as a direct graph: models are the nodes and arrows are the directed arcs, connecting two models sharing the same interface. Each possible simulation chain is a path on the directed graph, thus the selection of a single model, or a complex chain (e.g. WRF-NMM, RainFARM, RIBS and Delft3D) defines valid chains, supported in our science gateway.

2.1 Experiment configuration

The configuration of an experiment requires two major steps: at first the user has to select the models composing the experiment, then she/he has to configure each of them. At this aim, the science gateway provides the following set of functionalities:

- Experiment configuration: is the chain selection tool, and allows users to specify the desired simulation steps (meteorological, stochastic downsampling, hydrological and hydraulic) and the models to be run. After the model selection, further configuration features are enabled.
- Meteorological model configuration: this tool supports the definition of the geographic domain and time range, plus model-specific parameters. Some parameters may depend on the configuration of the other models belonging to the chain.
- Stochastic downsampling configuration: this implies multiple model instances' execution.
- Hydrological model configuration: this tool supports the definition of a simulation on a river basin. It requires complex geo-referenced configuration data describing the basin (such data are model specific), information on soil moisture and on expected rainfall (from simulation and/or observation).
- Hydraulic model configuration: calibrating an hydraulic model requires extremely complex desktop tools, so DRIHM relies on pre-calibrated

models freely available within the portal. Users can configure the experiments by selecting input discharge data.

The configuration steps are performed using the DRIHM portal. It has been designed with a strong focus on flexibility: instead of producing a monolithic user interface (UI), we decided to produce a component-based UI. For instance, the configuration of the hydrological models shares the basin definition, while dedicated UI components focus on model-specific parameters. With this approach we can easily add new models or modify their UIs without affecting the overall design. In the project we adopted the most relevant toolkit in Europe (it is currently used by about 30 gateways), the gUSE science gateway framework, based on the Liferay Portal. Therefore the UIs have been implemented using the portlet technology supported by Liferay.

It is worth considering that, configuring an experiment, we have to keep consistency among the steps composing it: in particular we have to consider coherent geographical domain, time range and time frame. We discovered that portlets, preventing inter-portlet communications, are a poor choice for a modular interface supporting experiment composition. Therefore we granted consistency relying on a complex and fixed dependency among portlets, as described in Section 4.1.

2.2 Experiment execution

The configuration of an experiment corresponds to the definition of a workflow composed of several blocks, one for each simulation step. When an experiment is submitted for the execution the portal creates, for each block, a set of files containing user-defined parameters, the mapping configuration with proper path and file names, and possible parallelism degree and calibration data. All these files are exploited by a general script that identifies the model selected, possibly downloads via rsync the related executables, downloads the necessary data and runs the simulation.

The actual execution of the blocks deeply depends on the simulation step. The numerical models in fact are characterized by specific and extremely heterogeneous (functional and non-functional) requirements. This is the reason why we exploit all the available European e-Infrastructure ecosystems as the European Grid Infrastructure (EGI)[4], the Partnership for Advanced Computing in Europe (PRACE)[5] and the EGI Federated Cloud[6]. In the companion project Distributed Research Infrastructure for Hydro-Meteorology

[4]The European Grid Infrastructure, https://www.egi.eu

[5]The Partnership for Advanced Computing in Europe, http://www.prace-ri.eu

[6]The EGI Federated Cloud, https://www.egi.eu/infrastructure/cloud/

to United States of America (DRIHM2US)[7] we considered also the use of the Extreme Science and Engineering Discovery Environment (XSEDE)[8]. This shared Distributed Computing Infrastructure (DCI) is described in details in [6].

The link between the front-end and the experiment execution has been implemented exploiting the gUSE Application-Specific Module Application Programming Interfaces (ASM API). Custom portlets guide the user in experiment configuration, produce input parameters and associate all the data with the experiment workflow steps. The workflows are managed using the gUSE services and the actual job submission is performed exploiting the DCI Bridge. It is worth mentioning that the DCI Bridge provides a standard Open Grid Services Architecture - Basic Execution Service (OGSA-BES) interface for submitting jobs to almost all DCIs: this represented one of the key aspect that motivated the selection of this toolkit.

When the model run ends, the outputs are stored on the DRIHM result repository and all files are deleted from the resources. In case the execution ends with a successful status, the portal submits the next simulation block following the same steps described above. Otherwise it provides error messages to the users. All these actions are supported by proper portlets.

2.3 The lesson learned

The DRIHM portal achieved the goal of providing a user-friendly front-end that hides all the low level ICT burdens and allows HM scientists to reduce from days to minutes the time required to run simulations using alternative models and experiment configurations [7, 8]. The adoption of gUSE allowed us to speed-up the portal development, by exploiting general-purpose services plus the possibility to submit jobs on almost all the DCIs in Europe and US. But, on the other side, during the development we identified several issues that led us to investigate alternative solutions [5]. Some issues arise from the integration of data and software used by the reference scientific community, with general purpose distributed infrastructures and other services (including middlewares and software components). A key aspect is that these components are produced/managed by different entities that have independent policies, roadmaps and goals. For example access rules based on individual capabilities represent a limiting factor. But a solution to this class of problems is out of our scope, and we can only provide recommendations.

[7]The DRIHM2US website, http://www.drihm2us.eu

[8]The Extreme Science and Engineering Discovery Environment, https://www.xsede.org/

From a technological point of view, the advanced support for the coherent experiment configuration required us to spend a large part of the effort in the UI and related logic, for proper parameters selection and consistency checks. Adopting the Liferay/gUSE architecture, we implemented the portal front end as a set of portlets. This led us to a first issue: portlet are self-contained components, and thus prevent direct intra-portlet communication. This is a main issue, because we have to keep consistency among the steps composing it: in particular we have to consider coherent geographical domain, time range and time frame. One possible solution is to have a single portlet for experiment configuration, with full support of intra-model constraints, but no flexibility to add new models. As an alternative we can split the configuration into multiple portlets, one for each model engine, gaining flexibility but loosing cross-model checks. Section 4.1 analyze in details the related issues and suggest a feature-rich yet flexible solution we are adopting now.

Moving from the UI to the underlying architecture we faced similar issues. The gUSE architecture is modular, but modules are tightly coupled and the framework is usually released in monolithic way: all components are upgraded at the same time, with little backward compatibility. Interfacing an evolving environment, like the computational resources and middlewares we exploit, with a platform that offers coarse-grained updates, is really difficult. For this reason we are moving to a more modular architecture, with loosely coupled components and harmonic but separated lifecycles. Section 4.2 focus on the detailed description of our solution, rooted in the trend of micro-services.

The first DRIHM portal implementation was intended as a proof of concept, providing data on a limited number of datasets. Nonetheless, in the few months after public release, the scientific community showed a great interest, with nearly two hundreds users and thousands of simulations. Peak activities (mainly after a flood) heavily loaded the system, consisting of a JavaEE state-full application hosted in a Tomcat webapp container. Scalability of this architecture is tricky, so we decided to move to lighter, stateless server components, as described in Section 4.3

Moreover, working on a proof-of-concept, we had the chance to focus on model interoperability, complex experiment definition, user-centric UI, but we spent a limited effort on data provisioning and management, focusing on a few test cases. When the system proved to be useful and effective, we have got many request to extend the system scope, to support additional test cases and to provide provenance information. Therefore the fourth improvement we are implementing is related to data management and data sharing: Section 4.4 outlines the new features plus some future directions.

3 Technologies and Software Tools for a Science Gateway

The various scientific communities involved in the development of a science gateway present different requirements due to the software and/or data they share and the goals they aim to achieve. For example some communities may be interested in running large simulations composed by a few software modules on powerful infrastructures (as in the geosciences [6, 9]), other communities in providing a collection of analysis tools that can be chained together (as in bioinformatics [10]), and some services can receive particular attention, as the data sharing [11] or the data visualization [12].

The result is that a science gateway has to provide a set of integrated and easy-to-use functionalities. In particular the most common ones are the following:

- user management (including accounting, billing and multi-site/service access);
- dedicated/general purpose UI to support experiment definition and configuration;
- workflow engine, to orchestrate heterogeneous software for the experiment execution;
- data management (i.e. provided and supported technologies to store, fetch and transmit data);
- job execution (based on local, remote services or heterogeneous DCIs);
- data provenance.

Some of these items can be addressed by general-purpose, ready-to-use solutions (e.g. Grid certificates, workflow management systems), some others instead rely on ad-hoc solutions (i.e. the UI) that can be developed using general-purpose technologies. However the key aspect is that a science gateway has to integrate a subset of/all these functionalities in one user-friendly infrastructure. For example privacy and security aspects, i.e. personal or group based access rights for data, software and computational resources access, should rely on a technology shared by most of the previous items.

The needs and opinions of the people involved in gateway initiatives have been investigated in 2014 with a survey having nearly 5,000 respondents [13]. The most relevant results, for the aim of the present paper, are the following. About 11 kinds of applications are provided within a gateway, but the most important ones are education tools (18%), computational tools (16%), data analysis tools (16%) and data collections (15%). At least 40% of the participants indicated that some help might be needed by a service provider in adapting technologies, usability services and choosing technologies. This also because the available technological solutions science gateway toolkits provide can be not enough flexible and powerful for all the possible communities and applications. In this Section we analyze the most important alternative "ingredients" - in terms of technologies, software tools and their features - we and other science gateway developers can exploit in implementing a new/updating a portal.

While gUSE is the most relevant toolkit in Europe, Apache Airavata [14] is the most relevant toolkit in US. The main interface of Airavata is represented by the API Server. It is based on Apache Thrift, which can be used to generate "client Software Development Kits" (SDK) in several languages as Java, PHP, Python and C++. Gateway developers integrate the selected SDK into their gateway front-end, which can access the server using the TCP/IP, HTTP(S) protocols. Airavata supports the execution of workflows on local or remote resources, as those provided by XSEDE, available via ssh, Globus, Unicore and Cloud middlewares. The Credential Store component manages user credentials needed by a gateway to securely interact with resources belonging to these infrastructures. The Application Factory (GFAC) instead provides a generic framework to wrap command line applications by generating a SOAP, REST or a native Java interface. The application provider has to specify the application input, outputs, temporary working directories and remote access mechanisms for file transfers and job submissions. These data are then stored in a Registry Service, that can host also information about datasets. The Orchestrator component acquires the workflow configuration, the characteristics of each steps from the registry, the input data from the front-end and interacts with GFAC for managing the execution of the jobs composing the workflow. At last, the Job Monitor is decoupled from the submission mechanisms. This choice allows the use of both pull approaches (i.e. "qstat") and push messaging.

The Vine Toolkit [15] is a Java library designed to provide a modular, extensible an easy-to-use API for Grid-enabling applications. This framework has been used mainly in the Polish Grid Infrastructure to develop gateways for their strategic areas, e.g. Nanomechanics, Quantum Chemistry and Molecular Physics. Vine provides the basic functionalities required to distributed applications submission and monitoring as well as data and workflow management, security and user management. These last functionalities are provided by a set of co-bundled components that, for example, support the integration with iRODS server for providing advanced search functionalities based on metadata information. A new service in Vine can be added by creating a separate software component and implementing a set of predefined APIs. Adobe Flex and BlazeDS technologies can be used to create Web-based user interfaces based on Flash, as well as the portlet technology. Vine Toolkit may be used in different ways, i.e. with Gridsphere, Liferay or in the standalone mode. Vine presently supports the execution on the following middleware stacks: gLite, Unicore, Globus and CosQosGrid, a middleware supporting advance reservation and co-allocation features required by distributed multi-scale experiments.

The "A grid and virtualization environment" (Agave) [16] provides a "Science-as-a-Service" (ScaaS) solution for hybrid cloud computing. With respect to pure "Platform-as-a-service" (PaaS) solutions it provides also computational, data, and collaborative services. It started as a pilot project of the iPlant cyberinfrastructure for plant biology in 2011 and now it supports experiments for many life science disciplines. In details the Agave platform runs in the Cloud as a hosted, multi-tenant service that support developers and bioinformaticians in building applications using the components of the iPlant cyberinfrastructure. Agave provides an API supports the registration and management of applications (currently more than 600 relevant scientific codes are available), multi-protocol data movement and data transformation tasks, structured and unstructured metadata management using the NoSQL technology, the submission and monitoring of jobs on heterogeneous resources ranging from HPC to Cloud infrastructures, plus the management of resources and users. Users can run their software as source code, binary code, virtual machine image, and it is possible to combine it with the available applications. The resulting workflow can be saved for future reuse and possibly shared. The Agave client SDK allows Java, Python, PHP, R, and Perl applications to interact with these services, besides the possibility for users to exploit the general purpose Agave ToGo Web application. The entire Agave Platform has been built as a collection of Docker images, i.e. images constructed from layered filesystems containing the software of the service and all the necessary dependencies, and can be installed on a PaaS infrastructure provided with a Docker container.

HUBzero [17] is an open source software platform for supporting collaborative research and educational activities. The defining characteristics of this platform are the delivery of visual simulation tools, that look like simple Java applets embedded within the browser, and the strong focus on the collaborative aspects. Hubs are places where users can share datasets, software and information as papers, presentations and other educational material. Presently there are about 400 online tools grouped in 23 hubs, e.g. for collaborative volcano research and risk mitigation, pharmaceutical product development and manufacturing, earthquake engineering simulation, besides the original nanotechnology community. The HUBzero infrastructure includes a tool execution and delivery mechanism based on the Virtual Network Computing (VNC) graphical desktop sharing system and OpenVZ, that creates multiple lightweight and isolated containers on a single physical server with a controlled access to file systems, networking, and other server processes. Any tool with a Graphical UI (GUI) can be installed on the hub in a near straightforward way. A GUI can be quickly created for command line software tools by using the Rappture toolkit. It reads an XML description of the tool's inputs and outputs and then automatically generates a GUI. The binding between this GUI and the software is provided though bindings for a variety of programming languages, e.g. C/C++, Fortran, and Python. The jobs can be dispatched XSEDE or other participating cluster resources.

A promising project is represented by the RADICAL Cybertools [18], a programming library to manage heterogeneous DCIs for extreme-scale applications using lightweight, extensible and interoperable building blocks. It currently consists of two components. RADICAL-SAGA is the infrastructure access layer of the RADICAL Cybertools stack. It provides a homogeneous programming interface that supports XSEDE and cloud computing platforms based on SAGA. RADICAL-Pilot is a flexible system that supports application-level resource management. It is based on the "pilot" abstraction, which combines the ability to support extreme-scale task-parallel workloads with a clean model and flexible execution of heterogeneous concurrent tasks. The EnsembleMD Toolkit (EnMDTK) project is a Python library providing a loose collection of RADICAL-Pilot based tools for molecular dynamics workflows.

This brief analysis shows that every toolkit has its characteristics, derived from the requirement of their reference user communities and infrastructures. To summarize, many of them are based on the portlet technology, are targeted to a subset of the available DCIs in Europe and US and moreover provide a collection of pre-defined components for connecting/supporting external frameworks, repositories and software applications that are hardly portable in a new gateway based on a different toolkit/technology. The present situation for the science gateways toolkits is similar to the scenario of the Grid infrastructure middlewares before the activities carried out in the European Middleware Initiative [19]. The main achievements of this project are the delivery of a consolidated set of middleware components for deploying DCIs and the establish of a sustainable model to evolve the components addressing partially overlapping and similar needs [20]. For example the usability has been enhanced by removing redundancy and consolidating the services, simplifying the security management, and adding better programmability interfaces. This because interoperability and compatibility can be granted only by removing proprietary interfaces in the middleware services and ensuring true interoperability through the adoption of community standards (whenever possible) or, at least, uniform interfaces. This choice allows the creation of a marketplace where competition in the provision of added-value services can take place.

To this extent some initiatives have been performed. Among them, the SHIWA simulation platform uses the gUSE technology to provide workflow interoperability across a number of workflow management systems [21]. Moreover, accessing data storages from gateways is a difficult and less investigated problem than running jobs in DCIs. Toolkit- or gateway-dependent solution in fact do not work if the applications requiring them are ported to other environments. A solution can be represented by DataAvenue [22], a bridging service capable of connecting jobs to various storage resources using different protocols and forward the accessed data via HTTP tunneling.

Widening the scope of our analysis, we can see that most of the toolkit are based on time-proved technologies (i.e. JavaEE), while it is possible to identify new technologies and different architectural approaches that are well received by the ICT community. The key aspects are: decomposition of monolithic Web services in a set of smaller, self contained, REST services (a.k.a. micro-services) that can be deployed independently; stateless services, which simplify load-balancing and resilience. By contrast, the client side is becoming state-full, with a richer set of functionalities and modular architecture. Client-side architecture are often based on evolution of the Model-View-Controller pattern [23], like in the widely adopted AngularJS[9] or, more recently, on Flow Based Programming (i.e. React/Flux[10] or the upcoming Angular 2[11])

The analysis of all these aspects led us to the following considerations:

- None of the previous toolkits provides a satisfactory solution for implementing the extensible and integrated UI able to support the coherent experiment configuration required by the HM community. On the other side most of the toolkits provide APIs to interact with the other components, in particular users, workflows and jobs management services. Therefore we decided to create a modular UI with reusable components based on AngularJS. We promote reuse of high-level components to enhance coherence among modules.

[9]The AngularJS website, https://angularjs.org/

[10]The Flux website, https://facebook.github.io/flux/

[11]The Angular 2 website, https://angular.io/

- We are migrating to a micro-service architecture, which grant us better resilience (load-balancing, fault tolerance) and more decoupled development lifecycle. We are still keeping high-value services like the gUSE DCI Bridge. Nevertheless, we will consider other possibilities, in particular the future evolution of the RADICAL Cybertools and the European Open Science Cloud for Research initiative [24].

- The experiments the DRIHM portal supports correspond to workflows with a simple structure. This, coupled with the lack of dedicated services for the HM community do not pose particular constraints on the adoption of a specific workflow management system except for the requirement of having a data provenance system associated with.

4 From the lesson learned to the DRIHM portal refactoring

The portal revamping has the goal to restructure its architecture, to improve scalability and maintainability. We decided to adopt an agile methodology, i.e. with frequent iterative steps [25]. We inherited the most important solutions provided by gUSE, as the DCI Bridge, coupled with new components to overcome current limits and providing new features.

Figure 2 shows the first step towards the new portal architecture. It consists of three layers: presentation layer is the DRIHM GUI, application layer exposes a set of (micro-)services, and foundation layer is responsible for interaction with DCIs. In details:

- Presentation layer is implemented as a modular Angular.js single page application, which offers dynamic load of UI modules. UI coherence is guaranteed by custom directives. It has been completely re-designed on the basis of developer and user feedback, collected on the first version of the DRIHM portal.

- Application layer main responsibility is user, data and workflow management. Currently we have replaced general purpose services (user management, data persistence and sharing) with node.js REST services. Workflow management is still based on the gUSE but we are evaluating alternatives, e.g. Taverna and custom implementation, because a strong requirement is the PROV compliance of the experiments.

- Foudation layer is required to dispatch jobs to the available computing resources. To grant flexibility we need adaptors for a wide set of middle-

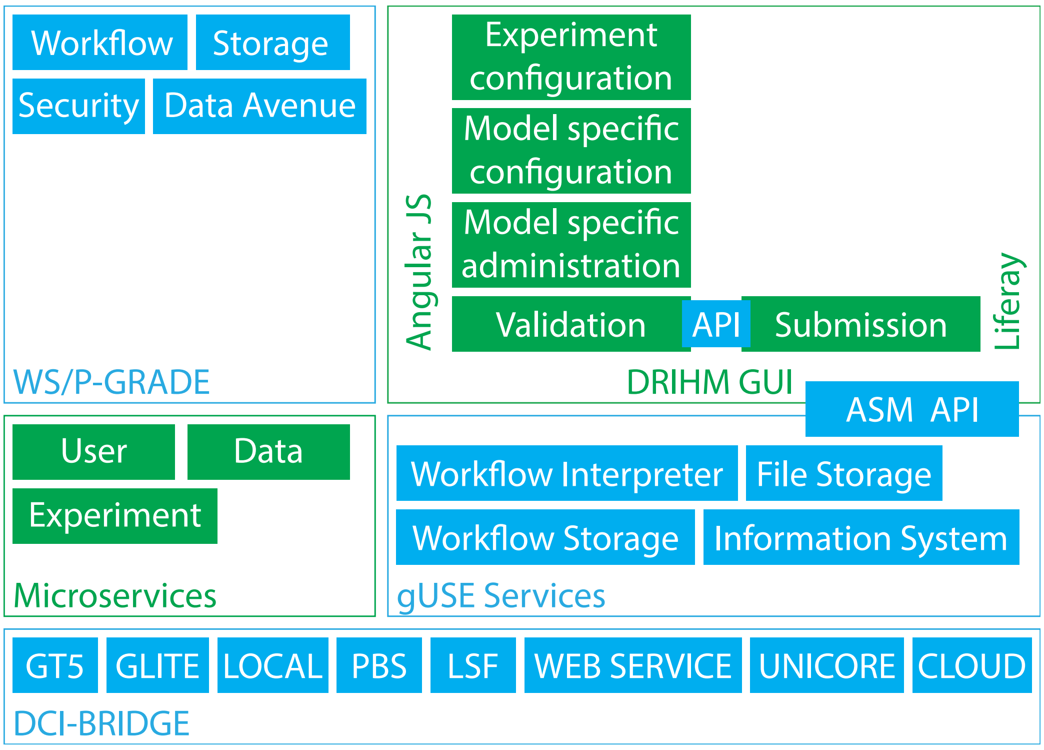


Figure 2: The DRIHM portal architecture. Green components have been created ad-hoc for DRIHM, while the blue ones are general-purpose inherited from gUSE framework.

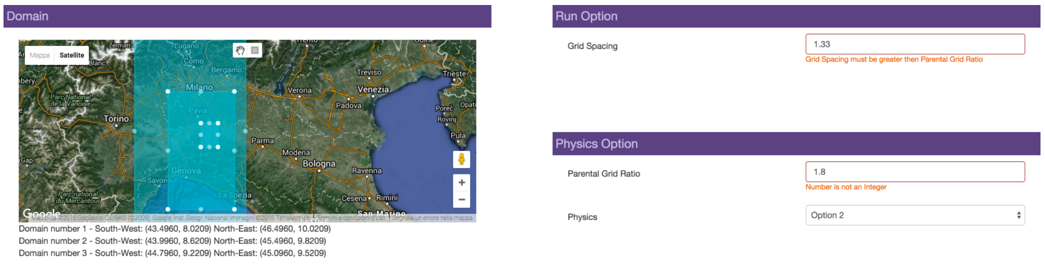
wares. DCI Bridge exposes a well accepted BES interface and supports a huge set of middlewares/computing resources. Unfortunately such wide scope makes it difficult to maintain and upgrade. As described in Section 3 we are monitoring the state-of-the-art and the modular architecture of DRIHM portal gives us the freedom to easily switch to an alternative toolset.

In the following subsections we analyze the key elements we have worked on, i.e. scalability, modularity, flexibility and the design of the new portal architecture.

4.1 Flexibility

HM simulations require a coherent configuration of all models involved in a simulation chain. Some constraints are quite obvious: the geographical domain and the simulated timeframe have to match; other constraints are related to input data selection (depending on data availability, data format, spatial and temporal resolution), model configuration (i.e. we can simulate river flow only on basins on which a model has been calibrated) etc. Model execution can be extremely time consuming and power hungry: a meso-scale, high resolution, 1 day meteorological simulation can takes 6-8 hours on a thousand CPU cores. It is wise to execute the simulation only if the whole simulation chain is properly configured and can correctly exploit the simulated rainfall. So we want to be able to perform consistency check ahead of model execution. To achieve this result, we associate some parameters (metadata) to each model configuration, and we use metadata to enforce coherency.

As a consequence, configuration of different model engines should be orchestrated, thus requiring some communication among model configuration modules. In the first version of the DRIHM portal we designed the GUI as a set of portlets (JSR 286 compliant), to guarantee a coherent integration with Liferay/gUSE. Direct communication among portlets is impossible, so we created a workflow among different portlets: at first the user select the model engines composing the simulation chain (with model-chain portlet), then, in case of hydrological simulations, she/he select one of the available model instances. Before moving to the model-specific parameters we have an additional portlet (chain-wide parameters portlet) asking for parameters that have to be shared with all models (i.e. timeframe). This approach grants us a coherent configuration but we have a complex dependency among portlets. If we need to move a parameter from model specific to chain-wide, we have to affect all the portlets.



Domain number 1 - South-West: (43.4960, 8.0209) North-East: (46.4960, 10.0209)
Domain number 2 - South-West: (43.9960, 8.6209) North-East: (45.4960, 9.8209)
Domain number 3 - South-West: (44.7960, 9.2209) North-East: (45.0960, 9.5209)

Run Option

Grid Spacing	1.33 Grid Spacing must be greater then Parental Grid Ratio

Physics Option

Parental Grid Ratio	1.8 Number is not an Integer
Physics	Option 2

Figure 3: Angular directives for model parameter configuration. Domain selection on the left, and meteo-specific parameters on the right.

While portlets are modular by nature, the dependency relation among model parameters introduces a tight coupling that could be relaxed with some kind of portlet communication. In the current version of the DRIHM portal, designed with Angular.js, we improve modular architecture, effectively decoupling model configurations: we provide a set of Angular directives (ie. for experiment domain selection, parameters configuration etc.), so each model UI can benefit from high-level components, able to cooperate and provide consistency check, together with a coherent UI. Figure 3 shows three Angular directives. Each directive is composed by a UI element, with all the logic supporting user interaction, a set of validators (i.e. domain selection portlet can be configured to enforce sub-domain nesting), and a communication mechanism to share the acquired value (so other analogous directives can be configured in a coherent way and/or hidden if the parameter has already be configured). Thanks to the intra-directives communication, we can avoid the chain-wide parameter selection step and improve User eXperience (UX). As a side effect, we perform consistency check at client side, decreasing server load. The actual submission is still performed via portlet, as shown in Figure 2, after the validation step.

4.2 Modularity

DRIHM portal has to interact with a complex DCI composed by High performance, Grid and Cloud resources managed by different institutions. Policies for resource management and software (operative systems, middleware, libraries) are extremely heterogeneous, and we have to adapt to evolving requirements. Having a large, monolithic, application we simply cannot evolve fast enough to keep interfaces to external resources up-to-date. Even a modular architecture with tightly coupled components, or with a lot of inter-modules dependencies can be problematic: it may force the update of a large set of modules due to dependencies, with related maintainability issues. In gUSE the DCI Bridge offers a nice modular architecture, but update policy focus on providing a fully integrated new versions, so to get a new DCI module you are somehow forced to update the whole gUSE framework.

A modular architecture that really give us the freedom to evolve and adapt our interfaces to the computing resource is thus required. We need loosely coupled components that can be updated independently. For this reason we focused on a microservice-based architecture[12]. A microservice architecture is composed by small components. Each component has a clear, and limited, scope and provides all the features (database, business logic, interfaces) to fulfil its duties. This lead to fully decoupled services, with independent lifecycle, cooperating with REST or messaging interfaces. It is possible to load balance only the services that are under stress, and save resources otherwise.

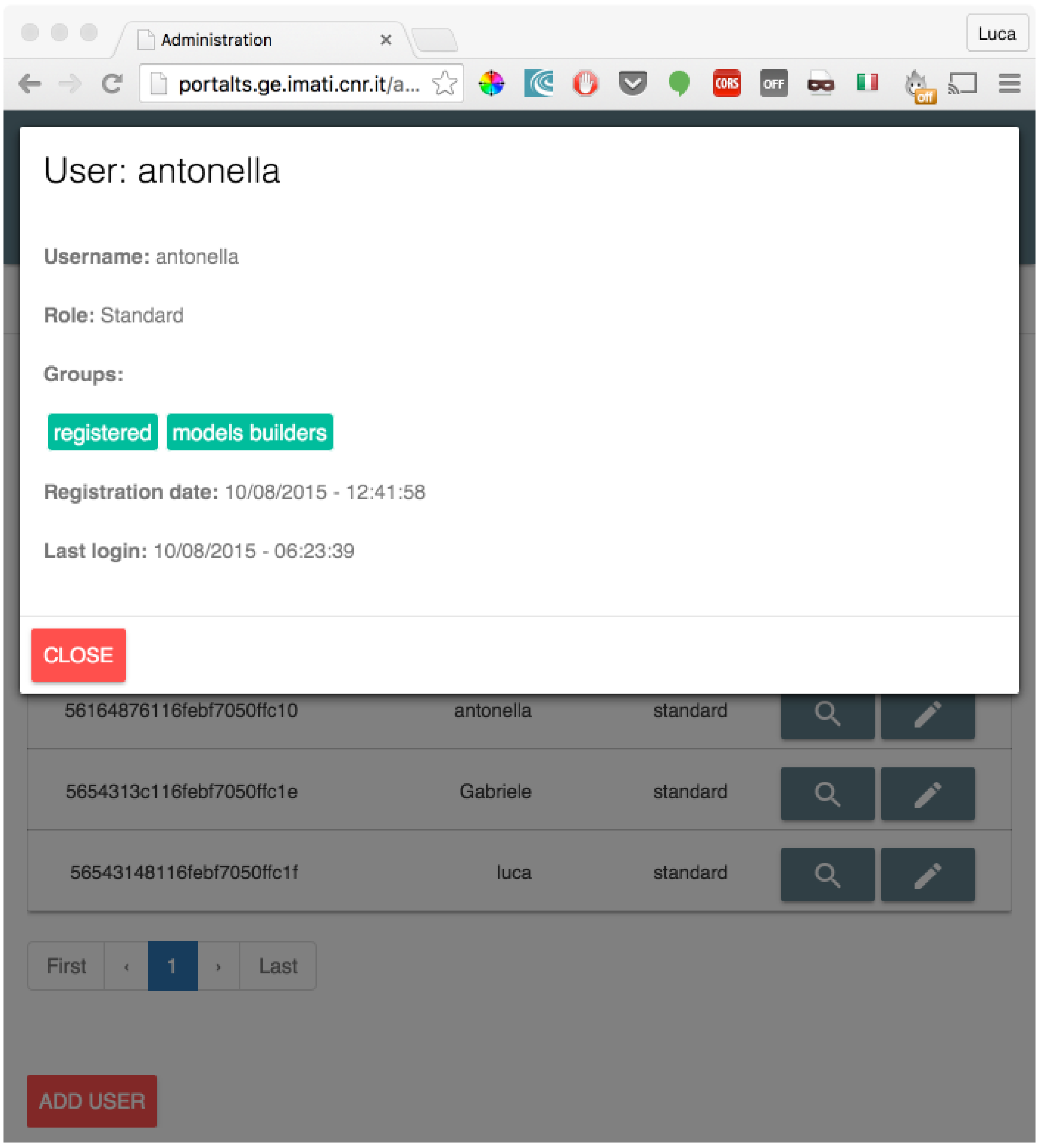
We are migrating to a set of independent microservices exposing REST interface for user management, experiment configuration, data persistency, data analysis. Currently deployed as linux virtual machines, we are considering a even lighter container like Docker as in Agave. Figure 2 shows that the new UI is still tightly coupled with the ASM API for the workflow management, while it is loosely coupled with services for user management, data management (persistence and sharing) and experiment configuration management (grouped in the Microservices box).

Microservices are implemented as node.js + express applications. We have dependencies among them: the data management micro-service is responsible for the persistence of experiment configurations, while user and group management provides authorization services to data and experiment management, i.e. given a user identity, it can authorize the access to a requested document or experiment configuration. Figure 4 shows the angular.js GUI of the micro-service for user and group management. Users can login with different credentials (username and password, facebook id, OAuth), and be associated with different groups. Their group affiliation is the basis for user authorization.

4.3 Scalability

gUSE is architected as a JavaEE state-full application hosted in a Tomcat webapp container. Concurrency is thread-based, thus exploiting a thread pool. Peak activities, e.g. after an high impact meteorological event as a flood, result in a heavy load of the system. We tried to overcome this issue

[12]http://martinfowler.com/microservices/



User: antonella

Username: antonella

Role: Standard

Groups:

registered models builders

Registration date: 10/08/2015 - 12:41:58

Last login: 10/08/2015 - 06:23:39

CLOSE

56164876116febf7050ffc10	antonella	standard
5654313c116febf7050ffc1e	Gabriele	standard
56543148116febf7050ffc1f	luca	standard

Figure 4: User management is a node.js+express micro-service with an angular.js responsive web interface, a user can belong to several groups.

by a proper tune of the thread pool size to balance concurrency and consumed memory. Moreover we tried to configure the Tomcat Java Virtual Machine with different maximum memory allocation pool size, finding a sweet-spot between 2 and 3 GB. Also the load balancing on multiple servers can be helpful, but is complicated by the state-full nature of the application. We can somehow load-balance and keep the state-full architecture by using sticky sessions (so a user is consistently always served by the same server instance), but this affect the balancing, and does not provide fault tolerance.

For these reasons we switched to a light, state-less, backend based on node.js with fully async management of concurrent user requests and data access. While node.js is inherently single-threaded, we can easily get fault-tolerance and load balancing (either at node level with Passenger and at cluster level, with the benefit of autoscaling). In fact, in a stateless scenario, a user request can independently get served by a random server instance. Session is kept client side and data are consistently fetched from the shared datastore. A simple architecture based on MongoDB + node.js is providing excellent response time. In case of increasing workload we are considering a data cache implemented with an in-memory database (i.e. Redis).

Beside node.js, web application relying on asynchronous programming are gaining momentum. Asynchronous services are the main feature of web frameworks like Play! or Vert, and the largest improvement of the recent JavaEE infrastructure (with Java NIO and Servlet 3.1).

4.4 The new portal architecture

A simulation chain has to deal with several kind of data: model binaries, model calibrations, configuration parameters, input data (with related access policies), intermediate and final results. Since gUSE data management was really basic, we introduced in the DRIHM infrastructure a few repositories for keeping model binaries, model instances, input and simulated data. Access rules are extremely simple: only the data owner in some cases (i.e. configuration parameters) or community-wide availability (i.e. input and calibration data).

In details the DRIHM virtual community is composed of different users, such as HM researchers, public organizations and citizen scientists interested in HMR and related Earth science disciplines. In particular, three users categories are envisaged for the DRIHM portal: citizen scientists, scientists and expert scientists:

- Expert scientists: they have the right to insert model instances and input data through dedicated services for making them available to the virtual community. This because, for example, the calibration of a basin for executing hydrological simulation has to be inserted by expert hydrologists, that can assure the correctness of data and therefore the scientific validity of simulation results all the scientists can perform.

- Scientists: they have the right to upload new input data for their own experiments.
- Citizen scientists: they have the right to execute simulations only using pre-defined scenarios.

In the new release of the portal we are going to grant user- and group-level data visibility. A persistence microservice, in cooperation with user and group microservice, provides services for document (JSON) data persistence, versioning, access policies and sanitization. This microservice has been designed to give the freedom to set accessibility rules to each document, and thus enable data sharing. We are willing to provide a flexible rules for setting simulated data visibility.

Beside this, the portal revamping has the further goal to introduce two new user-driven features: the multi-model ensemble capabilities and collection of provenance data.

The DRIHM science gateways keeps model configurations as model namelists (i.e. text files listing parameters as key-value pairs). Such information can be used to reproduce an experiment in the DRIHM platform. Each namelist is model specific. To introduce multi-model ensemble capabilities (i.e. the ability to perform a simulation on the same domain and timeframe with different model engines) we have to support the user in coherent configuration of all model engines. Given that, multiple simulations can be compared and analyzed, in order to better capture and understand extreme events. Since a large set of parameters are common to every models (i.e. geographical domain, timeframe, ...) we want to keep a higher-level representation of model configuration parameters (metadata), so that we can share common parameters among different models. UI directives are extremely helpful in providing this feature.

The second strong user request is to model namelist to full provenance information in PROV standard[13]: provenance data can be used to track back the model chain, model engines, model calibrations and parameters that produced a simulated result. gUSE has been enriched with a module for provenance data acquisition, but is not yet able to provide a full set of information. We are looking at Taverna workflow management system, which

[13]The PROV family of documents http://www.w3.org/TR/prov-overview/

is PROV compliant, or at a custom workflow management implementation adopting Flow-Based-Programming.

5 Conclusions and Future Directions

The first results presented in this paper, is the refactoring of the DRIHM science gateway for hydro-meteorological simulations. We started from the experience gained developing the first version of the DRIHM portal, based on the gUSE toolkit, and we evolved to a more robust yet modular architecture.

To achieve such result, we widened our analysis to the science gateway frameworks and recent general-purpose technologies, architectural patterns and best practices adopted in the development of enterprise web application. In general, the decoupling of the presentation, application and foundation levels of a gateway and, whenever possible, the split of services in smaller, more manageable components give science gateway developers more control and a smoother software development lifecycle. As a side benefit, working on decoupled components, it is possible to adopt the best technologies for each component. We are thus evolving the DRIHM portal from a monolithic toolkit to an ecosystem of components, that are able to cooperate. The refactoring process is ongoing, and further services have been designed.

In conclusion, current science gateway frameworks should inherit from the new approaches in developing enterprise web applications a short release cycle warranted by a modular architecture, better resilience and performance from small services, cooperating trough REST interfaces, often referred as micro-services, and more integrated UIs, still modular but more capable.

Given our experience, we suggest to exploit services coming from a single toolkit for the fast prototyping of a science gateway, while the architecture can evolve with the introduction of different components, cooperating trough standard interfaces. This is extremely relevant since input data, computational resources, middlewares and software components are produced by different entities that have independent policies, roadmap and goals. Moreover, to tackle these issue there is the need of a strong interplay between ICT and the domain-specific communities, plus initiatives aiming at the delivery of a consolidated set of components for science gateways development and the definition of a sustainable model to evolve the components addressing partially overlapping and similar needs.

References

[1] Seneviratne SI, Nicholls N, Easterling D et al. Changes in climate extremes and their impacts on the natural physical environment. In: Managing the Risks of Extreme Events and Disasters to Advance Climate Change Adaptation, Special Report of the Intergovernmental Panel on Climate Change, 2012; 109-230.

[2] Pappenberger F, Thielen J, Del Medico M. The impact of weather forecast improvements on large scale hydrology: analysing a decade of forecasts of the european flood alert system. Hydrological Processes; 25 (7):1091-1113, 2011.

[3] G. Theurich, C. Deluca, T. Campbell et al. The Earth System Prediction Suite: Toward a Coordinated U.S. Modeling Capability. Bulletin of the American Meteorological Society, in press.

[4] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, Á. Balasko, K. Karóczkai, and I. Márton, Ws-pgrade/guse generic dci gateway framework for a large variety of user communities, Journal of Grid Computing, vol. 10, pp. 601 - 630, 2012.

[5] Daniele D'Agostino, Emanuele Danovaro, Andrea Clematis, Luca Roverelli, Gabriele Zereik, Antonio Parodi and Antonella Galizia, Lessons learned implementing a science gateway for hydro-meteorological research. Concurrency and Computation: Practice and Experience, article first published online : 20 SEP 2015. DOI: 10.1002/cpe.3700

[6] D'Agostino D, Clematis A, Galizia A et al. The DRIHM Project: A Flexible Approach to Integrate HPC, Grid and Cloud Resources for Hydro-Meteorological Research. Proceedings of the International Conference For High Performance Computing, Networking, Storage and Analysis 2014 (SC14), 2014; 536-546.

[7] Danovaro E, Roverelli L, et al. Setup an hydro-meteo experiment in minutes: the DRIHM e-infrastructure for hydro-meteorology research. Proceedings of the International Conference on e-Science (e-Science), 2014; 47-54.

[8] Hally A, Caumont O, Garrote L et al. Hydrometeorological multi-model ensemble simulations of the 4 November 2011 flash-flood event in Genoa, Italy, in the framework of the DRIHM project. Natural Hazards and Earth System Sciences; 15 (3): 537-555; 2015.

[9] Hill, C., C. DeLuca, V. Balaji, M. Suarez, and A. da Silva (2004). Architecture of the Earth System Modeling Framework. Computing in Science and Engineering, Volume 6, Number 1, pp. 18-28.

[10] J. Kruger, R. Grunzke, S. Gesing, S. Breuers, A. Brinkmann, L. de la Garza, O. Kohlbacher, M. Kruse, W. E. Nagel, L. Packschies, R. Muller-Pfefferkorn, P. Schafer, C. Scharfe, T. Steinke, T. Schlemmer, K. D. Warzecha, A. Zink and S. Herres-Pawlis: The MoSGrid Science Gateway -ĂŞ A Complete Solution for Molecular Simulations, Journal of Chemical Theory and Computation, 2014, 10(6), 2232-2245.

[11] Goff, Stephen A. et al., "The iPlant Collaborative: Cyberinfrastructure for Plant Biology," Frontiers in Plant Science 2 (2011).

[12] Becciani U., Sciacca E., Costa A., Massimino P., Pistagna C., Riggi S., Vitello F., Petta C., Bandieramonte M. and Krokos M. (2015), Science gateway technologies for the astrophysics community, Concurrency Computat.: Pract. Exper., 27, pages 306-ĂŞ327, doi: 10.1002/cpe.3255

[13] Lawrence K, Zentner M, Wilkins-Diehr N, Wernert J, Pierce M, Marru S, Michael S. Science gateways today and tomorrow: positive perspectives of nearly 5,000 members of the research community. Concurrency and Computation: Practice and Experience 2015; 27(16):4252-4268.

[14] Marlon E. Pierce, Suresh Marru, Lahiru Gunathilake, Don Kushan Wijeratne, Raminder Singh, Chathuri Wimalasena, Shameera Ratnayaka and Sudhakar Pamidighantam, Apache Airavata: design and directions of a science gateway framework. Concurrency Computat.: Pract. Exper. 2015; 27:4282-4291

[15] P. Dziubecki, P. Grabowski, M. Krysiński, T. Kuczyński, K. Kurowski, D. Szejnfeld, Easy Development and Integration of Science Gateways with Vine Toolkit, Journal of Grid Computing: Volume 10, Issue 4 (2012), Page 631-645

[16] Dooley, Rion, et al. "Software-as-a-Service: The iPlant Foundation API", 5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS). IEEE, 2012.

[17] M. McLennan, R. Kennell, "HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering," Computing in Science and Engineering, 12(2), pp. 48-52, March/April, 2010.

[18] SAGA: A Standardized Access Layer to Heterogeneous Distributed Computing Infrastructure Andre Merzky, Ole Weidner, Shantenu Jha Software-X, 2015 DOI: 10.1016/j.softx.2015.03.001

[19] Aiftimiei, C.; Aimar, A. ; Ceccanti, A. ; Cecchi, M. ; Di Meglio, A. ; Estrella, F. ; Fuhrmam, P. ; Giorgio, E. ; Konya, B. ; Field, L. ; Nilsen, J.K. ; Riedel, M. ; White, J., Towards next generations of software for distributed infrastructures: The European Middleware Initiative. E-Science (e-Science), 2012 IEEE 8th International Conference on, pp. 1-10, 2012. DOI: 10.1109/eScience.2012.6404415

[20] EMI Roadmap and DCI Collaborations, Deliverable EMI-NA1-D1.4, 2010, http://cdsweb.cern.ch/record/1277542

[21] Gabor Terstyanszky, Tamas Kukla, Tamas Kiss, Peter Kacsuk, Akos Balasko, Zoltan Farkas, Enabling scientific workflow sharing through coarse-grained interoperability, Future Generation Computer Systems, Volume 37, July 2014, Pages 46-59, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2014.02.016.

[22] Hajnal A, Marton I, Farkas Z, Kacsuk P. Remote storage management in science gateways via data bridging. Concurrency and Computation: Practice and Experience 2015; 27(16):4398-4411.

[23] Krasner G. E., Pope S. T. A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80. J. Object Oriented Program. 1(3):26-49 SIGS Publications, Aug./Sept. 1988

[24] Position Paper: European Open Science Cloud for Research. http://dx.doi.org/10.5281/zenodo.32915

[25] Beck, Kent, et al. Manifesto for Agile Software Development. [Online] 2001. http://agilemanifesto.org.