WILEY

Toward a reference architecture based science gateway framework with embedded e-learning support

Gabriele Pierantoni[1] | **Tamas Kiss[1]** | **Alexander Bolotov[1]** | **Dimitrios Kagialis[1]** | **James DesLauriers[1]** | **Amjad Ullah[1]** | **Huankai Chen[1]** | **David Chan You Fee[1]** | **Hai-Van Dang[1]** | **Jozsef Kovacs[1,2]** | **Anna Belehaki[3]** | **Themistocles Herekakis[3]** | **Ioanna Tsagouri[3]** | **Sandra Gesing[4]**

[1]School of Computer Science and Engineering, University of Westminster, London, UK

[2]Parallel and Distributed Systems Laboratory, SZTAKI, Budapest, Hungary

[3]Institute of Astronomy, Astrophysics, Space Applications and Remote Sensing, National Observatory of Athens, Athens, Greece

[4]Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, Indiana, USA

Correspondence
Tamas Kiss, University of Westminster, London, UK.
Email: kisst@wmin.ac.uk

Abstract

Science gateways have been widely utilized by a large number of user communities to simplify access to complex distributed computing infrastructures. While science gateways are still becoming increasingly popular and the number of user communities is growing, the fast and efficient creation of new science gateways and the flexibility to deploy these gateways on-demand on heterogeneous computational resources, remain a challenge. Additionally, the increase in the number of users, especially with very different backgrounds, requires intuitive embedded e-learning tools that support all stakeholders to find related learning material and to guide the learning process. This paper introduces a novel science gateway framework that addresses these challenges. The framework supports the creation, publication, selection, and deployment of cloud-based reference architectures that can be automatically instantiated and executed even by nontechnical users. The framework also incorporates a knowledge repository exchange and learning module that provides embedded e-learning support. To demonstrate the feasibility of the proposed solution, two scientific case studies are presented based on the requirements of the plasmasphere, ionosphere, and thermosphere research communities.

KEYWORDS

cloud orchestration, embedded e-learning support, microservices, reference architectures, science gateways

1 | INTRODUCTION

Science gateways are typically defined as "a community-specific set of tools, applications, and data collections that are integrated together via a web portal or a desktop application, providing access to resources and services of distributed computing infrastructures" (e.g., clouds, grids or supercomputing resources).[1] The concept of science gateways is almost 20 years old and developed gradually as large-scale computational infrastructures evolved from grid computing[2] resources to clouds. Nowadays, there is a plethora of science gateways available supporting scientists in a wide range of disciplines. However, the original motivation of gateways remains the same: replacing command-line

interfaces and providing user-friendly access to very complex computational and data resources in order to support scientific research or industry applications.

While science gateways bring tremendous benefits for end users by simplifying access to and hiding the complexity of the underlying distributed computing infrastructure, building such gateways quickly and efficiently has always been a challenge. The first science gateways were custom developed for their specific targeted user communities. While such an approach is likely to result in fully customized gateways that fit the requirements of the end users, it also requires significant development and maintenance efforts. As all gateways are different—targeting various applications and user communities—their development needed to be started almost from scratch, resulting in a duplication of effort and limited reusability.

To overcome this problem, science gateway frameworks, such as WS-PGRADE,[3] HUBzero,[4] or the Catania Science Gateway Framework[5] were introduced. Such frameworks provide ready-made building blocks that significantly speed up the development of new science gateway instances. For example, WS-PGRADE provides a workflow engine and an associated graphical user interface (GUI) that enable application developers to construct complex workflows and map their computation to heterogeneous grid and cloud computing resources easily. End users can select and execute such workflows from associated workflow repositories. To further reduce the complexity for end users, WS-PGRADE also supports the creation of more customized end user interfaces via its end-user mode or by the application specific module API (application programming interface), which simplifies the creation of fully customized web interfaces. However, even with such tools available, the development of a new science gateway instance or the extension of a science gateway with a new custom user interface for a specific community still requires significant development effort. Once the new GUI is available, it needs to be integrated into the gateway, which typically requires a full restart by its administrator before it becomes accessible. Science gateways typically offer a predefined set of functionalities, extending them with new applications and user interfaces is not straightforward.

The rise of cloud computing, containerization and cloud-native solutions, on the other hand, provides new opportunities and paradigms for science gateway developers. One of these new paradigms is cloud-based reference architectures. While commercial vendors, for example, Amazon[6] refer to reference architectures as vendor specific solutions that simplify cloud-based application development, in this paper we take a more generic, cloud-native and fully vendor independent approach. In our definition, similarly to some earlier work such as,[7] a reference architecture is a complex set of interconnected microservices that can be automatically deployed and that implements a certain end-to-end functionality for the end users. Such reference architectures are composed of multiple application components and can be automatically deployed and managed at run-time by various cloud orchestrator tools (e.g., Kubernetes,[8] Terraform,[9] Occopus,[10] or MiCADO,[11] among many others). A reference architecture is described in the form of a deployment descriptor that is either specific for the targeted orchestrator (e.g., a Kubernetes manifest) or standardized (e.g., using the topology and orchestration specification for cloud applications (TOSCA)[12] language specification by OASIS[13]). A reference architecture can include various components, such as generic or custom GUIs, data analytics, machine learning, simulation or other scientific applications, databases, and any other components (application-level firewalls, data converters, load balancers, etc.) that are required to realize a particular user scenario. Moreover, due to its approach of combining multiple microservices expressed in a single deployment descriptor (typically a YAML Markup Language file), new reference architectures can be created by combining existing building blocks, either by the application developers or even automatically. Although reference architectures are not limited to be utilized in relation to science gateways (such approach can be applied for every typology of system/application built by deploying a number of interconnected services), in this paper we specifically explore how such architectures can ease the development and operation of science gateways. According to our knowledge, no science gateway framework exists currently that fully utilizes such approach.

While the focus of science gateways has always been user-friendly access to resources, with the emergence of new paradigms such as open science and citizen science, there is an even stronger motivation to widen individual user communities. Therefore, science gateways should also provide learning tools and environments to help less experienced users getting familiar with the scientific and technological background. Some gateways, for example, the NanoHUB Gateway,[14] already provide such learning resources. However, the learning environment in such gateways does not support the specifications of shared conceptualization in the form of models, learning curves and ontologies and is restricted to offering learning material (e.g. lecture notes, video recordings, exercises, and tutorials) in a static and linear way, following traditional learning approaches. The way knowledge is structured is often presented via static means (such as a web page) and cannot easily be changed to mirror the evolution of technologies and research efforts. Such a linear and "one size fits all" approach does not provide adequate support for describing structured knowledge maps and individual learning curves and makes it harder for potential users with very different backgrounds and profiles—from experts to everyday citizens—to become successful gateway users.

In this paper, we present a new, generic concept for developing science gateways based on the two key concepts described above: the use of reference architectures and a comprehensive knowledge management system. Such combined use of these two paradigms allows us to offer an extensible and flexible infrastructure for scientific research and to present and maintain the related knowledge so that it is consistent among the different actors involved.

The motivation for this work is also rooted in the PITHIA-NRF (plasmasphere ionosphere thermosphere integrated research environment and access services: a network of research facilities) project[15] that is funded by the European Commission. In PITHIA-NRF, we are building an

e-Science Center that enables the targeted research communities to share their various applications and databases, and utilize various machine learning, artificial intelligence and data analytics solutions to get scientific insights into their data. Due to the large and dynamically growing number of applications and distributed data sources, and the need to custom-develop and deploy applications quickly and flexibly for the targeted scientists, a traditional science gateway concept would not be suitable. On the other hand, an approach based on reference architectures enables scientists to utilize already published reference architectures, compose new ones by combining prepublished components, and deploy them on-demand in private or public cloud infrastructures. Therefore, one of the major contributions of this paper is to present a generic science gateway framework that supports the creation, storage, selection, and execution of reference architectures within PITHIA-NRF, and beyond. Additionally, to further support scientists, application developers, e-Science Center operators, or even to enable citizen scientists or the general public to understand some of the technical and scientific concepts, we also propose to incorporate intuitive graph-based e-learning support into PITHIA-NRF, capable of describing models, learning paths, and ontologies. The principles of science gateways with embedded ontology-based learning support[16] are further extended and incorporated into the generic science gateway framework that we are suggesting in this paper.

The rest of this paper is structured as follows. Related work on science gateways with specific focus on the dynamic generation of user interfaces, reference architectures and ontology-based e-learning support is provided in Section 2. Section 3 summarizes the PITHIA-NRF project and the requirements of its user communities that serve as the primary motivation for our work. The generic architecture of the science gateway framework based on the reference architecture model and ontology-based e-learning support is introduced in Section 4. Section 5 describes the current implementation of the components of the proposed framework. Section 6 details two case studies, inspired by PITHIA-NRF user communities, where reference architectures have been designed, implemented, launched, and evaluated. Finally, Section 7 concludes the paper and outlines future work.

2 | RELATED WORK

This section provides a short overview of related work from two different perspectives. First, the use of reference architectures in the design and development of science gateways is summarized, followed by an overview of related efforts in the area of embedded e-learning and ontology support.

2.1 | Science gateways and reference architectures

2.1.1 | Early science gateways: Job submission and batch systems

The concept of science gateways was developed about 20 years ago when it became obvious that grid computing and batch systems would enable applications of high-performance (HPC) and high-throughput (HTC) computing infrastructures on a large scale, but the uptake by research communities was slow to start. The requirement of using the command-line and becoming acquainted with the technical details of complex distributed research infrastructures formed a hurdle for researchers and educators who were not necessarily IT specialists. While different communities had different requirements in terms of domain-specific tools, data, and workflows, the building blocks in the backend were the same and were independent of the research domain (e.g., job submission to research infrastructures, which includes security, logging, and monitoring).

Quite a few projects and activities started standardizing job submission services and batch systems, representing the first steps in the direction of reference architectures. For example, the open grid forum (OGF)[17] developed standards for job description: fields for defining different layers of access and information, such as architecture and hardware dependencies, software dependencies, data dependencies, input and output and run-time requirements, as well as information about batch systems. Examples for standards include:

- the distributed resource management application API (DRMAA)[18] that defines an API to distributed research infrastructures,
- the job submission description language (JSDL)[19] that defines the requirements of computational jobs for submission to resources in grid environments,
- the grid laboratory uniform environment (GLUE)[20] that defines a conceptual information model for grid entities using the natural language and UML class diagrams,
- the Globus resource specification[21] that defines grid resources, including computational job information, and
- the simple API for grid applications (SAGA)[22] that defines high-level interfaces for common grid components.

Many science gateways and science gateway frameworks use the above-mentioned standards and solutions when implementing their services, making resource access, job description and submission, and other basic functionalities independent of middleware and resources (e.g., DECIDE[23] and FutureGateway[24] use SAGA, WS-PGRADE and HUBzero[25] use JSDL). Such an approach makes it easier to extend or adopt science gateways to a wide range of computing infrastructures and requirements. However, mainly due to technology limitations, it was not possible to dynamically instantiate and deploy such complex solutions automatically and with minimum or no user interaction.

2.1.2 | Comprehensive reference architectures for science gateways

A small number of projects attempted to define full reference architectures for science gateways. However, these efforts stayed mainly at a conceptual level and did not support fully automated instantiation and deployment.

gCube's[26] concept to create science gateways or virtual research environments (VREs) is a successful example for an implementation of a reference architecture on top of Liferay.[27] gCube offers core services for resource management, security, and managing a science gateway. Additionally, researchers are able to select different modules for working with geospatial or biodiversity data. gCube is very flexible and widely used, that is, D4Science[28] is built on gCube. It still needs quite some manual work for the orchestration of the services though and does not allow for dynamical instantiation.

The open science framework (OSF)[29] is a widely used science gateway that connects seamlessly to a variety of repositories and data storage services such as Google drive and Zenodo. OSF focuses on sharing data services and supports assigning DOIs to data objects, for example. It offers a RESTful API for its services. It does not support to submit jobs or simulations though.

Workflow-enabled science gateways[30] such as WS-PGRADE and Taverna have developed building blocks of services for workflow orchestration and job submissions and thus, have often very similar architectures. The group behind Taverna has created the community space myExperiment.org[31] where people can exchange their workflows from a variety of workflow management systems and share their research results and methods. While this exchange and sharing is a crucial step into open science and teaching, researchers still need to set up or use the corresponding workflow management systems to reuse the workflows. The workflow management systems are not automatically adapted to the specific use case.

The VRE4IG European[32] project has suggested a reference architecture for science gateways and VREs defined via the multitiers view approach and built upon the design of distributed information systems.[33] The project suggested three logical tiers in a VRE system: the application tier, the interoperability tier and the resource access tier. The utilization of different tiers resulted in the definition of building blocks based on microservices. The goal was to ensure that an architecture is easily expandable for adding new tools, supports reusability of existing tools and services, is domain agnostic, supports standardized services and is flexible for integration with novel standards.

The Science Gateways Community Institute has also developed a reference architecture[34] focusing on the features needed for the full research lifecycle, from services to authenticate users, to publications and the sharing of results. This reference architecture model provides a high-level definition of common science gateway components and the way science gateways support scientific research.

All these previous efforts have formed the basis for developing valuable services for reference architectures for science gateways or reference architectures themselves. While our approach aims at the same goals, it goes a huge step further when supporting research communities. Our reference architecture framework not only defines different layers depending on the required functionalities and features, but also, using the latest advancements in cloud orchestration, container technologies and microservices-based architectures, deploys them automatically on the targeted distributed computing infrastructures and manages their entire lifecycle.

2.2 | Science gateways with embedded e-learning support

Table 1 provides an overview of e-learning and ontology related features in science gateways and some on-line data repositories and learning platforms.

There are currently a very limited number of gateways that include significant learning/educational components. After extensive research, we only found three notable examples: the nanoHUB Gateway,[14] MyGeoHub,[35] and Wolfram.[36] However, these gateways all follow a traditional, linear approach, to learning and do not address learners' individual styles.

2.2.1 | Classical e-learning support in science gateways

The education component of NanoHUB (nanoHUB-U) is a digital repository of various educational resources which offers learning material in the classical form of various digital media, for example lecture notes or video recordings. MyGeoHub contains online courses

TABLE 1 Overview of e-learning features

Solution/platform	Science gateway	Learning component	Ontology
NanoHUB	✓	✓	✗
Wolfram	✓	✓	✗
European School Education Gateway	✗	✓	✓
BBC gateway	✗	✓	✓
Humanitarian assistance and disaster recovery (HADR)	✗	✗	✓
ENVRI reference model	✗	✗	✓
MyGeoHub	✓	✓	✗

related to hydrological modeling, offering SWATShare—a comprehensive online environment for sharing, executing, and visualizing soil and water assessment tool (SWAT) models.[35] The Wolfram ecosystem is a proprietary set of software with targeted licenses for all tiers of education, including scientific research. Its leading product, mathematica,[37] has been used by educators as a computer algebra system (CAS) for over 30 years. Only very recently has the possibility emerged for building open science gateways around Wolfram. The Wolfram Engine was released in 2019 for developers of projects in preproduction. For learning, Wolfram offers the Wolfram Education Portal, which, similarly to NanoHub-U, takes a classical approach to learning: a dynamic textbook, a lesson plan, and an interactive demonstration are some of the common standard features offered on the portal. We believe that perhaps, due to being made available for open development only recently, Wolfram does not have any community that has established strong open gateways around it yet. Furthermore, any integration with the licensed educational portal or licensed tools for educators does not currently appear to be possible.

2.2.2 | Ontology-based e-Learning platforms

Besides science gateways with learning support, there are numerous platforms that are dedicated specifically to learning. Here we provide an account of a rather small number of those that utilize ontologies. A notable example is the European School Education Gateway.[38] This educational gateway provides a "toolkit to support the exchange and experience among school practitioners and policy makers." Although ontology is not explicitly mentioned in the description of the platform, our analysis of its services has shown that the material published for users is structured in the form of an ontology. Another example of an explicit use of ontologies in an educational gateway is provided by the BBC.[39] Here one can find a collection of ontologies related to various BBC activities—politics, sport, education, and so forth. A dedicated education related ontology is the "curriculum ontology" which presents a data model for "formally describing the national curricula across the UK." This ontology, according to its description, organizes various learning resources and allows users to discover content via the national curricula. The published material is structured by the metadata classification to "topic, field of study and programme of study" that are common in the curriculum domain. From the technical point of view, the ontology is written in resource description framework (RDF)[40] and is linked to dedicated distinct vocabularies, the curriculum ontology and the Schema.org educational vocabulary contributed by Dublin core metadata initiative (DCMI).[41]

Although the above learning gateways incorporate ontologies, these are not necessarily science gateways by the definition of the IEEE technical committee on scalable computing,[1] and they do not offer a generic gateway architecture with the capabilities to build subject-specific applications and their related learning material.

2.2.3 | Ontology-based research platforms

Finally, we mention two further platforms summarized in Table 1—The humanitarian assistance and disaster recovery (HADR)[42] and environmental research infrastructures reference model (ENVRI RM). HADR offers services to discover and access relevant services and smart city ICT assets. This enables the exchange of data between multiple HADR agencies based on a common ontology. The aim of the ENVRI RM is to provide a framework for specifying and building the data management services required by the environmental and Earth sciences research infrastructures. This platform utilizes an ontology framework designed to facilitate analysis, classification, and validation of the design of a research infrastructure.

Based on the analysis above, we can state that there is an emerging tendency for science gateways to include learning tools, and there is also evidence that ontologies are already used in several platforms that offer e-learning. These two observations make our approach even more practically useful as we aim at filling the gap in the development of science gateways: we provide a generic platform which can be used for enriching gateways with both learning and ontologies.

3 | THE PITHIA-NRF PROJECT

The Earth's ionosphere, thermosphere and plasmasphere are a coupled system of "spheres" governed by electromagnetic coupling and thermospheric wind dynamics, that lead to plasma variability of long- and short-time scales and ranges, and to plasma irregularities.[43,44] This complex and variable environment in the near-Earth space, is the source of many scientific, operational, societal, and environmental challenges that affect the smooth and uninterrupted operation of technological systems such as high frequency (HF) radio communication and geolocation systems, ground- and satellite-based augmentation systems, and space-based communications, as well as communications between the Earth and ground stations (or rovers) on the Moon, Mars, and other planets, low-frequency radio astronomy and synthetic-aperture radars observations.

While the scientific community understands the broad features of this coupled system, it lacks the depth of understanding regarding the variability that would allow us to build models with real predictive power. To further advance research, the PITHIA network of research facilities (PITHIA-NRF) project aims to build a European distributed network integrating observation facilities, data collections, data processing tools and prediction models dedicated to ionosphere, thermosphere and plasmasphere research. PITHIA-NRF is designed to provide organized access to findable, accessible, interoperable, re-usable (FAIR) data, standardized data products, training and innovation services. PITHIA-NRF paves the way for new observation technologies, procedures and tools for the end-to-end transition of research models to applications, linking best-in-class research and development facilities to provide seamless multitechnology services.

PITHIA-NRF has the ambition to support innovative scientific developments, including a better understanding of the physical mechanisms responsible for the plasma dynamical processes and the development of realistic predictive models. To achieve these ambitious goals, it is necessary to develop tools that support experimentation with empirical, physics-based models, simulations, and model validation using certified quality data.[45,46] A central component of the project is the PITHIA-NRF e-Science Center that provides scientists with a central access point to shared data facilities and to run various applications evaluating a large variety of models. Based on requirements collected from the community, the PITHIA-NRF e-Science Center should be designed according to the following main principles:

- offer a flexible and user-friendly framework capable of catering for users with different levels of expertise, interests, and preferences,
- provide a graphical user interface capable of offering the required information in the most intuitive way by allowing the users to define their own preferences through a set of policies,
- provide individualized support for users depending on their level of expertise, specific interests and the storage and computational resources that they can/are willing to access,
- offer navigation tools supporting users with different profiles to utilize the resources (data sets, applications, results, and processes) most suitable to them,
- provide access to replicated and cleaned datasets that facilitate better and more sophisticated knowledge discovery when compared to the original raw data sets,
- be as comprehensive as possible in order to facilitate the utilization of a wide range of heterogeneous data sets, applications, results and processes,
- provide access to a wide range of heterogeneous computational resources in order not to be dependent on one particular technology or resource provider,
- be implemented based on widely used open-source state of the art technologies, with the necessary integration and extensions.

PITHIA-NRF's concept revolves around a series of providers (called "nodes" in the project) which represent theoretical expertise, software products (algorithm implementation) and computational resources. To allow interoperability among those nodes and to support the creation of joint research that spans multiple aspects, it becomes fundamental to allow their products and resources to become interoperable and to enable the sharing of knowledge. Such requirements are met with the possibility of deploying reference architectures in a technologically agnostic way and the ability to describe research approaches, algorithms and tools in a flexible and composable fashion.

4 | GENERIC ARCHITECTURE OF THE PROPOSED SCIENCE GATEWAY FRAMEWORK

The high-level architecture of the proposed science gateway framework based on the concept of reference architectures and extended with embedded e-learning support is illustrated in Figure 1. The proposed framework is generic, and its components can be implemented in various ways and using various technologies. In this section, the generic architecture and the roles of its components are explained in a technology agnostic way.

The proposed science gateway framework incorporates two conceptually different major components: the e-Science Center and various reference architectures. The e-Science Center is a centrally deployed and maintained component that provides user management services, e-learning support, and the capability to store, search, compose and launch reference architectures. Reference architectures, on the other hand, are dynamically created and managed infrastructures that are launched and destroyed on-demand.

The e-Science Center GUI is the primary entry point for those users who wish to publish or compose new reference architectures or who wish to launch an already existing reference architecture from the repository. Such users are typically technology experts, application developers or system administrators with significant technical expertise. However, as launching a reference architecture is practically a "1-click" process, end user scientists can also use this interface to launch their own reference architectures (for their individual use or for their community), or with some training, even compose new reference architectures from the existing building blocks.

Authentication, authorization, generic user management and security are handled by the User Management module.

Reference architectures are stored in the reference architecture repository. The repository contains the deployment descriptor that is required to launch the reference architecture, and also a rich set of metadata that enables the various users to find the desired reference architectures and to provide sufficient information for the composition of new ones

The reference architecture launcher (RAL) is capable of taking a deployment descriptor from the reference architecture repository and instructing the orchestrator to set up the desired infrastructure on the targeted cloud resources. The launcher should also be capable of destroying the reference architecture on-demand. An important requirement toward this component is its cloud agnostic nature, with which the launcher can support a wide variety of resource providers, avoiding vendor lock-in and allowing the research community to utilize a wide variety of public and private resources.

The reference architecture composer is an intelligent component that guides the user when creating a new reference architecture from pre-published building blocks (e.g., by providing semantic support and matching various components together), and automatically composing the new deployment descriptor by reusing and extending the descriptors of its building blocks.

The final component of the e-Science Center is the knowledge repository exchange and learning module (KREL) that provides structured and flexible e-learning support for all user profiles. KREL is a generic concept for the creation and exchange of structured learning material related to a particular discipline or an area and the technological solutions that address it. The KREL philosophy is that there are many
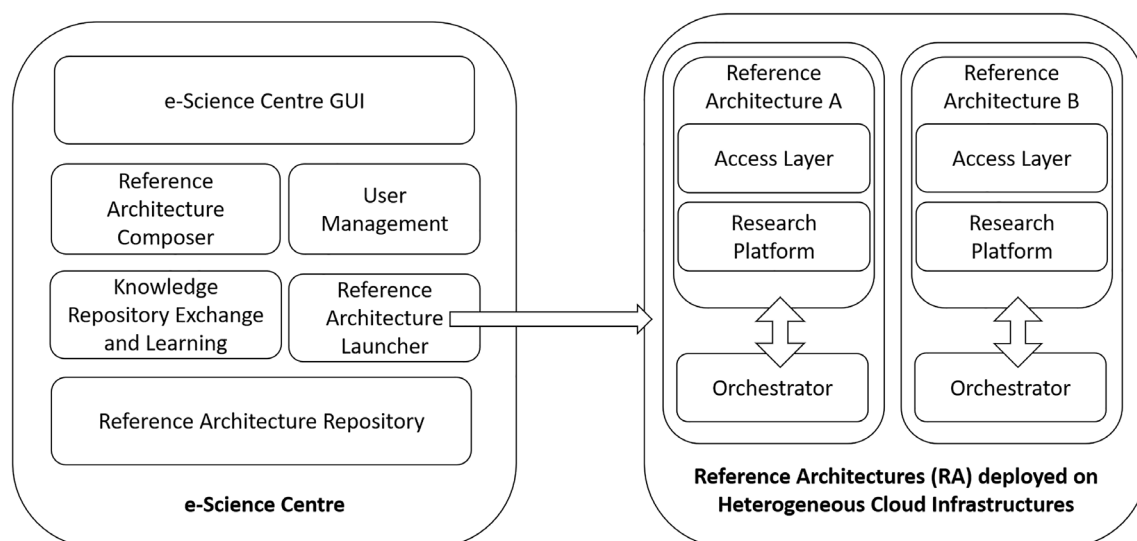
FIGURE 1 High-level architecture of the proposed science gateway framework based on reference architectures and with embedded e-learning support
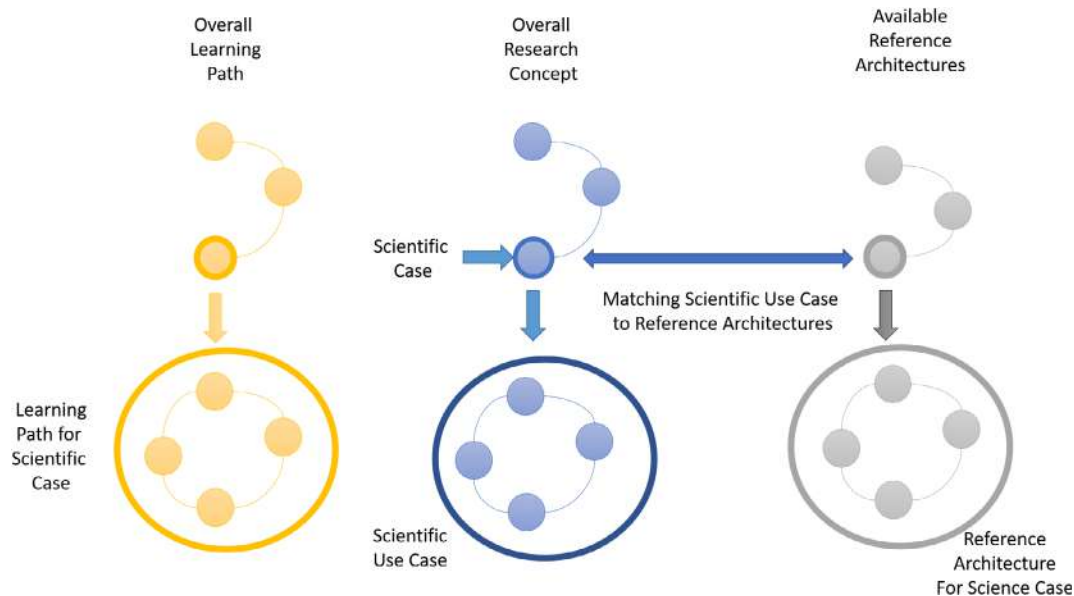
FIGURE 2 Conceptual aspects described in the KREL system

different perspectives (learning paths) that a learner can explore to study a topic and these learning paths may cater for users with different learning profiles. The KREL offers an abstract and structured view which breaks the concepts of a research effort into three conceptual facets in two levels, as illustrated in Figure 2. This structured overview of various facets related to a research effort can be categorized as "What is it", "How is it done", and "How to learn it". The top level consists of the description of the overall research concept, which offers a comprehensive view of the several research activities and how they relate with each other. The overall learning path offers a view on how to understand the topics of the overall research concept, and the available reference architectures detail the available tools and services in support of the research activities. Such an approach is also helpful in finding the appropriate reference architecture for the scientific cases. Each of these three facets is then mirrored for each of the science cases (in the lower half of Figure 2), resulting in learning paths, scientific use cases and reference architectures for each of the research activities. Each of these concepts can be expressed as interconnected recursive graphs.

The second major group of components of the generic science gateway framework are a number of reference architectures launched by the e-Science Center. Such reference architectures can be short-lived solutions that may be destroyed after the execution of a set of jobs or a series of experiments but can also be long-running services accessing persistent storage resources and serving multiple users. Each reference architecture can be regarded as an independent science gateway that serves one or more users and that can be instantiated and destroyed on-demand. Reference architectures can have their own graphical user interfaces (or access layer) and can incorporate one or many analytical tools and their associated data sources (consisting of the research platform).

Reference architectures, together with their associated orchestrators, are launched by the reference architecture Launcher component of the e-Science Center. The RAL is responsible for creating a new instance of the orchestrator on the targeted cloud resource and passing on the deployment descriptor of the reference architecture to this orchestrator. On the other hand, the orchestrator is responsible for deploying the components of the reference architecture, configuring them based on the received description, and managing their run-time behavior (e.g., scaling the resources below the reference architecture based on workload or specified deadlines, or dynamically setting security policies, if required). There are several cloud orchestrators available that can be potential candidates for the orchestrator role in the proposed architecture. When selecting this component, an important requirement could be its cloud-agnostic nature, potential multicloud or even edge/fog computing support, and advanced run-time management capabilities, such as user-defined dynamic autoscaling

Two major distinctive usage scenarios of the launched reference architectures are envisaged. Researchers may launch their own "single-user" reference architectures, which then can be utilized by them, after authentication via the e-Science Center. Alternatively, more complex multiuser reference architectures, with their own user management capabilities, can also be launched and offered for various user communities. The graphical user interface layer (access layer) of the reference architecture can also vary, from simple command-line interfaces to generic access layers such as JupyterHub or Jupyter Notebooks,[47] and sophisticated custom-developed GUIs.

Figure 3 provides an operational view of the described science gateway framework, detailing the various user profiles and their interaction with the system. Although this may be extended in the future, we envisage at least four different user profiles: administrators, domain experts,
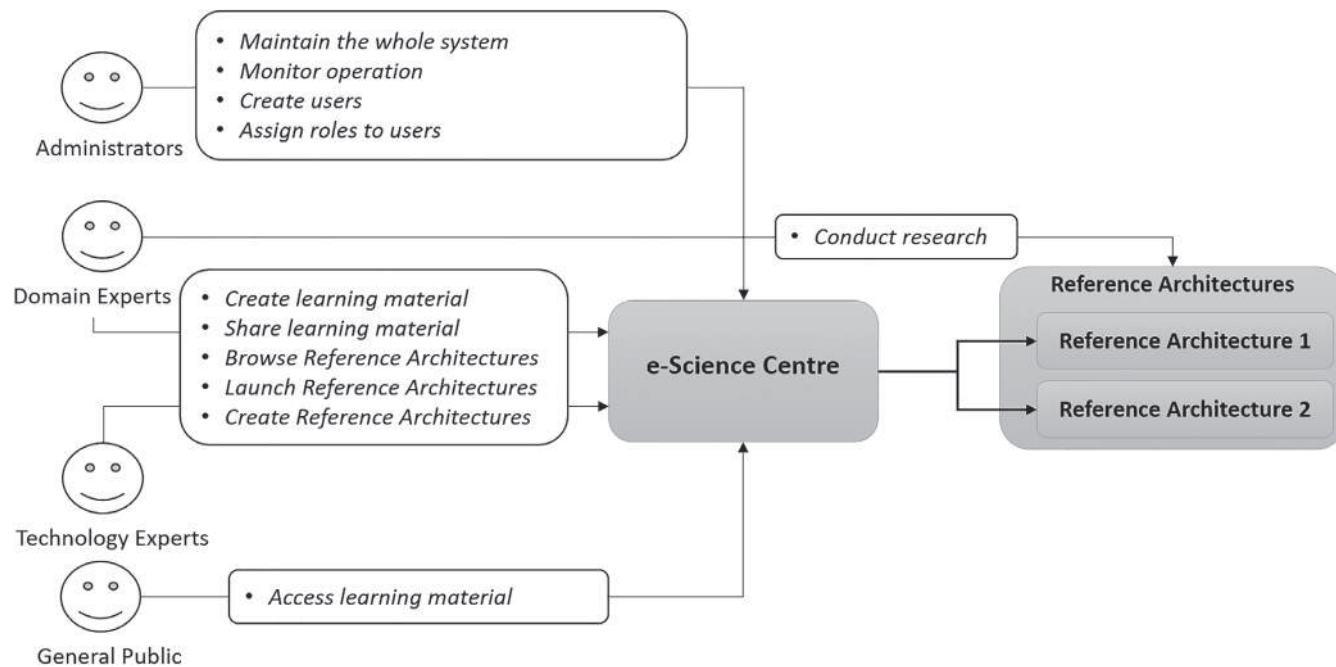
**FIGURE 3** User profiles and main framework components

technology experts, and general public. Administrators are responsible for maintaining the entire system (science gateway framework), monitoring its operation, and managing user creation and roles. Domain experts (scientists) can connect directly with the launched reference architectures to conduct their research. Additionally, they can also create and share learning material related to the science cases, and some of them may also be trained to browse, launch or even create new reference architectures. Technology experts primarily provide technical support for domain experts. They can browse, launch and create reference architectures (especially for larger user communities), and they can also create and share learning material related to the technological concepts. Finally, the general public (or citizen scientists) can also engage with the research by accessing the learning material.

5 | IMPLEMENTATION OF FRAMEWORK COMPONENTS

The implementation of the generic science gateway framework described in Section 4 is currently ongoing in the PITHIA-NRF project. In this section, we present the current state of this implementation which is based on existing components developed in the current and past projects and provides a suitable proof of concept for our approach. When implementing the PITHIA-NRF e-Science Center, the aim is to integrate these various, already existing components, based on widely utilized open-source technologies, and to customize and extend them into the full science gateway framework described in Section 4. Figure 4 illustrates these existing components, and subsequently we describe them in detail, as crucial building blocks of the final solution.

The various components in Figure 4 are color coded, based on their current status. Components in dark gray are fully implemented, operational and are part of the demonstration scenarios presented in Section 6. Components in light gray are implemented and operational but are not in the context of this science gateway framework. These components require customization and integration into the final solution. Finally, components in white do not currently exist and require full implementation. We also note that each box in Figure 4, naming a concrete technology, corresponds to a generic building block of the framework, as indicated in Figure 1.

A key component of the implementation is the MiCADO multicloud orchestrator[11] that is responsible for deploying the reference architectures and managing their lifecycle, based on user-defined policies. Each of these reference architectures has its own MiCADO orchestrator, launched by the MiCADO Launcher. As user management and reference architecture repository components, we are customizing the EMGUM (emGORA user management) and EMGREPO (emGORA repository of executable artifacts) components of the CloudiFacturing Platform, implemented within the CloudiFacturing project.[48] SMARTEST,[49,50] a knowledge repository that assists and facilitates learning by representing knowledge and learning activities as graphs, is utilized as KREL. The reference architecture composer and the e-Science Center GUI are currently nonexisting components and will be fully developed in PITHIA-NRF. However, similar concepts, the digital marketplace of CloudiFacturing and the reference architecture
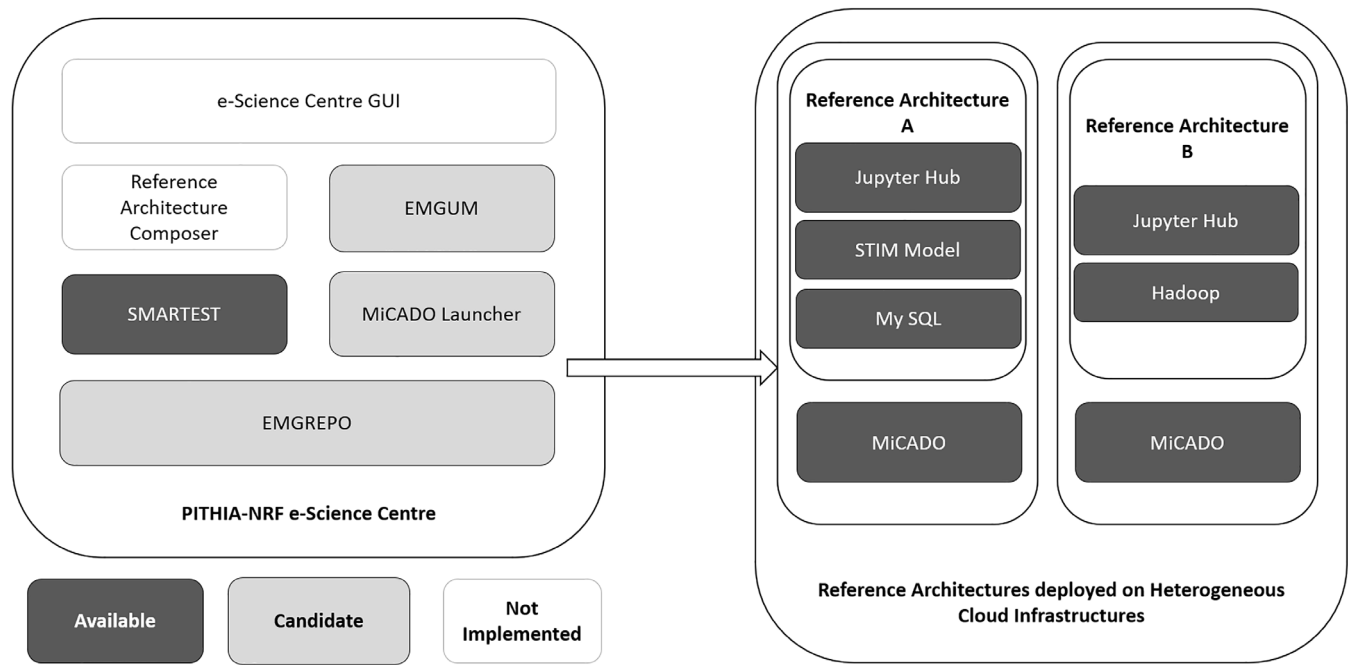
FIGURE 4 Implementation of the generic science gateway framework

composition solutions developed in the DIGITbrain project,[51] will be reused. The two concrete reference architectures shown in Figure 4 will be described in Section 6.

5.1 | MiCADO multicloud orchestrator

MiCADO[11] is an application-level cloud agnostic orchestration and auto-scaling framework. A number of cloud service providers and middleware are supported by MiCADO, including both commercial clouds such as Amazon AWS, Microsoft Azure, Google Cloud Platform, or Oracle Cloud Infrastructure, as well as private cloud systems based on OpenStack or OpenNebula. MiCADO is fully open-source and implements a microservices architecture in a modular way. The modular design[52] supports varied implementations where any of the components can easily be replaced with a different realization of the same functionality.

The cloud agnostic design of MiCADO is based on two major principles. First is the need for a generic orchestration framework providing support for launching and managing applications in various clouds. Therefore, the framework is tied to no specific cloud service provider and supports a mix of public, private and community clouds. It also provides flexibility at the application level, regardless of the underlying cloud. This includes automated deployment and optimized run-time orchestration with features such as automated scaling and enhanced security. Second, a single generic interface to this framework is required. The interface acts as an abstraction layer over the various underlying components of the framework and describes the application, its cloud resources and any policies which govern performance, cost, security or other nonfunctional application requirements.

The high-level architecture of MiCADO is presented in Figure 5. The input to MiCADO is a TOSCA-based application description template[53] (ADT) defining the application topology (containers, virtual machines, and their interconnection) and the various policies (e.g., scaling and security policies) that govern the full lifecycle of the application. MiCADO consists of two main logical components: Master node and worker node(s). The submitter component on the MiCADO Master receives and interprets the ADT as input. Based on this input, the cloud orchestrator creates the necessary virtual machines in the cloud as MiCADO worker nodes, and the container orchestrator deploys the application's microservices in Docker containers on these nodes. After deployment, the MiCADO monitoring system monitors the execution of the application and the policy keeper performs scaling decisions based on the monitoring data and the user-defined scaling policies. Optimizer is a background microservice performing long-running calculations on-demand for finding the optimal setup of both cloud resources and container infrastructures.

Currently there are various implementations of MiCADO based on its modular architecture, which enables changing and replacing its components with different tools and services. As Cloud orchestrator, the latest implementation of MiCADO can utilize either Occopus[10] or Terraform.[9] These both are capable of launching virtual machines on various private or public cloud infrastructures. However, as the clouds supported by these
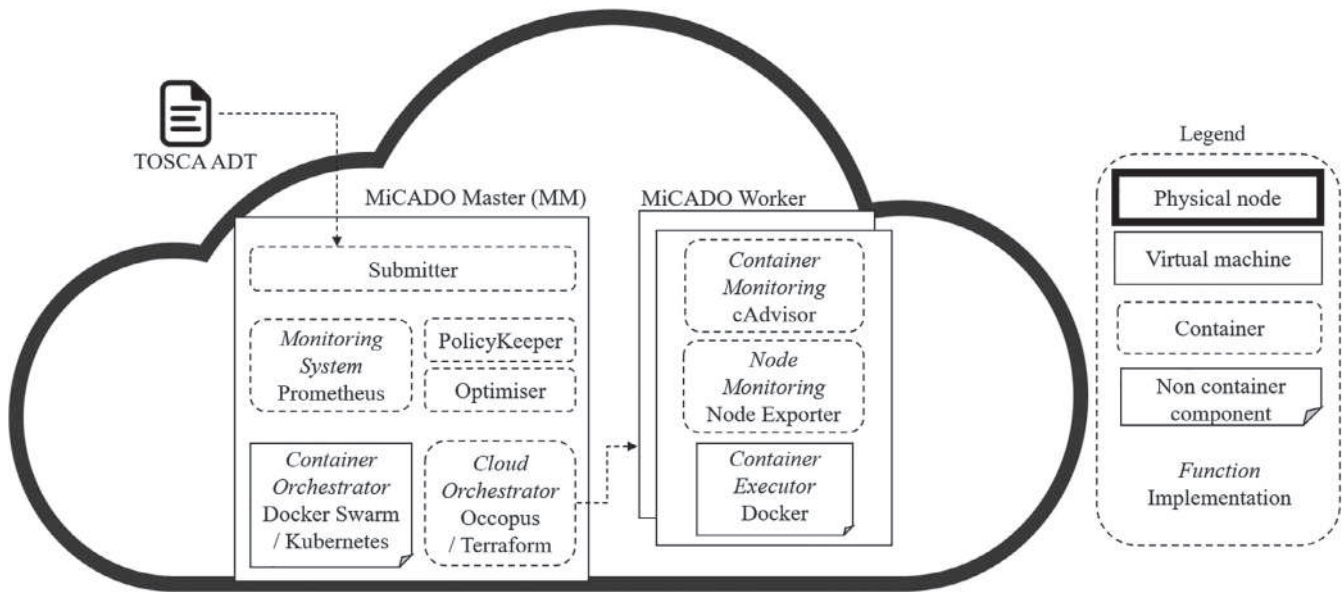
FIGURE 5 High-level architecture of MiCADO

two orchestrators differ, MiCADO can support a wider variety of targeted resources. As container orchestrator, MiCADO uses Kubernetes,[6] which is probably the most popular open-source container orchestrator at the moment. The monitoring component is based on Prometheus,[54] a lightweight, low resource consuming, but powerful monitoring tool. The submitter, policy keeper[55] and optimizer components were custom implemented for MiCADO.

In the proposed science gateway framework, MiCADO master nodes are started on—demand by the MiCADO launcher component. Based on the ADT received from the launcher, MiCADO master deploys the specified reference architecture on the targeted cloud resources. Once deployed, MiCADO starts monitoring the deployed microservices and scales those up or down, based on the specified policies. At the end of the application microservice's lifecycle, MiCADO is also responsible for stopping it and deallocating resources.
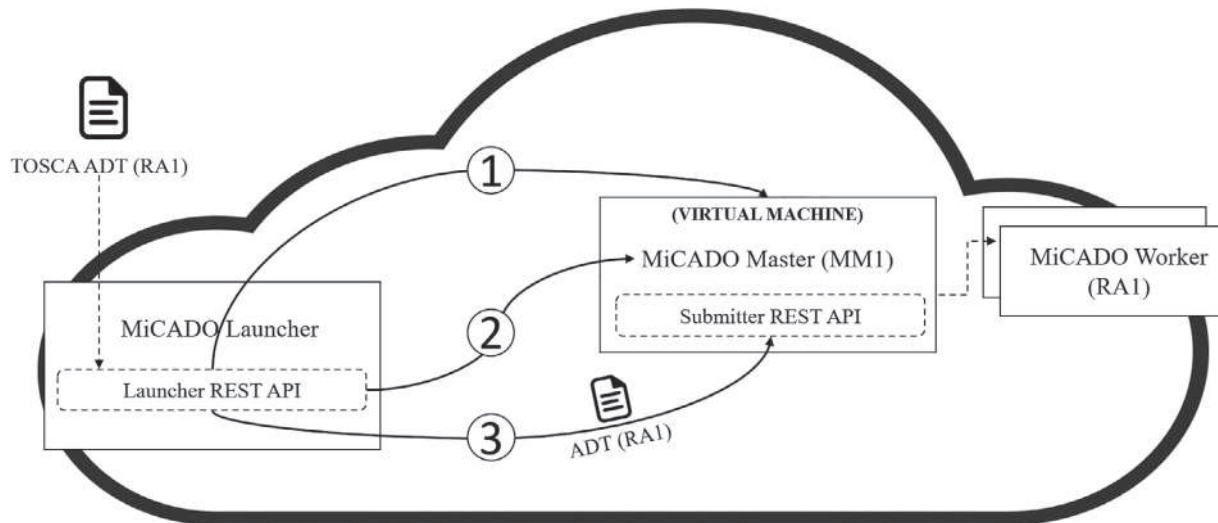
5.2 | MiCADO launcher

The MiCADO launcher component is responsible for deploying individual instances of the MiCADO cloud orchestrator, each of which will be itself responsible for the deployment and management of a single reference architecture. To this end, the MiCADO launcher provisions an adequate virtual machine on the target cloud, installs the MiCADO orchestrator on that virtual machine, and then submits the ADT for a given reference architecture to the ready-to-use instance of MiCADO.

The launcher features a RESTful API which can be used to create, query, update, and delete MiCADO orchestrators and their respective reference architectures. It is necessary to provide the launcher with details of the desired cloud infrastructure for hosting new MiCADO orchestrators and, at deployment time, submit the ADT which describes the reference architecture to be deployed.

Figure 6 gives a high-level overview of the basic functionalities of the MiCADO Launcher. Once deployed behind a secure domain and configured with the desired specification and credentials for hosting MiCADO orchestrators, the launcher is ready. Then, a POST request—with the ADT for a desired reference architecture in the payload—can be sent to the REST API of the launcher. The launcher will start a new thread for the deployment of that reference architecture, carried out in the following steps and also depicted in Figure 6:

1. Provision and configure a new virtual machine from the cloud service provider.
2. Run an Ansible playbook against the new virtual machine to install and configure MiCADO.
3. Submit the ADT of the provided reference architecture to the MiCADO submitter REST API.

On submission of the ADT to the submitter REST API, the MiCADO master node takes over. MiCADO worker nodes are deployed, container microservices are scheduled across worker nodes, and any policies are enforced across the lifetime of the application. The MiCADO launcher can be then used to query the status of the MiCADO master and its application, or, when appropriate, destroy the MiCADO master and its

FIGURE 6 High-level functionality of MiCADO launcher

reference architecture, deallocating any cloud resources that were created for that deployment. The MiCADO launcher can handle the deployment and management of multiple MiCADO/reference architecture pairings at the same time, treating each one as a unique resource in the RESTful API.

The MiCADO launcher has already been implemented within the DIGITbrain project and is part of its current testbed infrastructure. This implementation will be reused and further customized within the PITHIA-NRF e-Science Center.

5.3 | SMARTEST knowledge repository as KREL

The KREL component of the PITHIA-NRF e-Science Center is based on SMARTEST,[48] a web application hosting a knowledge and learning repository which is based on graph-based representations of conceptual models and learning paths. The theoretical foundations of SMARTEST can be found in two streams of pedagogical research which link together the structure of the concepts with the learning approaches. The first is Novak's theory of "concept maps", where the author sought to follow and understand changes in children's knowledge of science.[56,57] The second stream concerns learning paths[58] that guide users through the learning content. SMARTEST enables the creation of structured models and learning maps for knowledge sharing. For content creators, the platform offers a simple and intuitive editor to create models and learning paths, and a user-friendly learning view which allows one to check the learning process and to communicate directly with content creators to ask questions. SMARTEST allows describing a topic with different kinds of graphs to cover different views: learning paths, conceptual structures, and ontologies. SMARTEST describes the links among concepts as edges between nodes, similarly to data graphs. This allows users to learn the disciplines deeper and enable them to grasp learnt concepts and problems in a context.[59]

SMARTEST was originally developed as an e-learning tool in an academic context as the continuation of the EnAbled project.[60] Its potential to be employed as a more generic knowledge creation and content tool beyond the boundaries of a teacher-student scenario has led to restructuring the code to support research environments whereby the roles of lecturers and students are extended to more generic content creators and content consumers. SMARTEST is based on a web-based graphical user interface connected to an API offering an abstraction layer for such functionalities as authentication and authorization, graph manipulation and storage (nodes and edges) representation, and communication support for each of the nodes. SMARTEST can be further extended to import existing external concept maps and data models (e.g., Neo4j graph database[61]). To allow SMARTEST to use ontologies to describe the structure of research activities, ontologies can be imported from dedicated systems such as Protégé[62] in OWL format.[63]

5.4 | EMGUM user management system

EMGUM,[64] developed within the scope of the CloudiFacturing project,[48] is a generic component that is responsible for authentication, authorization and security policy management of complex frameworks consisting of multiple distributed components, such as the generic science gateway framework. EMGUM handles access control policies, and issues access tokens to the distributed components of the framework. More specifically, it

provides the following key functionalities: (1) a single point for end-user management by facilitating the central storage and management of users, credentials, roles and organizations, (2) a centralized authentication and access control mechanism that enables single sign-on and token-based authentication to a platform, (3) a platform-level authentication mechanism to facilitate secure and authenticated inter-component interactions, and (4) a centralized authorization mechanism for the facilitation of individual components to define and manage authorization policies and decisions.

The EMGUM functionalities are provided through the OpenID connect (OIDC) standard[65] protocol, and its implementation is based on a popular open-source identity and access management solution called Keycloak.[66] EMGUM uses Keycloak as the OIDC server that centrally stores and manages users, credentials, roles, and organizations. Keycloak is responsible for fulfilling the authentication and authorization requirements. In addition, EMGUM also provides an API server that facilitates the communication between Keycloak and the rest of the framework.

Figure 7 illustrates the architectural view of EMGUM in relation to the different components of the proposed science gateway framework. It should be noted that this is not the execution flow of the different events to be processed but rather a pictorial representation of the relationship between the different components illustrating how the overall system works. All users and all components (also referred to as the OIDC clients) of the framework register themselves with EMGUM. Once registered, EMGUM is responsible for handling the related authentication and authorization decisions through a Keycloak specific client adapter configured at the application (hosting) server (the entity responsible for the execution of a particular framework component) that directly interacts with Keycloak.

5.5 | EMGREPO: Repository of executable artifacts

The generic science gateway framework presented in Figure 1 requires a repository where the reference architectures can be stored and searched for. Using MiCADO as its orchestrator implies that the reference architectures will be stored as application description templates (ADT). Such ADTs can be directly fed to MiCADO as input which then acts on the ADT to deploy and manage the reference architecture at runtime.

For a service to be deployed as part of a reference architecture, the service should meet two independent requirements. First, the service should be containerized in an OCI-compliant image format (e.g., a Docker image) and pushed to a remote container repository (e.g., the public DockerHub, or a self-hosted container registry). Second, the service should be described following the TOSCA format that MiCADO follows for its ADTs. This description contains the desired configuration of the container and can be authored manually, or automatically translated from a Docker-compose template or Kubernetes manifest using tools developed as part of the MiCADO project.[67]

A repository to store executable artifacts, called emGORA repository of executable artifacts (EMGREPO), has already been developed by the authors within the CloudiFacturing project.[48] EMGREPO can store executable artifacts of various execution engines, one of those being MiCADO.
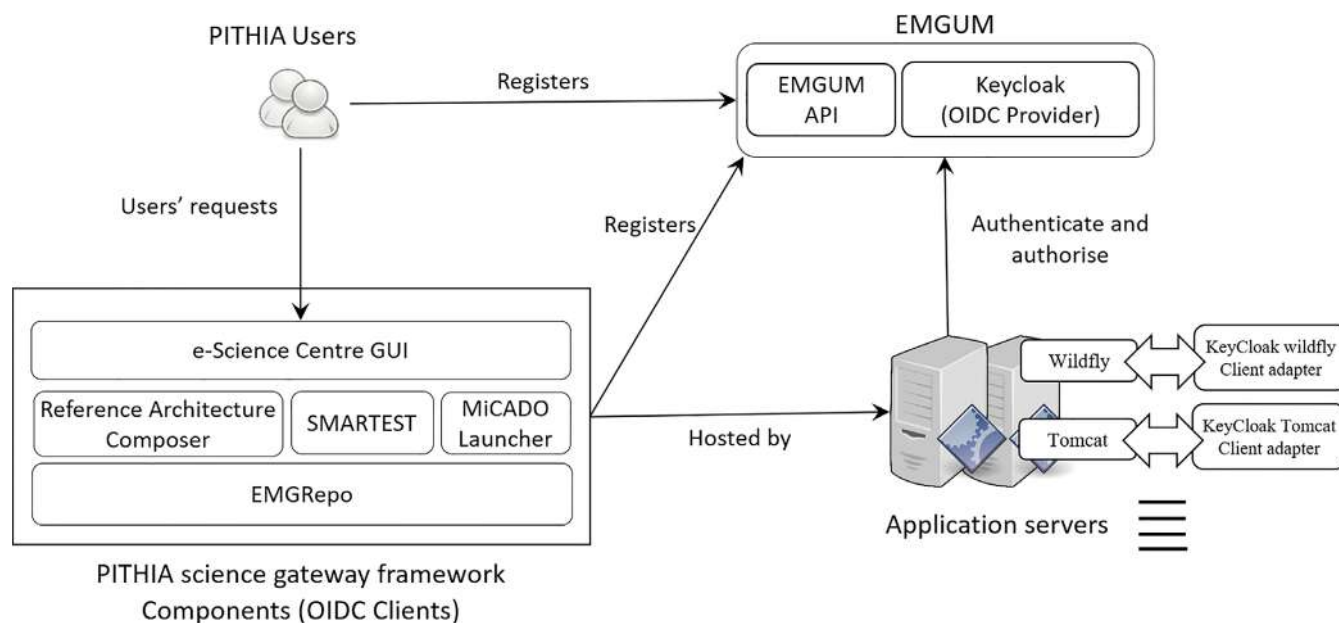
FIGURE 7 Architectural view of overall system architecture in the context of EMGUM

Therefore, EMGREPO already supports storing and searching for MiCADO ADTs, providing a ready to use solution as reference architecture repository within the proposed framework.

EMGREPO is developed under the open-source version of the nexus repository framework[68] and stores artifacts (e.g., binary executable, workflow definition XML, YAML configuration file, simple URL reference link, bash script) along with metadata in a defined directory structure. The metadata describes the artifacts and includes information such as versioning, dependencies and the parameters required for running in the specific execution engine. To provide custom metadata management, a new EMGREPO plugin has been developed for nexus. EMGREPO is accessible through a RESTFUL API, providing easy integration with other components. Additionally, it is already integrated with the EMGUM service providing support for authentication and authorization.

When customizing EMGREPO for the proposed science gateway framework, one of the biggest challenges is the definition of the metadata structure that is required to describe reference architectures. While this work is currently ongoing and the final set of metadata will be defined as future work, the required fields should describe the generic behavior/characteristic of the reference architecture (name, description, functionality, creator, version etc.), together with detailed technical description of the composing microservices and their composition (e.g., container configuration, hardware requirements, operating system requirements, and input and output data).

5.6 | Reference architecture composer

The role of the reference architecture composer (RAC) component is to support the creation of reference architectures from smaller building blocks, for example, from the ADTs of individual microservices, or by combining numerous reference architectures into new ones. Although such a component is not yet available, a similar development task is currently ongoing in the DIGITbrain project.[51] The aim is to automatically construct ADTs from the descriptions of already published components. For example, individual microservices representing user interfaces, data analysis solutions, scientific models, or visualization tools can be published in the repository, and the RAC can be used to combine these into new reference architectures, automatically generating the required ADT. In early versions, the composition will be solely based on the understanding of the human actor, utilizing the rich set of metadata published in the repository. However, an automated semantic matchmaking is also envisaged in later editions.

The RAC will be particularly useful for improving the re-use of reference architectures. Reference architectures are initially composed in a vendor-free and cloud agnostic way. The choice of a specific cloud service provider or cloud middleware can be made later, at the time of deployment, with the RAC adding the required cloud infrastructure specifications and parameters to an ADT that originally contained only abstract descriptions of the required compute, storage or network resources.

6 | PROOF OF CONCEPTS AND RESULTS

This section presents proof of concept implementations that demonstrate the previously described principles and the way the framework is intended to work. The implemented prototypes demonstrate a subset of the proposed functionalities, the ones shown in dark gray in Figure 4. The major objective was to illustrate the deployment and the run-time management of reference architectures, and to show how embedded ontology-based e-learning support can be provided.

The two presented case studies demonstrate different aspects and capabilities of the framework. The first case-study demonstrates how a reference architecture, composed of JupyterHub[47] and Apache Hadoop[69] can be deployed and auto-scaled at run-time by MiCADO. The second case study deploys a model, together with JupyterHub and a MySQL database, that is utilized and developed by the PITHIA-NRF community, and also demonstrates how embedded e-learning support for the same scenario can be provided with SMARTEST.

As described in Section 5.5, before composing these components into reference architectures for the two above case studies, two prerequisites first had to be met. First, containerized versions of JupyterHub, Apache Hadoop and MySQL were found in the public DockerHub, and deemed appropriate for the use cases. Additionally, the targeted PITHIA application (model) has also been containerized and published in DockerHub. Next, a description of the configuration of each of these containers was written using the TOSCA format of the MiCADO ADT. With these steps completed, the components could then be used to build the respective reference architectures described below.

6.1 | Scalable JupyterHub deployment with Apache Hadoop

JupyterHub and Apache Hadoop are popular solutions, widely utilized by various user communities in big data analytics. As both tools are potential candidates for several PITHIA-NRF user scenarios, our first reference architecture deploys these two together in a scalable way. JupyterHub serves

groups of users accessing computational resources and applying flexible configurations and policies. It offers users their own workspaces on shared resources via Jupyter Notebook Servers and accelerates application development, information sharing and usability of data analytics frameworks, such as Apache Hadoop. On the other hand, Apache Hadoop helps users with distributed processing of large datasets across clusters of computers, using the MapReduce[70] programming model. Applying the two environments together, a user can work with Hadoop distributed file system (HDFS) and process data by submitting Hadoop jobs, through their own workspaces provided by JupyterHub.

The deployment of both JupyterHub and Apache Hadoop is complex and time-consuming. Additionally, workloads may change dynamically based on the number of users and applications, requiring the automated scaling of computational resources. The presented solution addresses both issues. A reference architecture, incorporating both components, is deployed using a single deployment descriptor, in the form of a MiCADO ADT. Such automated deployment supports easy portability between various cloud infrastructures as moving between clouds requires only minor modification of the ADT.[52] Additionally, a scaling solution for JupyterHub was developed that scales Jupyter Notebook servers at both virtual machine and container levels.

A high-level architecture of the implemented solution is shown in Figure 8. A MiCADO ADT has been prepared that describes the virtual machines and containers running JupyterHub and Hadoop. After submitting the ADT, the MiCADO master creates the virtual machines in the targeted cloud (in this case Amazon EC2) and deploys both components in Docker containers, encapsulated into Kubernetes pods. Following deployment, users can log in to JupyterHub, start their own workspace, and work with Apache Hadoop.

Scalability of Jupyter Notebook servers is also managed by the implemented solution. A scaling policy, described in the ADT, for Jupyter Notebooks was developed, where MiCADO works alongside JupyterHub to scale Notebook servers at both virtual machine and container levels. JupyterHub comes with various spawners that manage the creation of Jupyter Notebook servers on the targeted computing resources. To support automated scalability, we used the KubeSpawner of JupyterHub that utilizes Kubernetes[8] to deploy and scale Jupyter Notebooks Servers. However, using the KubeSpawner has two notable disadvantages: (1) a Kubernetes cluster needs to be set up prior to the deployment of JupyterHub that may, in some clouds, require significant expertise and effort; and (2) Kubernetes only scales at the level of containers while scaling virtual machines under the Kubernetes cluster requires a different mechanism, that may not be available in the targeted cloud. To overcome these shortcomings, we combined the capabilities of the KubeSpawner with MiCADO, as illustrated in Figure 8. In the implemented solution, the KubeSpawner utilizes the Kubernetes API running inside the MiCADO Master and manages the deployment and scaling of the containers hosting Jupyter Notebook servers. Additionally, the cloud orchestrator and policy keeper components in MiCADO are responsible for the deployment and scaling of virtual machines hosting the Kubernetes cluster.
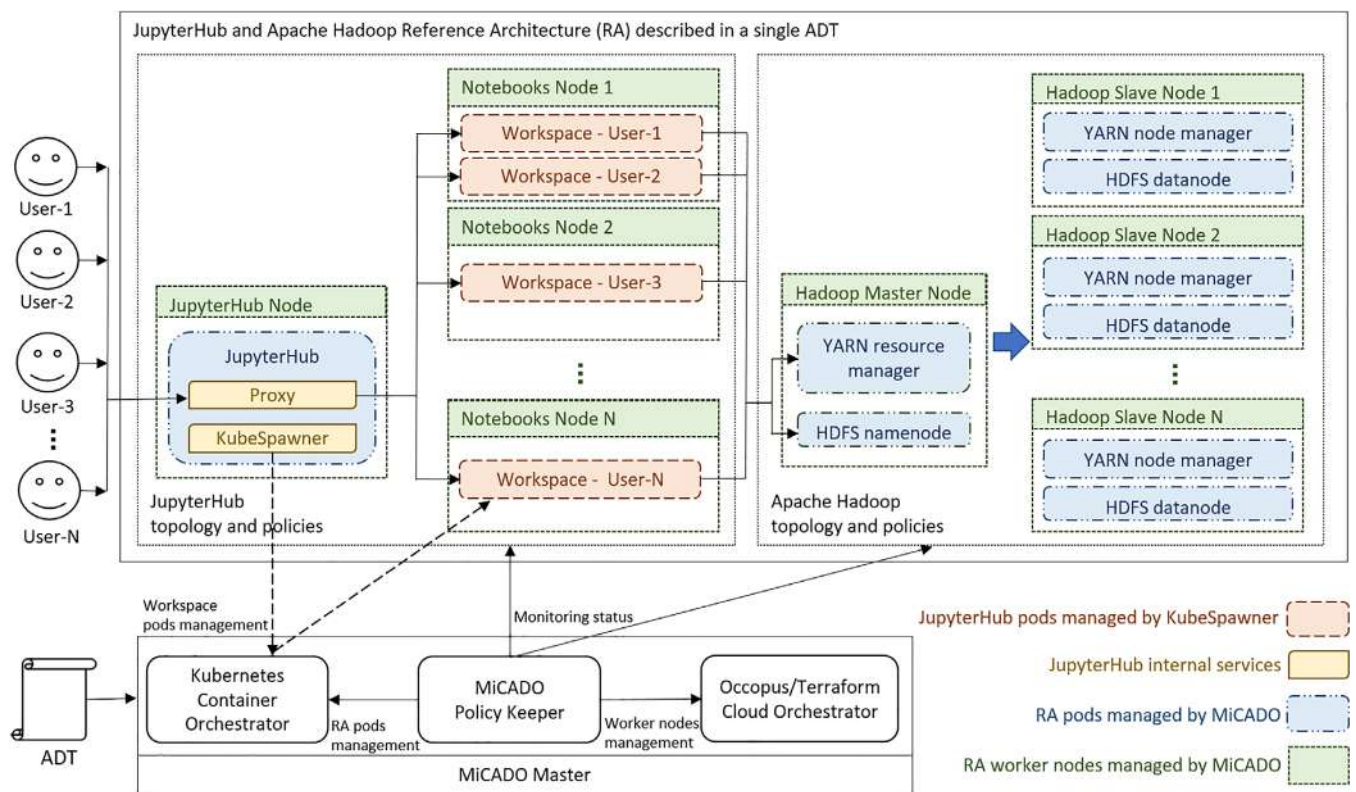

FIGURE 8 Scalable JupyterHub with Apache Hadoop as a reference architecture deployed and managed by MiCADO

A unique feature of MiCADO is that scaling policies are not fixed or preprogrammed[11] but can be created dynamically by application developers. The policy keeper component of MiCADO takes a policy description in the ADT as input for handling the monitoring sources (Prometheus exporters), the monitoring queries (Prometheus expressions), the monitoring alerts (Prometheus alerts) and the scaling rules (decision making in Python). Therefore, custom scaling policies can be easily implemented without any major limitation. In the presented demonstrator, we created a scaling policy for JupyterHub that scales the number of virtual machines hosting Jupyter Notebook servers. The objective was to demonstrate that a meaningful scaling policy can be implemented in MiCADO for this scenario. However, comparing the efficiency of the scaling logic to other alternatives was out of scope of this investigation.

The implemented scaling policy and the related scaling logic scales the number of virtual machines (MiCADO worker nodes) hosting user workspaces (Jupyter Notebook servers) running as Kubernetes Pods. The scaling policy is based on the number of running workspaces per worker node and executed periodically, every 15 s by default. If nodes become overloaded, meaning that the average number of workspaces running on the nodes reaches a threshold, MiCADO issues an overloaded alert and deploys a new worker node. After this, KubeSpawner will start scheduling workspaces to the newly created node to balance out the load. On the contrary, when the average number of workspaces per node falls below a threshold, MiCADO tries to find a node to remove to spare resources.

A primary objective of the policy is to maintain users' workloads noninterrupted when scaling down. Therefore, we check the nodes before removing them, and we only scale down empty nodes. Unfortunately, as KubeSpawner instantiates workspaces on all available JupyterHub nodes by default, we potentially have a race condition: KubeSpawner may try creating a workspace on a node that MiCADO tries removing from the cluster. To eliminate this race condition, we disable scheduling on nodes (using the Kubernetes API) that are selected for potential removal from the cluster (typically the least utilized node at the time of making the scaling down decision), and we only remove nodes that are empty and where scheduling is disabled.

Based on the above, Prometheus may issue any of the following three alerts in every monitoring cycle. *Overloaded alert* means that the average number of running workspaces for nodes with scheduling enabled has increased above a predefined threshold. In this case MiCADO needs to scale up. Therefore, it checks whether there are any nodes where scheduling is disabled. If it finds such nodes, then it makes the most utilized one schedulable (to increase the chance of releasing less utilized nodes) and allows KubeSpawner to deploy new workspaces on it. Otherwise, it deploys a new node and makes it schedulable. *Underloaded alert* means that the average number of running workspaces for nodes with scheduling enabled has fallen below a threshold. In this case, MiCADO counts the number of schedulable nodes. If this number is one, then no action will be taken (one node will always remain schedulable to accept a potential new workload). However, if the number of schedulable nodes is larger than one then MiCADO disables scheduling on the node with the smallest workload, marking it as a candidate for removal. Finally, *unschedulable alert* means that there is at least one node where scheduling is disabled. In this case, MiCADO checks all such nodes, and the empty ones are removed from the cluster. Graphical representation of these algorithms is shown in Figure 9.
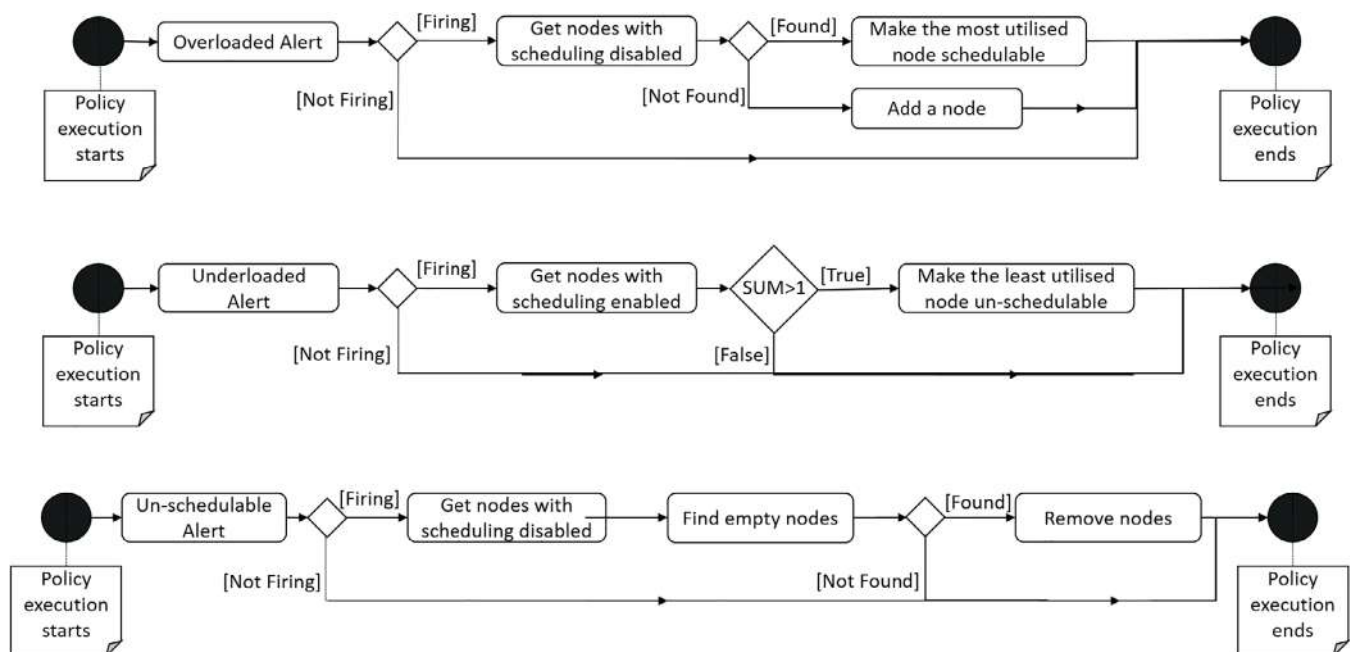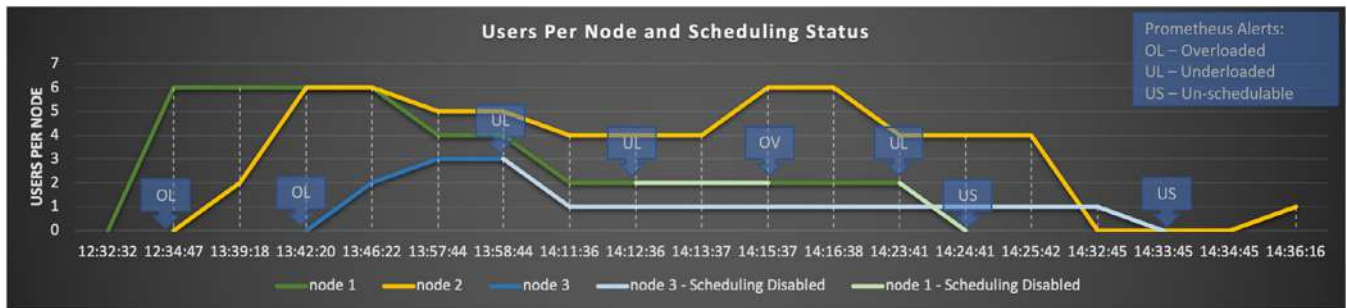

FIGURE 9 Scaling policies implemented in MiCADO

FIGURE 10 Experimental results of the implemented JupyterHub scaling policy

The above-described solution was tested in an experimental scenario to observe the intended scaling behavior, as shown in Figure 10. Jupyter Notebook Servers were started manually from the deployed JupyterHub. Initially, one node hosting notebooks was started and became overloaded at 12:34:47. As a result, a new node was started automatically. Following the continued creation of new notebooks, Prometheus fired another *overloaded alert* at 13:42:20 and MiCADO added another worker node as a result. As load started to decrease, an *underloaded alert* was fired for the first time at 13:58:44. As a result, MiCADO found the least utilized node (*node 3*) and changed its status to scheduling disabled. As users continued to leave the service, at 14:12:36 another *underloaded alert* was fired, and the status of *node 1* was also changed to scheduling disabled. While *nodes 1* and *3* had scheduling disabled status, Prometheus fired *unschedulable alerts* and MiCADO tried to remove those nodes from the cluster. However, as both nodes had running workspaces, no further action was taken, and no new workspaces were scheduled on the un-schedulable nodes. When workload increased again, and a new *overloaded alert* was fired at 14:15:37, MiCADO selected the most utilized node with scheduling disabled (*node 1*) and enabled scheduling on it. Finally, as workload decreased again, a new *underloaded alert* was fired at 14:23:41, and the two nodes (*node 1* and *node 3*) were finally destroyed at 14:24:41 and 14:33:45, as they became empty. We emphasize, that one node, in this case *node 2*, always stays alive, due to the implemented policy.

6.2 | Reference architecture for the deployment of solar wind-driven empirical model for the middle latitude ionospheric storm-time response

The National Observatory of Athens (NOA)[71] has built a solar wind-driven empirical model for the middle latitude ionospheric storm-time response, namely the storm-time ionospheric model (STIM).[72] The model forecasts ionospheric storm effects at middle latitudes triggered by solar wind disturbances. It collects near-real-time datasets describing the solar wind conditions in the Earth's vicinity from a number of sources in variable data formats and temporal resolutions. It homogenizes and resamples data at a standard temporal resolution. Finally, an empirical model analyses temporal variations of the interplanetary magnetic field (IMF) parameters, detects intervals of ionospheric storms, and stores the relevant data in an appropriate relational database schema.

A high-level architecture of the implemented model is shown in Figure 11. As input, the model receives IMF parameters obtained in real-time from the magnetometer (MAG) instruments onboard advanced composition explorer (ACE) spacecraft or deep space climate observatory (DSCOVR) mission, with the purpose to assess the solar wind conditions at L1 Lagrangian point. The model runs as a crontab task every 5 min and monitors a number of online resources providing ionospheric data in various temporal windows (5 min, 2 h, 6 h, 1 day, 3 days, 7 days) and at various temporal resolutions (1 m,1 h).

Data input is provided as both JSON and/or custom ASCII formatted information. The model parses both input types and serializes records in a common format (Python Pydantic serializing library). Data is then transformed into database ready records (Python SQLAlchemy ORM model), further cleaned, and stored into temporary MySQL database tables. Finally, aggregate queries are used to calculate 1 h resolution values and store/update the appropriate database tables. Metadata information regarding the sources is also stored.

When updated input information is available, a stored procedure (ionosphere storm procedure) is triggered by the model. The procedure applies empirical thresholding rules on the input data to issue an alert and forecast ionospheric storms which are then stored in the database. When certain criteria are met, critical values are initialized and further updated on each run until the incident is considered closed. Finally, using the stored results and further external ionospheric data (i.e., 30-day running median estimates of key ionospheric parameters that are representative of normal ionospheric variation), the forecasted local ionospheric storm-time variation for the next hours is calculated. The calculations are based on a set of empirical expressions. Their implementation is driven by the latitude of the observational point and its local time at the time of the alert. More precisely, the model distinguishes two latitudinal zones (one for latitudes less than 45 degrees and one for latitudes greater than 45 degrees) and
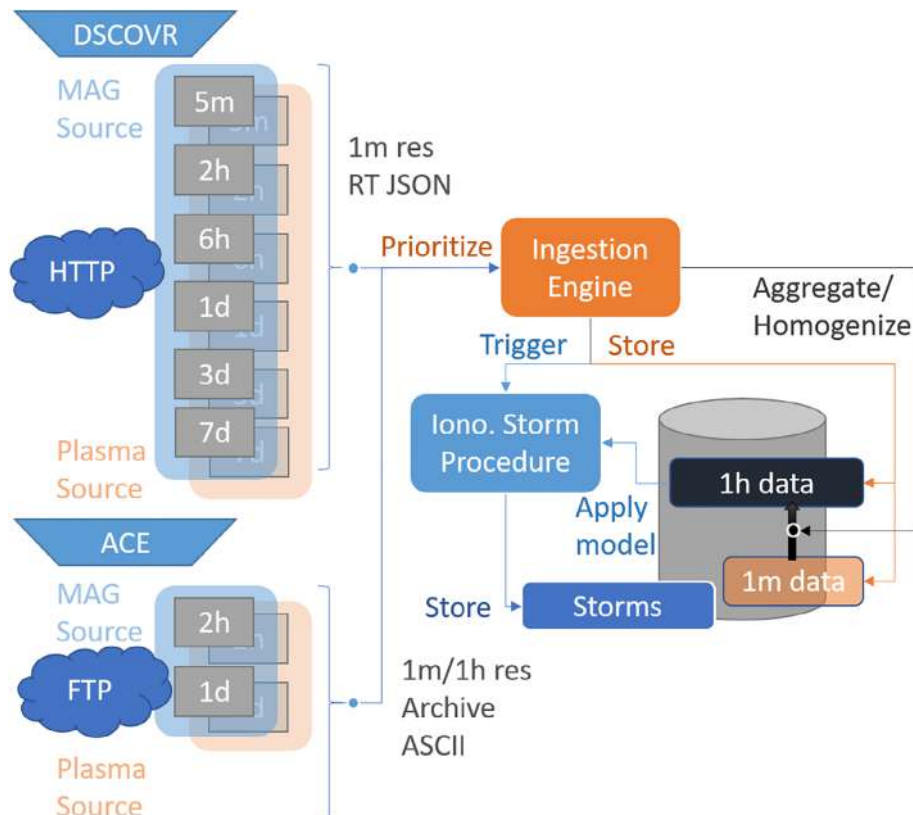
FIGURE 11 High-level architecture of the STIM model

four local time sectors (morning, prenoon, afternoon, evening) to identify the proper one among the total of eight empirical expressions anticipated. Ionospheric storm-related results are also stored in the database.

The current version of the STIM model is deployed in the NOA local computing environment. As such, the access is restricted to local users, and the deployment is complex and time consuming. Additionally, extending/replacing the current command line interface with a more convenient solution, for example, facilitating access from Jupyter Notebook[73] or a custom user interface, requires further work and significant expertise. In order to overcome these limitations, a reference architecture has been developed, incorporating the STIM model, a MySQL Server and JupyterHub.[74] Based on the concept presented in Section 4, such reference architecture can be selected from the reference architecture repository and deployed automatically on a wide range of cloud infrastructures, significantly reducing complexity and supporting easy portability and instantiation.

The reference architecture is illustrated in Figure 12. The STIM model contains the model codebase alongside all the required configurations. The MySQL server hosts a database with all model-related data. JupyterHub is the programming interface that provides access to both the STIM model and the database. Each of these components is deployed in Docker containers encapsulated into Kubernetes pods. The entire reference architecture is described in an ADT and deployed by MiCADO. MySQL data is stored in MiCADO workers using persistent volumes, while JupyterHub data is in an external network file system (NFS).

After deployment, the user can log in to JupyterHub and start a Notebook server. Jupyter console can be used to SSH to the STIM model, check its status, or execute it on-demand. Jupyter Notebooks support querying the MySQL database, creating graphs and analyzing the results.

For testing purposes, the STIM reference architecture was deployed by MiCADO. Figure 13 illustrates the graphical outputs of a sample query executed on-demand from Jupyter Notebook. The aforementioned query requests datasets describing the Solar Wind conditions for a given date/time interval (i.e., 2019–06-14 T00:00:00 to 2019–06-15 T00:00:00), and returns the total magnitude and the x, y, and z components of the interplanetary magnetic field (IMF) in a multiline plot. The above query also identifies IMF disturbances detected by the model over time to be related with ionospheric storms, and in such case, a semitransparent vertical zone depicts the disturbed interval.

To provide embedded e-learning support in the form of learning paths, the science case and its reference architecture are also described in the SMARTEST knowledge repository, as illustrated in Figure 14. The top graph connects the science case (STIM investigation) with learning
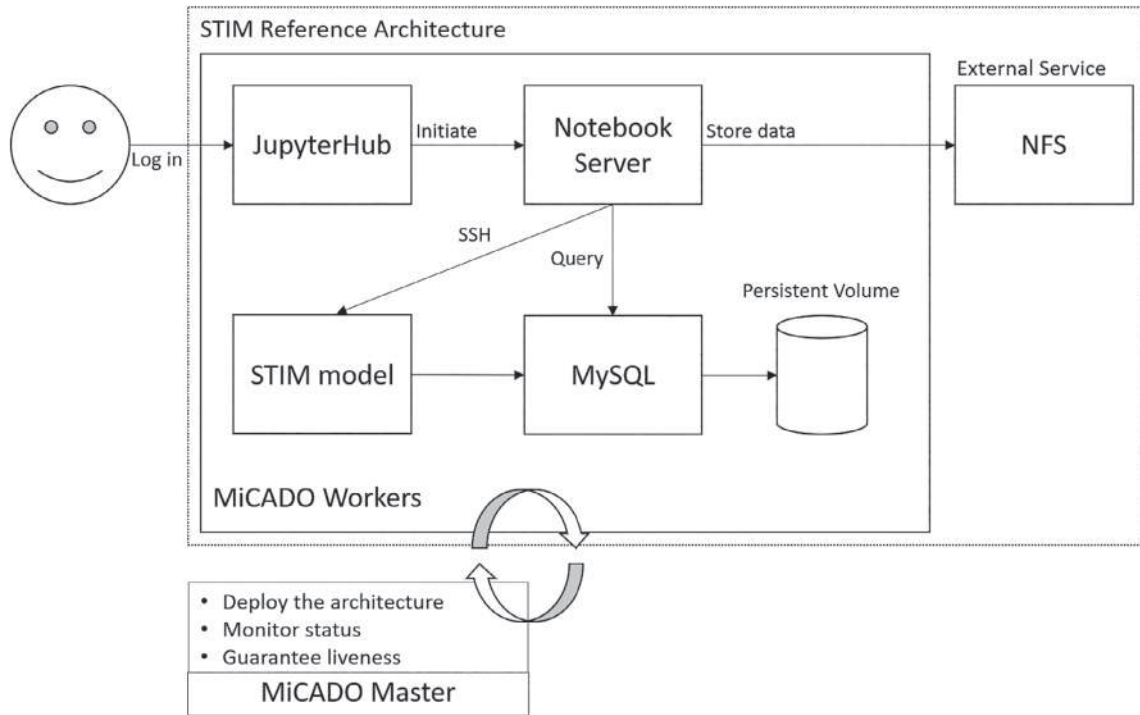
FIGURE 12 Deployment of the STIM reference architecture
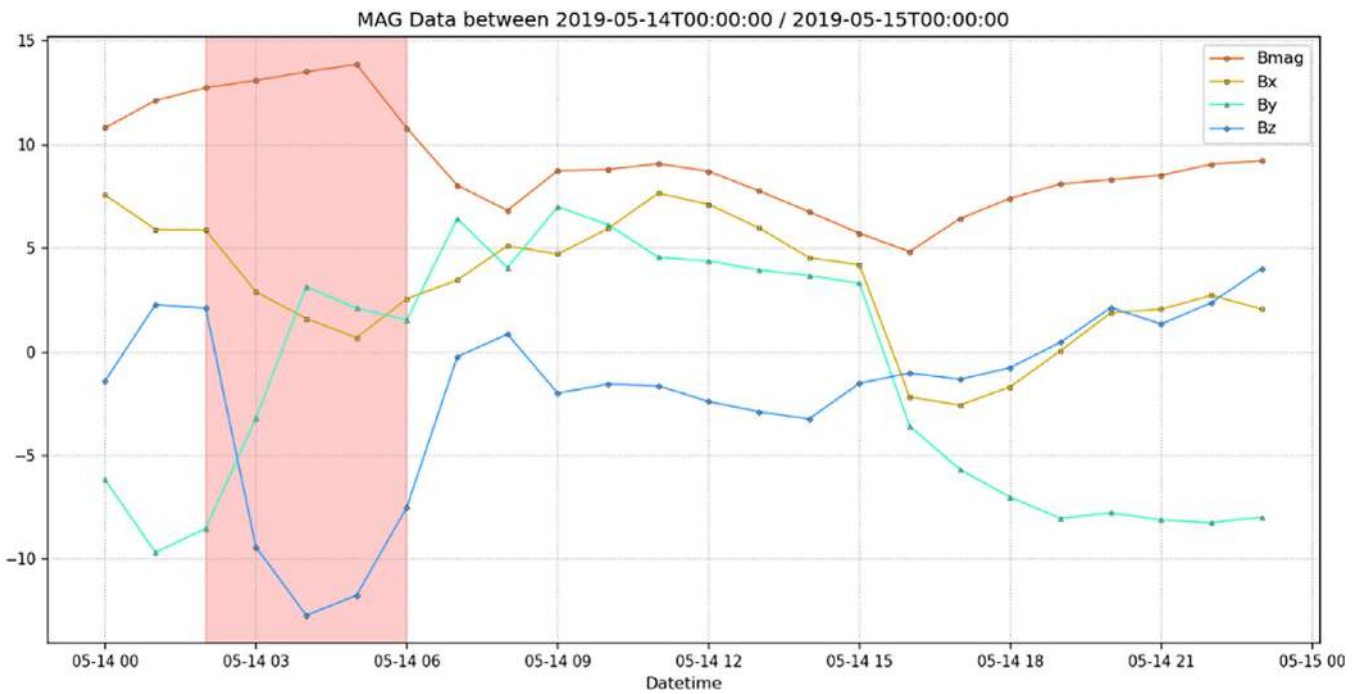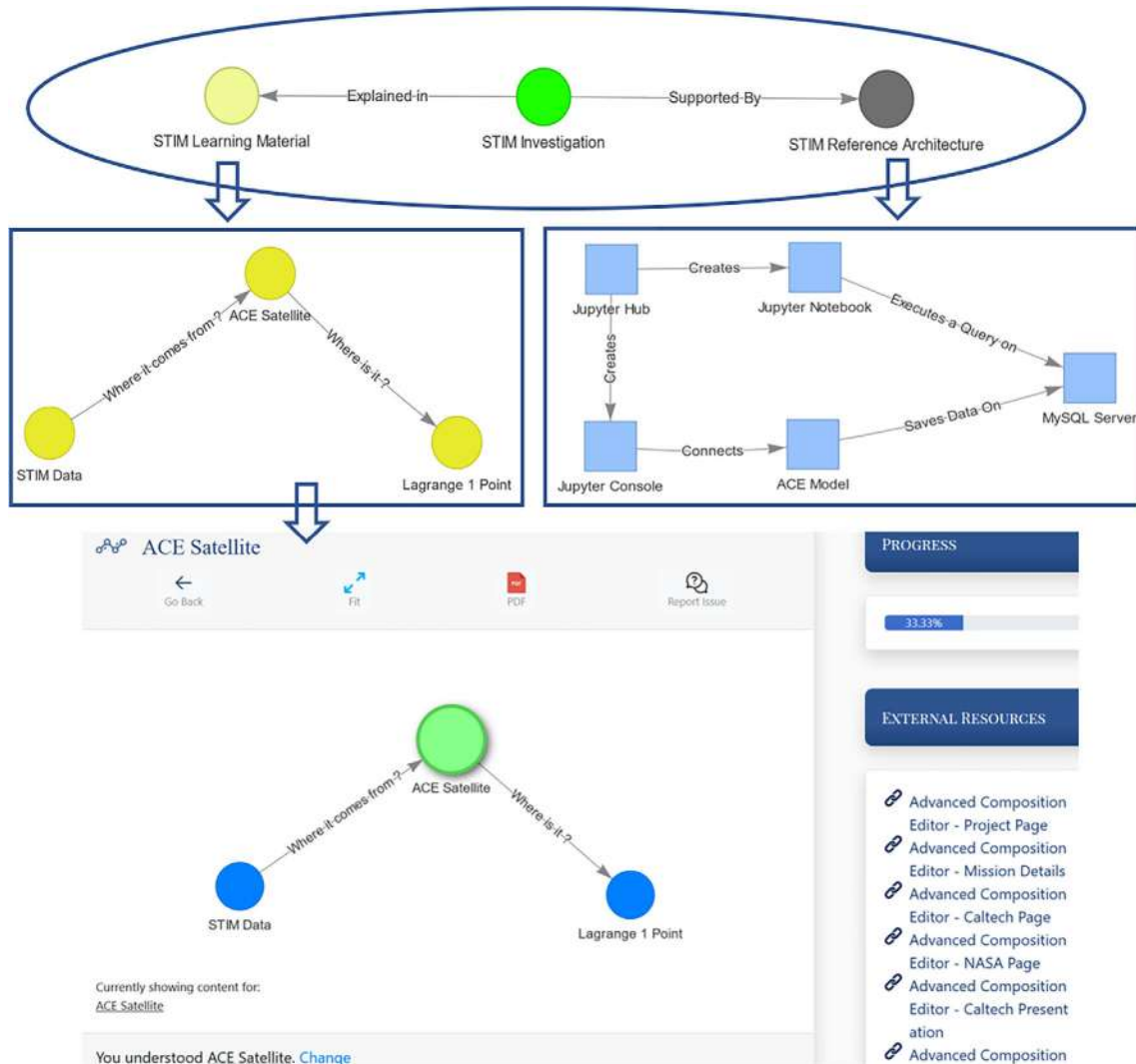

FIGURE 13 Sample query results from the STIM model using Jupyter Notebook

FIGURE 14 Learning paths of the STIM case study in SMARTEST

curves (STIM learning material) and the reference architecture (STIM reference architecture). The two graphs in the middle show the learning paths for the related science concepts (left hand side) and the implemented reference architecture (right hand side). Nodes are used to describe research topics, learning material and reference architectures, while relationships between nodes are described as edges. SMARTEST also allows linking a node of one graph to an entirely different graph, in order to represent different levels of abstraction of related concepts. Such a feature is used to link the nodes on the top of Figure 14 to the graphs of the learning material and the reference architecture. This initial description of the STIM science case provides the first implementation of the three conceptual views introduced in Figure 2: the matchmaking among a science case, a reference architecture and some learning material on the science case, including, for example, the location and nature of the data sources.

SMARTEST supports the e-learning process in the following way: First, the learner is presented with an overview of the structure of the concepts and the related learning paths. This is displayed in nodes, which contain the links to the learning material and the edges which represent the relationships between the various concepts. The learner can start navigating the graph at any point. However, an entry node is usually provided as a recommended first step. The learner follows the relationship between the graphs and the nodes. As an example, she/he would start with the graph at the top of Figure 14 and open either the node marked as STIM Learning Material to investigate the theoretical aspects, or the node marked as STIM reference architecture to learn about the technical details of the implementation. Once a node is clicked, it leads automatically to the selected graph (middle layer of Figure 14) where the learner can navigate through the different topics. For example, the graph on the left-hand side of the middle layer displays a learning path related to STIM data. The learner's journey commences by studying STIM learning material first, looking at the content of the node "STIM data". The edge from this node to the subsequent node ("ACE Satellite") on this learning curve indicates the relationship between these concepts, namely that STIM Data come from the ACE Satellite. Finally, the learner is guided to investigate the content of the node

"Lagrange 1 point". For each node a list of external links to learning material is offered on the right-hand side of the GUI, and the learner can also mark the node as understood (or not yet understood) to reflect her/his progress (bottom layer of Figure 14). If the learner requires any additional information, SMARTEST offers the possibility of sending a message specific to the node to the creator of the content.

7 | CONCLUSIONS AND FUTURE WORK

This paper presented a novel concept of a science gateway framework based on cloud-based reference architectures with embedded e-learning support. The proposed framework goes significantly beyond the state of the art of science gateways by allowing the creation, publication, selection and execution of reference architectures that can each incorporate various components, from end-user interfaces to complex analytical, optimization or simulation modules, and data access mechanisms. Once deployed, reference architectures can be utilized by scientists as on-demand science gateways. Additionally, the framework incorporates a Knowledge repository exchange and learning module that provides structured and flexible e-learning support for various user profiles.

The implementation of the framework is currently ongoing in the PITHIA-NRF project, utilizing building blocks of open-source technologies, implemented in previous and ongoing projects. To provide evidence for the feasibility of the proposed framework, a couple of case studies have been implemented and presented in the paper, demonstrating the automated deployment, multilevel autoscaling and embedded e-learning support capabilities of the framework.

Future work will concentrate on both the technical implementation of the framework in the form of the PITHIA-NRF e-Science Center, and further refinement of the various building blocks to provide higher level automation and more flexibility to users. Some components (e.g., e-Science Center GUI, reference architecture composer) need to be fully implemented, while others, such as user management, repository handling, reference architecture launching and instantiation, require refinement and integration. On the other hand, significant research challenges are also ahead of us, demanding more investigation. For example, supporting the automated composition of reference architectures from individual microservices or reference architecture fragments, or the automated generation of learning graphs related to a particular reference architecture, are some of the many challenges.

DATA AVAILABILITY STATEMENT

All components of the presented science gateway framework implementation and the application examples of Section 6 are open source and available in GitHub under https://github.com/UoW-CPC/.

ORCID

Tamas Kiss https://orcid.org/0000-0002-3241-633X
James DesLauriers https://orcid.org/0000-0003-0336-3213

REFERENCES

1. Technical area on science gateways; 2021. Accessed April 14, 2021. https://sites.google.com/site/ieeesciencegateways/
2. Foster I, Kesselman C, Tuecke S. The anatomy of the grid: enabling scalable virtual organizations. *Int J High Perform Comput Appl.* 2001;15(3):200-222. doi:10.1177/109434200101500302
3. Kacsuk P, Farkas Z, Kozlovszky M, et al. WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities. *J Grid Comput.* 2012;10(4):601-630. doi:10.1007/s10723-012-9240-5
4. Mclennan M, Kennell R. HUBzero: a platform for dissemination and collaboration in computational science and engineering. *Comput Sci Eng.* 2010;12(2):48-52. doi:10.1109/MCSE.2010.41
5. Barbera R, Bruno R, Fargetta M, La Rocca G. The Catania science gateway framework in the ReCaS environment. In: Andronico G, Bellotti R, De Nardo G, Laccetti G, Maggi G, Merola L, Russo G, Silvestris L, Tangaro S, Tassi E, eds. *High Performance Scientific Computing Using Distributed Infrastructures.* World Scientific; 2017:473-483.

6. Reference architecture examples, diagrams, and best practices; 2021. Accessed May 15, 2021. https://aws.amazon.com/architecture/

7. Pintye I, Kail E, Kacsuk P, Lovas R. Big data and machine learning framework for clouds and its usage for text classification. *Concurr Comput*. 2020;33. doi:10.1002/cpe.6164

8. Kubernetes. Production-grade container orchestration; 2021. Accessed May 07, 2021. https://kubernetes.io/

9. Terraform by HashiCorp; 2021. Accessed April 14, 2021. https://www.terraform.io/

10. Kovács J, Kacsuk P. Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. *J Grid Comput*. 2018;16(1):19-37. doi:10.1007/s10723-017-9421-3

11. Kiss T, Kacsuk P, Kovacs J, et al. MiCADO—microservice-based cloud application-level dynamic orchestrator. *Futur Gener Comput Syst*. 2019;94:937-946. doi:10.1016/j.future.2017.09.050

12. Topology and orchestration specification for cloud applications version 1.0; 2017. Accessed March 30, 2017. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

13. OASIS | Advancing open standards for the information society; 2018. Accessed March 18, 2018. https://www.oasis-open.org/

14. Klimeck G, Mclennan M, Brophy SB, Adams Iii GB, Lundstrom MS. Advancing education and research in nanotechnology; 2008. Accessed April 14, 2021. [Online]. nanoHUB.org. http://docs.lib.purdue.edu/nanodocs/80

15. The PITHIA-NRF project — PITHIA-NRF research infrastucture; 2021. Accessed May 07, 2021. https://pithia-nrf.eu/

16. Kiss T, Bolotov A, Pierantoni G, et al. Science gateways with embedded ontology-based E-learning support. Proceedings of the 12th International Workshop on Science Gateways; 2020.

17. Open Grid Forum. https://www.ogf.org/ogf/

18. Distributed resource management application API version 2 (DRMAA) - draft 8 | semantic scholar; 2021. Accessed May 13, 2021. https://www.semanticscholar.org/paper/Distributed-Resource-Management-Application-API-2-8-Troger-Brobst/d09836331969dd49d7345612710b403c586893ee

19. Anjomshoaa A, Brisard F, Drescher M, et al. Job submission description language (JSDL) specification, version 1.0; 2011.

20. Andreozzi S, Burke S, Ehm F, et al. GLUE specification v. 2.0. *Open Grid Forum Recomm Doc*. 2009. http://lup.lub.lu.se/record/1454660

21. Foster I, Kesselman C. Globus project: a status report. *Futur Gener Comput Syst*. 1999;15(5):607-621. doi:10.1016/S0167-739X(99)00013-8

22. S. Jha, H. Kaiser, A. Merzky, and O. Weidner, "Grid interoperability at the application level using SAGA.," 2007, pp. 584–591.

23. Ardizzone V, Barbera R, Calanducci A, et al. The DECIDE science gateway. *J Grid Comput*. 2012;10(4):689-707. doi:10.1007/s10723-012-9242-3

24. Bruno R, Barbera R, Fargetta M, Rotondo R, Anagnostou A, Taylor SJE. Science reproducibility and reusability with FutureGateway and a Zenodo-like repository: the PALMS experiment; October 2019. doi: 10.15161/OAR.IT/23512.

25. Gesing S, Zentner M, Clark S, Stirm C, Haley B. Hubzero®: novel concepts applied to established computing infrastructures to address communities' needs; 2019. doi: 10.1145/3332186.3332238

26. Assante M, Candela L, Castelli D, et al. The gCube system: delivering virtual research environments as-a-service. *Futur Gener Comput Syst*. 2019;95:445-453. doi:10.1016/J.FUTURE.2018.10.035

27. Digital experience software tailored to your needs | liferay. Accessed September 15, 2021. https://www.liferay.com/

28. Assante M, Candela L, Castelli D, et al. Enacting open science by D4Science. *Futur Gener Comput Syst*. 2019;101:555-563. doi:10.1016/J.FUTURE.2019.05.063

29. Foster ED, Deardorff A. Open science framework (OSF). *J Med Lib Assoc*. 2017;105(2):203–206. doi:10.5195/JMLA.2017.88

30. Liew CS, Atkinson M, Galea M, Ang TF, Martin P, van Hemert J. Scientific workflows: moving across paradigms. *ACM Comput Surv*. 2017;49(4):66-39. doi:10.1145/3012429

31. Goble CA, Bhagat J, Aleksejevs S, et al. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucl Acids Res*. 2010;38(Web Server Issue):W677. doi:10.1093/NAR/GKQ429

32. VRE4EIC – Home; 2021. Accessed May 13, 2021. https://vre4eic.ercim.eu/

33. A Europe-wide interoperable virtual research environment to empower multidisciplinary research communities and accelerate innovation and collaboration deliverable D3.5 final architecture design; 2021. Accessed May 13, 2021. http://www.vre4eic.eu/

34. Pierce ME, Miller MA, Brookes EH, et al. Towards a science gateway reference architecture; 2018. Accessed May 15, 2021. https://nanohub.org/citations

35. Kalyanam R, Zhao L, Song C, et al. MyGeoHub—a sustainable and evolving geospatial science gateway. *Futur Gener Comput Syst*. 2019;94:820-832. doi:10.1016/j.future.2018.02.005

36. Wolfram: computation meets knowledge; 2021. Accessed May 07, 2021. https://www.wolfram.com/

37. Wolfram mathematica: modern technical computing. Accessed May 11, 2021. https://www.wolfram.com/mathematica/

38. School education gateway – homepage; 2021. Accessed May 07, 2021. https://www.schooleducationgateway.eu/en/pub/index.htm

39. BBC - ontologies – home; 2021. Accessed May 07, 2021. https://www.bbc.co.uk/ontologies

40. RDF - semantic web standards; 2021. Accessed May 07, 2021. https://www.w3.org/RDF/

41. DCMI: home; 2021. Accessed May 07, 2021. https://www.dublincore.org/

42. Pradhan M, Fuchs C, Noll J. Deployment architecture for accessing smart city and coalition assets for multi-agency HADR operations; June 2020. doi: 10.1109/WF-IoT48130.2020.9221431

43. Belehaki A, Tsagouri I, Altadill D, et al. An overview of methodologies for real-time detection, characterisation and tracking of traveling ionospheric disturbances developed in the TechTIDE project. *J Space Weather Space Clim*. 2020;10:42. doi:10.1051/swsc/2020043

44. Merka J, Narock TW, Szabo A. Navigating through SPASE to heliospheric and magnetospheric data. *Earth Sci Inform*. 2008;1(1):35-42. doi:10.1007/s12145-008-0004-5

45. Belehaki A, James S, Hapgood M, et al. The ESPAS e-infrastructure: access to data from near-earth space. *Adv Space Res*. 2016;58:1177-1200. doi:10.1016/j.asr.2016.06.014

46. Jacquey C, Génot V, Budnik E, et al. AMDA, automated multi-dataset analysis: a web-based service provided by the CDPP. *Astrophys Space Sci Proc*. 2010;(202469):239-247. doi:10.1007/978-90-481-3499-1&uscore;16

47. Project Jupyter | Home; 2021. Accessed May 10, 2021. https://jupyter.org/

48. Home | CloudiFacturing; 2021. Accessed May 10, 2021. https://www.cloudifacturing.eu/

49. Bolotov A, Pierantoni G, Yerashenia N, et al. SMARTEST - knowledge and learning repository. Proceedings of the 4th Student-STAFF Research Conference 2020 School of Computer Science and Engineering SSRC2020.

50. SMARTEST - home page. Accessed May 10, 2021. https://smartestknowledge.org/

51. Home - DIGITbrain. Accessed May 10, 2021. https://digitbrain.eu/

52. DesLauriers J, Kiss T, Ariyattu RC, et al. Cloud apps to-go: cloud portability with TOSCA and MiCADO. *Concurr Comput.* 2020;33:e6093. doi:10.1002/cpe.6093

53. Pierantoni G, Kiss T, Terstyanszky G, DesLauriers J, Gesmier G, Van Dang H. Describing and processing topology and quality of service parameters of applications in the cloud. *J Grid Comput.* 2020;18(4):761-778. doi:10.1007/s10723-020-09524-0

54. Prometheus - monitoring system & time series database; Accessed May 10, 2021. https://prometheus.io/

55. Kovács J. Supporting programmable autoscaling rules for containers and virtual machines on clouds. *J Grid Comput.* 2019;17(4):813-829. doi:10.1007/s10723-019-09488-w

56. Theoretical origins of concept maps, how to construct them, and uses in education. Accessed May 10, 2021. https://www.researchgate.net/publication/228761562_Theoretical_origins_of_concept_maps_how_to_construct_them_and_uses_in_education

57. Kinchin IM. *Visualising Powerful Knowledge to Develop the Expert Student: A Knowledge Structures Perspective on Teaching and Learning at University.* Sense Publishers; 2016.

58. De Smet C, Schellens T, De Wever B, Brandt-Pomares P, Valcke M. The design and implementation of learning paths in a learning management system. *Interact Learn Environ.* 2016;24(6):1076-1096. doi:10.1080/10494820.2014.951059

59. Boyce S, Pahl C. Developing domain ontologies for course content. *J Educ Technol Soc.* 2007;10(3):275-288.

60. Bolotov A, Pierantoni G, Wisudha A, Abduraimova Z, Fee DCY. EnAbled: a psychology profile based academic compass to build and navigate students' learning paths. Proceedings of the CEUR Workshop Proceedings; Vol. 2193, 2018.

61. Graph database platform | graph database management system | Neo4j. Accessed May 11, 2021. https://neo4j.com/

62. protégé. Accessed May 11, 2021. https://protege.stanford.edu/

63. Rubin DL, Knublauch H, Fergerson RW, Dameron O, Musen MA. Protege-owl: creating ontology-driven reasoning applications with the web ontology language. Proceedings of the AMIA Annual Symposium; Vol. 2005:1179; American Medical Informatics Association. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1560433/

64. EMGUM [Online]. https://github.com/UoW-CPC/cfg-emgum-api

65. OpenID. http://openid.net/

66. Keycloak. Accessed May 11, 2021 [Online]. https://www.keycloak.org/

67. GitHub - UoW-CPC/k8s2adt: translate K8s manifest(s) into a MiCADO ADT; 2021. Accessed September 15, 2021. https://github.com/UoW-CPC/k8s2adt

68. Nexus repository OSS - software component management | sonatype; 2021. Accessed May 15, 2021. https://www.sonatype.com/products/repository-oss

69. Apache Hadoop. Accessed May 10, 2021. https://hadoop.apache.org/

70. Dean J, Ghemawat S. MapReduce. *Commun ACM.* 2008;51(1):107-113. doi:10.1145/1327452.1327492

71. Homepage - national observatory of athens; 2021. Accessed May 10, 2021. https://www.noa.gr/en/

72. Tsagouri I, Belehaki A. An upgrade of the solar-wind-driven empirical model for the middle latitude ionospheric storm-time response. *J Atmos Sol-Terr Phys.* 2008;70(16):2061-2076. doi:10.1016/j.jastp.2008.09.010

73. Perkel JM. Why Jupyter is data scientists' computational notebook of choice. *Nature.* 2018;563(7729):145-146. doi:10.1038/d41586-018-07196-1

74. Sarajlic S, Chastang J, Marru S, Fischer J, Lowe M. Scaling JupyterHub using kubernetes on jetstream cloud: platform as a service for research and educational initiatives in the atmospheric sciences. Proceedings of the Practice and Experience on Advanced Research Computing; 2018:1-4. doi: 10.1145/3219104.3229249