# Easing Scientific Computing and Federated Management in the Cloud with OCCI

Zdeněk Šustr[1], Diego Scardaci[2,3], Jiří Sitera[1], Boris Parák[1] and Víctor Méndez Muñoz[4]

[1]*Department of Distributed Computing, CESNET, Zikova 4, 160 00, Praha 6, Czech Republic*

[2]*European Grid Initiative, Science Park 140, 1098 XG Amsterdam, Netherlands*

[3]*INFN Sezione di Catania, Via S. Sofia 64, I-95123 Catania, Italy*

[4]*Computer Architecture & Operating Systems Department (CAOS), Universitat Autònoma de Barcelona (UAB), Bellaterra, Spain*

Keywords: Federated Cloud, Cloud Standards, Cloud Interoperability, Cloud Solution Design Patterns, Cloud Application Architectures, Cloud Middleware Frameworks, Open Cloud Computing Interface.

Abstract: One of the benefits of OCCI stems from simplifying the life of developers aiming to integrate multiple cloud managers. It provides them with a single protocol to abstract the differences between cloud service implementations used on sites run by different providers. This comes particularly handy in federated clouds, such as the EGI Federated Cloud Platform, which bring together providers who run different cloud management platforms on their sites: most notably OpenNebula, OpenStack, or Synnefo. Thanks to the wealth of approaches and tools now available to developers of virtual resource management solutions, different paths may be chosen, ranging from a small-scale use of an existing command line client or single-user graphical interface, to libraries ready for integration with large workload management frameworks and job submission portals relied on by large science communities across Europe. From lone wolves in the long-tail of science to virtual organizations counting thousands of users, OCCI simplifies their life through standardization, unification, and simplification. Hence cloud applications based on OCCI can focus on user specifications, saving cost and reaching a robust development life-cycle. To demonstrate this, the paper shows several EGI Federated Cloud experiences, demonstrating the possible approaches and design principles.

## 1 INTRODUCTION

OCCI, the Open Cloud Computing Interface (Nyrén et al., 2011; Nyrén et al., 2011; Metsch and Edmonds, 2011b; Metsch and Edmonds, 2011a), is a standard developed by the Open Grid Forum to standardize virtual resource management in a cloud site (OGF, 2016). It was released in 2011 and several implementations and real-world deployments followed. Among others, OCCI also became the standard of choice for the management of virtualized resources in EGI's Federated Cloud Platform (Wallom et al., 2015).

The EGI Federated Cloud Platform has been conceived as an alternative way to access EGI's considerable computing resources, which are primarily accessed through grid middleware. Cloud-flavoured services are meant to:

- Attract user communities relying on tools that are not easily ported to grid but can scale well in the cloud – for instance tools only available for operating systems not compatible with available grid infrastructures, tools distributed by vendors in the form of virtual machine images, etc.
- Lower the acceptance threshold for users trained in using their existing environment without having to change it or re-qualify for a different one.

On reflection, this suits the need of users in the "long tail of science" who typically lack the resources to have their solutions tailored or adjusted to the grid and cannot invest in re-training for different tools or environments. Even relatively small-scale resources, available in the EGI Federated Cloud Platform at the time of writing, can fit the needs of these "long-tail" communities.

Firstly, this article will introduce relevant implementations of OCCI, both on the server and client side (Section 2). Secondly, Section 3 will introduce a scale of cloud usage patterns relying on OCCI for interoperability and abstraction, and also briefly expand on other mechanisms users must ensure beyond the basic OCCI functionality for their solutions to be truly productive. Finally, Section 4 will briefly outline new developments, and Section 5 will sum up the topic of experience with OCCI in the area of scientific computing.

## 2 OCCI IMPLEMENTATIONS

OCCI was gradually gaining support throughout its lifetime and there are currently multiple implementations available on the server side as well as the client side. At the time of writing, the version of OCCI supported by most implementations mentioned below is 1.1. They will all follow their own, independent schedules to adopt OCCI 1.2, which is due to come into effect shortly.

### 2.1 Server-side OCCI Support

Server-side OCCI implementations originate mostly from service providers who wish to make their resources available in a standardized way. Where applicable, the OCCI interfaces are being pushed upstream to Cloud Management Framework developers.

In the EGI Federated Cloud Platform, multiple sites make their cloud resources available to users. Although different contributing sites employ different Cloud Management Frameworks (CMFs), OCCI (plus an X.509-based authentication mechanism) is the unifying factor, allowing users to access any site in a uniform way, indeed, without actually knowing what flavor of CMF is installed on the end-point (Fig. 1). There are several of such server-side frameworks whose resources can be managed through OCCI:

- *OpenStack*, which comes with an OCCI interface (*occi-os*) as part of the stack and is soon due to receive an all-new implementation in the form of OOI – the OpenStack OCCI Interface (García et al., 2016).
- *Synnefo* with its own OCCI implementation (GRNET, 2016).
- *OpenNebula*, which is accessed through an OCCI translation service – the *rOCCI-server* (Parák et al., 2014).

Although there is currently no such site participating in the Federated Cloud Platform, it is also technically possible to use OCCI to manage resources in a fourth type of cloud management framework:

- *Amazon Web Services (AWS)*, which can be accessed with OCCI through the *rOCCI-server* as well (CESNET, 2016).

Aside from the aforementioned, there are other cloud services which support OCCI or its subsets (Fogbow, 2016), or which are going to be made OCCI-capable in the foreseeable future. For instance, the development work is underway to implement an OCCI translation layer for Microsoft Azure.

For the sake of completeness, it is also worth mentioning that there are OCCI implementations currently being used in existing solutions, but apparently no longer maintained. They are:

- *pyssf* – the Service Sharing Facility (PySSF, 2016), which strives to provide building blocks for grid, cloud, cluster or HPC services.
- *occi-os* – the original OpenStack OCCI interface (occi os, 2016), which is gradually being replaced with OOI, already mentioned above.

While the latter is scheduled to phase out in favor of OOI, the former – it must be acknowledged – has never been declared as discontinued, and development activity may resume; especially with the introduction of OCCI 1.2 specification.

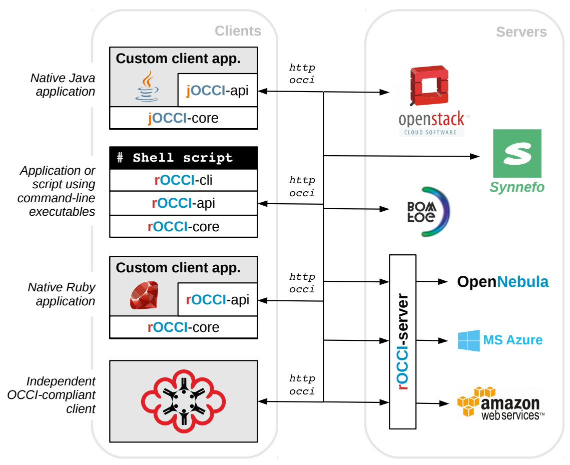

Figure 1: Ecosystem of OCCI implementations envisioned for products relevant to EGI Federated Cloud and its user communities.

### 2.2 Client-side Tools and Libraries Available

OCCI is a text-based protocol. Although there are unofficial extensions to support OCCI transport through message queues (Limmer et al., 2014), the only officially standardized transport method for OCCI is HTTP. Therefore it is technically possible to interact with an OCCI-capable server through a generic HTTP client, and implement at least a subset of OCCI functionality with a few pre-formatted OCCI messages. This approach is often used in interoperability testing, where a fixed set of actions is tested over and over, but is highly unsuitable for real world applications wherein a general-purpose client is required.

At present, there are at least two independent client-side stacks available: the *rOCCI framework* and the *jOCCI* library (Kimle et al., 2015).

The *rOCCI framework* offers a set of Ruby libraries (*rOCCI-core* and *rOCCI-api*), wherein the former implements the OCCI class structure, parsing and rendering, while the latter implements HTTP transport. Calls to these libraries may be used by native Ruby clients to interact with any OCCI-enabled cloud site.

The *rOCCI framework* also comes with a general-purpose command line client – the *rOCCI-cli* – which can be used by end users to complete simple tasks, or to wrap around with scripts. Some user communities also choose to wrap around the command line interface if the programming language of choice cannot use Ruby or Java libraries directly.

Finally the *jOCCI* is an independent OCCI implementation in Java. Like *rOCCI*, it exposes the OCCI class structure, parsing and rendering functions, plus HTTP transport. It is intended primarily for use by orchestration or submission frameworks implemented in Java.

## 3 CLOUD USAGE PATTERNS WITH OCCI

The ability to manage cloud resources in a standardized way is useful in many different scenarios. Although the benefits are greatest with automation, OCCI can also be used by end users if necessary.

From the users' perspective, the main advantage of using OCCI is that a single client-side solution works with multiple server-side implementations. That provides for better scaling across heterogeneous resources and helps protect users' investment into developing their client-side tools as it allows for seamless transition between providers. There are other possible approaches, but the discussion of their relative merits is out of the scope of this article. A comparison is made for instance in (Parák and Šustr, 2014). The OCCI based approach was chosen as a key part of the EGI Cloud Federation Platform architecture by the EGI Federated Cloud Task as described in the EGI Federated Cloud Blueprint (EGI, 2014).

The following sections discuss distinct OCCI usage patterns as seen mainly in communities gathered around the EGI Federated Cloud.

### 3.1 Science Gateways and Workload Management Frameworks

The great potential offered by OCCI is evident when high-level tools are built on or connected to its interface. End users of PaaS and SaaS services that exploit cloud resources through OCCI can benefit from a large amount of resources belonging to heterogeneous cloud sites adopting different cloud management frameworks.

In the context of the EGI Federated Cloud (Wallom et al., 2015; del Castillo et al., 2015), this opportunity has been exploited by many technical providers who have extended their platforms to support the OCCI standard providing an alternative and automated way to access the Federated Cloud, hiding the IaaS layer from their users.

This has greatly increased the added value that the EGI Federated Cloud offers to its users, who can choose to access its resources at IaaS level or via one of the various OCCI-compliant tools now available.

Notable high-level tools currently available in the EGI ecosystem supporting OCCI are:

- for PaaS: Vac/VCycle (VCycle, 2016), Slipstream (Slipstream, 2016) and Infrastructure Manager (IM) (IM, 2016)
- for SaaS development frameworks: VMDirac (VMDirac, 2016) (also discussed in greater detail in 3.1.1), COMPSs (Lezzi et al., 2014), the Catania Science Gateway Framework (Ardizzone et al., 2012) and WS-PGRADE (WS-PGRADE, 2016).

The Vac/VCycle cloud infrastructure broker has been developed by the University of Manchester and has been adopted by CERN, who have developed an OCCI connector for their WLCG experiments. Slipstream (SixSq) is the central broker of the Helix-Nebula infrastructure (Helix-Nebula, 2016). IM (by Universitat Politècnica de València) is a tool that deploys complex and customized virtual infrastructures on IaaS Cloud deployments, automating the VMI selection, deployment, configuration, software installation, monitoring and update of Virtual Appliances. IM will be used to implement the broker features in the EGI Cloud Marketplace (EGI-CM, 2016) hosted by the AppDB (AppDB, 2016).

VMDirac, the Catania Science Gateway Framework by INFN, and WS-PGRADE by SZTAKI are well-known tools to develop Science Gateways in the grid environment. These have been extended to exploit cloud resources, too. COMPSs by BSC is a programming framework, composed of a programming model and an execution runtime which supports it, whose main objective is to ease the development of applications for distributed environments keeping the programmers unaware of the execution environment and parallelization details.

Furthermore, the EGI Federated Cloud and its access model based on OCCI has been envisioned as the ideal infrastructure to host several community platforms exposing services tailored for specific user groups. Integration of several platforms into the infrastructure is currently underway; the most relevant are listed below:

- The Geohazard and Hydrology Thematic Exploitation Platforms (TEPs) (ECEO, 2016) developed by the European Space Agency (ESA) (ESA, 2016);
- the D4Science infrastructure (D4Science, 2016) that hosts more than 25 Virtual Research Environments to serve the biological, ecological, environmental, and statistical communities world-wide;
- a uniform platform for international astronomy research collaboration developed in collaboration with the Canadian Advanced Network for Astronomical Research (CANFAR) (CANFAR, 2016);
- selected EPOS thematic core services (TCS) (EPOS, 2016);

#### 3.1.1 dirac.egi.eu Case Study

Dirac is introduced in greater depth as an example of a typical OCCI-enabled scientific gateway.

DIRAC platform eases scientific computing by overlaying distributed computing resources in a transparent manner to the end-user. DIRAC integrates the principles of grid and cloud computing (Foster et al., 2009) and its ecology for virtual organizations (VOs) (Foster and Kesselman, 1999). Depending on context, this category is commonly referred to as either Virtual Research Environments (VREs) (Carusi and Reimer, 2010) or Scientific Gateways (SGs) (Wilkins-Diehr, 2007), among other terms. The *dirac.egi.eu* service is a deployment of a DIRAC instance with a particular setup for multiple communities in EGI, accessing the resources on a per-VO basis, providing storage and computing allocation, dealing with grid and cloud in an interoperable manner from a single access point.

The interfaces to connect to *dirac.egi.eu* are (1) a Web Portal serving as a generic human interface, (2) a DIRAC client oriented to simplify bulk operations with the command line, and (3) a python API and a REST interface to be used by VREs/SGs in order to delegate resource and service management in the DIRAC platform, thus focusing on the high level requirements in their communities (Mendez et al., 2014).

Hereby, *dirac.egi.eu* engages two roles: *the one-off user* and *the VRE/SG support*. The former actually uses DIRAC Web portal as a basic VRE for job submission, retrieval and data management operations, such as searching in a catalog, data transfers or downloading results. The latter is disaggregating the concept of VRE/SG in two levels, the back-end is the DIRAC platform for all the resource management and service science. The front-end, the specific VRE/SG providing a Web environment for housing, indexing, and retrieving specifics of large data sets, as well as, eventually, suppling leverage Web 2.0 technologies and social networking solutions to give researchers a collaboration environment for resource discovery. This disaggregated VRE model eases the social component of collaborations built in a walnut shell of well-known technologies, which are dealing with the increasing complexity of the interoperability between different resources. Therefore, in the years to follow, generic VRE from scratch will be avoided, relaying infrastructure management in standard practices and tools such as the DIRAC platform.

So far, *dirac.egi.eu* is connecting several third party infrastructures and services in a coherent manner, accepting logical job and data management request, which are processed with the corresponding computing and storage resources to obtain results. In this sense, a central part of DIRAC framework is a *Workload Management System* (WMS), based on the pull job scheduling model, also known as the pilot job model (Casajus et al., 2010). The pull model only submits a pre-allocation container, which performs basic sanity checks of the execution environment, integrates heterogeneous resources and pulls the job from a central queue. Thus, proactive measures (before job matching) can be applied in case of problems, and several transfers and platform issues are avoided. The second asset is provided from the DIRAC WMS service side, and the late binding of jobs to resources allows further optimization by global load balancing.

Experience using *dirac.egi.eu* as a back-end service for the WeNMR VRE (Bencivenni et al., 2014) in structural biology and life science, and the VINA virtual drug screening SG (Jaghoori et al., 2015), shows an efficiency improvement up to 99%, saving on the usage of important resources and alleviating daily operations in comparison with the previous push job model.

In the particular case of cloud computing, DIRAC treats the virtual resources as standard worker nodes and assigns work to them, following the same pull job scheduling model. For this purpose, WMS is using the DIRAC cloud extension named VMDIRAC (Méndez et al., 2013) to pre-allocate virtual machine resources. This VMDIRAC scheduler has finally converged in a common cloud driver technology for most of the IaaS – a rOCCI client wrapped in DIRAC Python code.

There are certain historical highlights worth mentioning. Initially, VMDIRAC was designed for AWS boto python library, then the *pre-standard OCCI 0.8 for OpenNebula*. VMDIRAC 1.0 was adopting a flexible architecture for federated hybrid cloud, also integrating OpenStack by means of *libcloud*. This heterogeneous cloud driver ecosystem was lacking in software convergence. Then, OCCI 1.1 became available, with rOCCI on the client side and also as an OCCI interface for OpenNebula, followed by more OCCI implementations for OpenStack and other cloud management frameworks. Since OCCI APIs are in Ruby or Java, and DIRAC is in Python, the decision was made was to adopt the *rOCCI-cli* client. Among its other traits, *rOCCI-cli* is the most frequently used interface, thus, the best updated.

From *dirac.egi.eu* experience the main assets in the rOCCI adoption are:

- The releasing convergence and updated features; once a new feature is out, then it is ready for all the underlying IaaS technologies.
- The alignment with EGI Federated cloud computing model, for example, the straightforward implementation of the credentials based on X.509, avoiding dealing with HTML text request to different native OCCI servers with their particular details, as always *devil is in the details*.

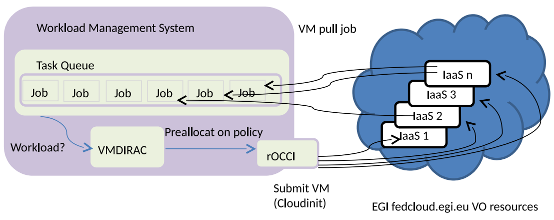

Figure 2: dirac.egi.eu stage with fedcloud.egi.eu.

Currently, *dirac.egi.eu* has two main stages supported by rOCCI client connecting EGI Federated cloud resources. Figure 2 shows a first stage with the pull job scheduling model draw for the *dirac.egi.eu* service and *fedcloud.egi.eu* VO resources. Any user belonging to *fedcloud.egi.eu* is automatically included in *dirac.egi.eu* by polling the VOMS server, so that they can use the service through the Web portal or through the other interfaces. VMDIRAC submits VMs to the IaaS endpoints of the EGI FedCloud, with a credential of the VO proxy. VMs are contextualized with *cloud-init*, which is supported by OCCI implementations and at the same time it is required to be supported by the cloud manager and the requested image. It is a single method to dynamically contextualize VMs for different IaaS endpoints and DIRAC client install in VO basis.

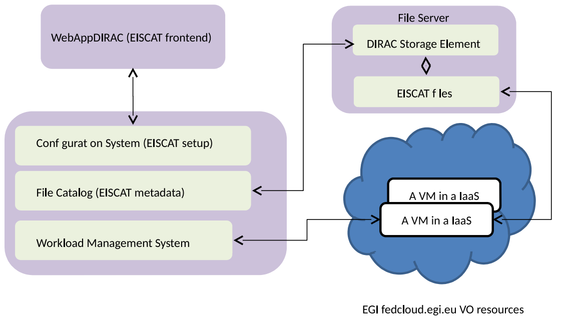
WebAppDIRAC (EISCAT frontend)
File Server
DIRAC Storage Element
EISCAT f les
Conf gurat on System (EISCAT setup)
File Catalog (EISCAT metadata)
Workload Management System
A VM in a IaaS
A VM in a IaaS
EGI fedcloud.egi.eu VO resources

Figure 3: EISCAT use case in *dirac.egi.eu* with fedcloud.egi.eu.

Figure 3 shows the EISCAT use case of the *fedcloud.egi.eu* stage of Figure 2. EISCAT is a work in progress in *dirac.egi.eu* for the data processing of incoherent scatter radar systems to study the interaction between the Sun and the Earth as revealed by disturbances in the ionosphere and magnetosphere. The overall architecture in Figure 3 is showing the integration of storage and computing resources, with the explained disaggregated VRE schema. In the top left, the EISCAT user front-end is a web application developed for the community data and metadata management needs, using for this purpose the WebAppDIRAC framework, a tornado development kit completely integrated in DIRAC engine, or alternatively another VRE Portal accessing by APIs to the *dirac.egi.eu* back-end service. An EISCAT File Catalog contains all the metadata of the project. The data are stored in an EISCAT filesystem, accessed by a DIRAC Storage Element built on the top of such filesystem, securing connection with VO credentials. The data processing jobs are submitted from the EISCAT front-end to *dirac.egi.eu* back-end to pre-allocate VMs in EGI FedCloud, then matching jobs, downloading input data from EISCAT file server, processing the workload and uploading output files to the EISCAT file server.

A second stage overall architecture is in Figure 4, connecting *dirac.egi.eu* with cloud resources of the *training.egi.eu* VO. This stage is used in EGI training sessions, starting with a pre-configured VM deployment, including a DIRAC client and dynamically requesting a temporal PUSP proxy to the EGI service (Fernandez et al., 2015). DIRAC is matching the VM user and the corresponding PUSP proxy with a pool of generic DIRAC users. Then, following the same pull job scheduling model preallocating VM *train-*

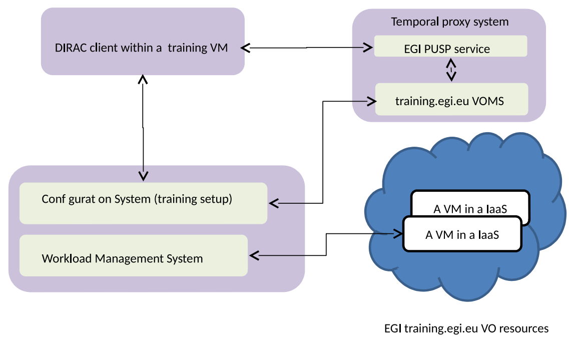
DIRAC client within a training VM
Temporal proxy system
EGI PUSP service
training.egi.eu VOMS
Conf gurat on System (training setup)
Workload Management System
A VM in a IaaS
A VM in a IaaS
EGI training.egi.eu VO resources

Figure 4: dirac.egi.eu stage with *training.egi.eu*.

*ing.egi.eu* resources to latter match the temporal user jobs.

## 3.2 Processing in a Remote Data-holding Domain

A standard-compliant management interface can be exposed by a cloud site – regardless even of whether it is part of a wider cloud federation or not – to allow users to run prepared and pre-approved workload in the provider's domain. This may come particularly useful in view of the needs of new communities.

An example of this is bioinformatics, where the principal problem with specific and very strict data policies, often subject to legal restrictions, can be solved by moving the computation into the administrative domain holding the data. In other words: as long as the data must not leave a certain domain, let the computation follow that data into the domain and run there.

Providing researchers with templates for virtual compute resources to process those data gives, on one hand, the providers a chance to precisely choose what tools will be available and how the researcher will be able to use them, while on the other hand it gives the user more flexibility and is easier to implement than a full-featured interface that would have to assume all possible methods of processing the data beforehand.

Offering an open standard-based interface to do that is only logical, and becomes necessary if such site wishes to be included in a heterogeneous federation.

## 3.3 Small-scale and One-off Use of the Cloud

OCCI is primarily a machine-to-machine protocol wherein the client side is expected to (and usually does) expose a user-friendly front-end that hide the internals of OCCI-based communication behind a portal. Additional scheduling and workload management logic is also often hidden behind the scenes. Yet use cases also exist with so little dynamism and so simple requirements that users can easily set up their virtual resources themselves, "by hand".

This is most frequently done through the command line interface (*rOCCI-cli*), which makes it possible to set attributes for all common OCCI actions as arguments in the command line, and send them to a selected OCCI endpoint. Though somewhat crude, this is sufficient for use cases wherein a reasonably low number of virtual machines – a few dozen at maximum – can be started and kept track of manually. Such machines are then usually created at the same time at the beginning of the experiment, not necessarily at a single site within the federation, and disposed of once the experiment ends.

This is a "poor-man's" approach to cloud resource provisioning and management, but it is fully sufficient for user groups or single users who require a fixed set of resources for batch processing over a limited period of time – typical in the often cited "long tail of science". The advantage of using OCCI here is the same as in use cases where OCCI is used by automated client-side tools – users do not need to care about the flavor of cloud management framework they find on the server side. The interaction is always the same.

## 3.4 Beyond the Current Scope of OCCI

Although users can use OCCI to instantiate resources on demand across large heterogeneous infrastructures, there are additional tasks they need to complete before they can use their freshly spawned resources efficiently, and which cannot be taken care of through OCCI at the moment. The purpose of this section is not a thorough overview, but rather a warning for potential users to think about the missing pieces, and also showing where future OCCI might be addressing some of the areas that are currently out of its scope

First and foremost, one needs to realize that procuring compute resources from an IaaS cloud leaves the user with just that – a set of virtual machines. A tool to manage the workload is always needed:

- One that runs outside the pool of cloud resources:
  - either one that is aware of the nature of the resources, having them possibly instantiated itself, (such as in use cases discussed in Section 3.1) in which case all the components required are probably already in place for the user and the infrastructure is complete.
  - or such that treats the virtual resources as standard worker nodes and can assign work to

them. This can be a local resource management system such as TORQUE, or even a simple script. It just needs to be made aware of the resources, and then users can submit their work. Here, OCCI could be of further use in the future, when an intended OCCI Monitoring extension is finished.

- One that runs inside the pool of cloud resources: That borders, or even falls into the scope of PaaS (Platform as a Service) cloud provisioning model. As of the upcoming OCCI 1.2 specification, an OCCI PaaS extension will be available, applicable exactly to these cases.

OCCI is going to gradually address these more complex workflows, and provide extensions to cover relevant areas for larger-scale computing. At the moment, service providers, integrators as well as users must keep in mind that with OCCI, they are only managing their resources while managing their workload is up to them.

## 4 FUTURE WORK

There may be usage patterns as yet unexplored, but there are also several patterns that are already known and will be made possible with the OCCI 1.2 specification. Therefore much of the foreseeable future work, both among cloud service developers and user community specialists, is going to revolve around OCCI 1.2 adoption.

One of the most visible new usage patterns will be enabled by improved support for resource template prototyping. That is, it will be possible to derive templates from existing virtual machines, and have them instantiated multiple times not only on the local site, but with sufficient support also across the federation.

OCCI 1.2 will also introduce a JSON rendering specification, which will make it possible to describe resource collections and other complicated concepts with better precision, avoiding the possible ambiguity of `text/occi` rendering used to-date.

Finally, OCCI 1.2 is first OCCI release to introduce a PaaS specification. This will have to be carefully assessed, and candidates for adoption will have to be selected from among the plethora of server-side products first. Then it will be possible for user communities to experiment with the new model of cloud service provisioning.

## 5 CONCLUSIONS

OCCI proves to be an invaluable binding component in heterogeneous cloud federations. Although it may require additional effort from service providers to achieve and maintain standard compliance, it simplifies the life and work of user communities, especially the small-scale ones coming from the long tail of science. Traditionally short on technical support and development staff, or lacking it completely, an interoperability standard such as OCCI helps them by providing a single interface to interact with multiple different cloud specifications, helping them protect their investment into whatever workload management solution they have developed, and avoid vendor lock-in. With their client side being OCCI-capable, they can simply move among provider sites, or scale across multiple providers, without having to adjust their own processes.

## ACKNOWLEDGMENTS

This work is co-funded by the EGI-Engage project (Horizon 2020) under Grant number 654142.

## REFERENCES

AppDB (2016). [Online] https://appdb.egi.eu/. Accessed: March, 2016

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

Bencivenni, M., Michelotto, D., Alfieri, R., Brunetti, R., Ceccanti, A., Cesini, D., Costantini, A., Fattibene, E., Gaido, L., Misurelli, G., Ronchieri, E., Salomoni, D., Veronesi, P., Venturi, V., and Vistoli, M. C. (2014). Accessing grid and cloud services through a scientific web portal. *Journal of Grid Computing*,13(2):159–175

CANFAR (2016). [Online] http://www.canfar.phys.uvic.ca/canfar/. Accessed: March, 2016

Carusi, A. and Reimer, T. (2010). Virtual research environment collaborative landscape study.

Casajus, A., Graciani, R., Paterson, S., Tsaregorodtsev, A., and the Lhcb Dirac Team (2010). Dirac pilot framework and the dirac workload management system. *Journal of Physics: Conference Series*, 219(6):062049.

CESNET (2016). rOCCI ec2 backend documentation. [Online] https://wiki.egi.eu/ wiki/rOCCI:EC2_Backend. Accessed: March, 2016

D4Science (2016). [Online] https://www.d4science.org/. Accessed: March 10, 2016.

del Castillo, E. F., Scardaci, D., and Álvaro Lopéz García (2015). The egi federated cloud e-infrastructure. *Procedia Computer Sceince*, (68):196–205.

ECEO (2016). [Online] http://www.ca3-uninova.org/project_eceo. Accessed: March, 2016

EGI (2014). Egi federated cloud blueprint. [Online] https://documents.egi.eu/document/2091. Accessed: March, 2016

EGI-CM (2016). [Online] https://appdb.egi.eu/browse/cloud. Accessed: March, 2016.

EPOS (2016). [Online] http://www.epos-eu.org/. Accessed: March, 2016.

ESA (2016). [Online] http://www.esa.int/ESA. Accessed: March, 2016.

Fernandez, E., Scardaci, D., Sipos, G., Chen, Y., and Wallom, D. C. (2015). The user support programme and the training infrastructure of the egi federated cloud. In *the 2015 International Conference on High Performance Computing & Simulation (HPCS 2015)*. IEEE.

Fogbow (2016). [Online] http://www.fogbowcloud.org/. Accessed: March, 2016.

Foster, I. and Kesselman, C., editors (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Foster, I. T., Zhao, Y., Raicu, I., and Lu, S. (2009). Cloud computing and grid computing 360-degree compared. *CoRR*, abs/0901.0131.

García, A. L., del Castillo, E. F., and Fernández, P. O. (2016). ooi: Openstack occi interface. In *SoftwareX(2016)*. ScienceDirect.

GRNET (2016). Synnefo OCCI documentation. [Online] https://www.synnefo.org/docs/snf-occi/latest/. Accessed: March, 2016.

Helix-Nebula (2016). [Online] http://www.helix-nebula.eu/. Accessed: Helix-Nebula

IM (2016). [Online] http://www.grycap.upv.es/im/index.php. Accessed: March, 2016.

Jaghoori, M. M., Altena, A. J. V., Bleijlevens, B., Ramezani, S., Font, J. L., and Olabarriaga, S. D. (2015). A multi-infrastructure gateway for virtual drug screening. *Concurrency and Computation: Practice and Experience*, 27(16):4478–4490.

Kimle, M., Parák, B., and Šustr, Z. (2015). jOCCI – general-purpose OCCI client library in java. In *ISGC15, The International Symposium on Grids and Clouds 2015*. PoS.

Lezzi, D., Lordan, F., Rafanell, R., and Badia, R. M. (2014). *Euro-Par 2013: Parallel Processing Workshops: BigDataCloud, DIHC, FedICI, HeteroPar, HiBB, LSDVE, MHPC, OMHI, PADABS, PROPER, Resilience, ROME, and UCHPC 2013, Aachen, Germany, August 26-27, 2013. Revised Selected Papers*, chapter Execution of Scientific Workflows on Federated Multi-cloud Infrastructures, pages 136–145. Springer Berlin Heidelberg, Berlin, Heidelberg.

Limmer, S., Srba, M., and Fey, D. (2014). *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II*, chapter Performance Investigation and Tuning in the Interoperable Cloud4E Platform, pages 85–96. Springer International Publishing, Cham.

Méndez, V., Casaju's, A., Graciani, R., and Fernández, V. (2013). How to run scientific applications with dirac in federated hybrid clouds. ADVCOMP.

Mendez, V., Casajus Ramo, A., Graciani Diaz, R., and Tsaregorodstsev, A. (2014). Powering Distributed Applications with DIRAC Engine. In *Proceedings of the symposium "International Symposium on Grids and Clouds (ISGC) 2014"(ISGC2014). 23-28 March, 2014. Academia Sinica, Taipei, Taiwan*, page 42.

Metsch, T. and Edmonds, A. (2011a). Open cloud computing interface – http rendering. [Online] http://ogf.org/documents/GFD.185.pdf. Accessed: March 10, 2016.

Metsch, T. and Edmonds, A. (2011b). Open cloud computing interface – infrastructure. [Online] http://ogf.org/documents/GFD.184.pdf. Accessed: March 10, 2016.

Nyrén, R., Edmonds, A., Papaspyrou, A., and Metsch, T. (2011). OCCI specification. [Online] http://occi-wg.org/about/specification. Accessed: March, 2016.

Nyrén, R., Edmonds, A., Papaspyrou, A., and Metsch, T. (2011). Open cloud computing interface – core. [Online] http://ogf.org/documents/GFD.183.pdf. Accessed: March, 2016.

occi os (2016). [Online] https://wiki.openstack.org/wiki/Occi. Accessed: March, 2016.

OGF (2016). Open grid forum. [Online] http://www.ogf.org. Accessed: March, 2016.

Parák, B., Šustr, Z., Feldhaus, F., Kasprzak, P., and Srba, M. (2014). The rOCCI project – providing cloud interoperability with OCCI 1.1. In *ISGC14, The International Symposium on Grids and Clouds 2014*. PoS.

Parák, B. and Šustr, Z. (2014). Challenges in achieving iaas cloud interoperability across multiple cloud management frameworks. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 404–411.

PySSF (2016). [Online] http://pyssf.sourceforge.net/. Accessed: March, 2016.

Slipstream (2016). [Online] http://sixsq.com/products/ slipstream/. Accessed: March, 2016.

VCycle (2016). [Online] https://www.gridpp.ac.uk/vcycle/. Accessed: March, 2016.

VMDirac (2016). [Online] https://github.com/ DIRACGrid/VMDIRAC/wiki. Accessed: March, 2016.

Wallom, D., Turilli, M., Drescher, M., Scardaci, D., and Newhouse, S. (2015). Federating infrastructure as a service cloud computing systems to create a uniform e-infrastructure for research. In *IEEE 11th International Conference on e-Science, 2015*. IEEE.

Wilkins-Diehr, N. (2007). Special issue: Science gateways - common community interfaces to grid resources. *Concurrency and Computation: Practice and Experience*, 19(6):743–749.

WS-PGRADE (2016). [Online] https://guse-cloud-gateway.sztaki.hu/. Accessed: March, 2016.