The rOCCI Project – Providing Cloud Interoperability with OCCI 1.1

Boris Parák[*],[a] **Zdeněk Šustr,**[a] **Florian Feldhaus,**[b] **Piotr Kasprzak**[c] **and Maik Srba**[c]
[a]*CESNET z. s. p. o.*
Zikova 4, 160 00, Praha 6, Czech Republic
[b]*NetApp Deutschland GmbH*
Gladbecker Str. 5, 40472, Düsseldorf, Germany
[c]*GWDG mbH*
Am Faßberg 11, 37077, Göttingen, Germany
E-mail: boris.parak@cesnet.cz, zdenek.sustr@cesnet.cz, florian.feldhaus@gmail.com, piotr.kasprzak@gwdg.de, maik.srba@gwdg.de

OCCI (Open Cloud Computing Interface) is an open protocol for management of tasks in the cloud environment focused on integration, portability and interoperability with a high degree of extensibility. It is designed to bridge differences between various cloud platforms and provide common ground for users and developers alike. The rOCCI framework, originally developed by GWDG, later adopted and now maintained by CESNET, was written to simplify implementation of the OCCI 1.1 protocol in Ruby and later provided the base for working client and server components giving OCCI support to multiple cloud platforms while ensuring interoperability with other existing implementations. The initial server-side component provided basic functionality and served as a proof of concept when it was adopted by the EGI Federated Cloud Task and was chosen to act as the designated virtual machine management interface. This led to further funding from the EGI-InSPIRE project, development of a full featured client and a new rOCCI-server suitable for production environment. It has also prompted further proliferation of the OCCI protocol, spawned multiple connector/backend implementations and provided developers with valuable feedback and opportunities to test their own implementations of the OCCI standard. This paper aims to provide basic information about the OCCI protocol, introduce its implementation in rOCCI, describe some of the core functionality provided by the rOCCI client and rOCCI-server along with their impact on interoperability in the cloud environment. It also briefly examines its use in the EGI Federated Cloud Task environment and explores the possibility of further integration with other cloud platforms. All this with interoperability in mind. The paper also describes a carefully chosen subset of problems encountered whilst trying to provide interoperability with multiple cloud platforms through the use of the OCCI protocol, with real-world examples and chosen solutions.

International Symposium on Grids and Clouds (ISGC) 2014
23 – 28 March, 2014
Taipei, Taiwan

[*]Speaker.

1. Introduction

The landscape of cloud computing continues to grow and existing offerings have become mature in the past few years. Cloud services are being rapidly adopted by companies and end users. Often a single provider dominates a specific kind of service. The interfaces offered to the customers are usually proprietary and often these interfaces become de-facto standards due to the market domination. Competitors try to offer compatible interfaces based on these de-facto standards, but as many services are rapidly evolving, this often leads to vendor lock-ins, and the market leader can use changes to the interface as a weapon against competitors.

To give customers a choice between different cloud providers, an interoperable interface is required to seamlessly change from one provider to another. Ultimately, cloud users would like to federate resources and services from different providers to improve characteristics such as reliability, cost efficiency and world wide distribution of services. *Cloud Federation* stands for the transparent connection of different cloud service providers to larger entities. Among the pressing issues are interfaces to discover and manage the services of a provider, accounting/billing, monitoring as well as authorization and authentication.

To federate between different cloud providers, interoperable interfaces are needed for different use cases. NIST came up with the SAJACC use cases for Infrastructure as a Service [1]. The EGI Federated Cloud has extended these use cases for an European-wide cloud infrastructure [2]. The use cases define scenarios for managing virtual machines, data management, capability detection of different cloud providers, accounting, authentication and OS image management.

Due to their pioneering role and market dominance, EC2 and S3 interfaces of the Amazon Web Services present a de-facto standard in IaaS environments [3]. The EC2 interface offers a wide range of functionality for managing virtual infrastructures. The S3 interface offers functionality to manage object storage. Several other projects such as Eucalyptus, OpenStack and OpenNebula have implemented subsets of the functionality of EC2 and S3.

In contrast to these de-facto standards, the de-jure standards OCCI, CIMI and CDMI have been established in recent years. CIMI is a standard by DMTF, targeted at IaaS management and offering fine-grained functionality to manage virtual infrastructures [4]. OCCI by OGF is a more generic approach offering a high level of flexibility to describe and manage arbitrary cloud services [5]. A basic model for IaaS management is available, extendable by each provider to offer specific functionality. CDMI by SNIA [6] is an ISO standard targeting object storage management. It offers functionality similar to S3, but also allows for detailed access control management as well as the incorporation of established enterprise file systems such as NFS and CIFS.

Several tools have been created to allow users to access cloud services in a uniform way, helping them to overcome interoperability issues. These can be grouped into two categories.

On one hand, there are client libraries offering a basic and uniform interface to different cloud providers. Differences between various cloud APIs are hidden on the client side. They often allow interaction with provider-specific features using direct access to native interfaces. Prominent examples for this group are the *jClouds* library for Java [7] and the *fog.io* gem for Ruby [8].

The second group consists of tools acting as collectors for different cloud services and offering their functionality using a common server-side API. Prominent example for this group is *DeltaCloud* offering REST APIs DeltaCloud, CIMI and EC2 [9]. Another tool is *CompatibleOne*, which offers an OCCI interface to manage federated IaaS and PaaS environment including SLAs and brokering [10].

2. The OCCI Standard

The Open Cloud Computing Interface is a set of open community-led specifications delivered through the Open Grid Forum [11]. It was originally designed with IaaS clouds in mind, allowing for the development of interoperable tools for common tasks including virtual machine deployment, autonomous scaling and monitoring. The standard has since evolved with a strong focus on integration, portability, interoperability and a high degree of extensibility to a boundary-level protocol and API that acts as a service front-end to the provider's internal management framework [12] – see Figure 1.

The Open Cloud Computing Interface (OCCI) standard specification consists of three separately published documents:

1. GFD.183 – OCCI Core [13]
2. GFD.184 – OCCI Infrastructure [14]
3. GFD.185 – OCCI HTTP Rendering [15]

Each document covers a specific part of the standard. Work on additional documents such as OCCI JSON Rendering, OCCI Accounting or OCCI Monitoring is in progress.

2.1 OCCI Core

The OCCI Core model defines a representation of instance types (class diagram shown in Figure 2), which can be manipulated through an OCCI rendering implementation. It is an abstraction of real-world resources, including the means to identify, classify, associate and extend those resources [13].

One of the basic features of the OCCI model is its extensibility.

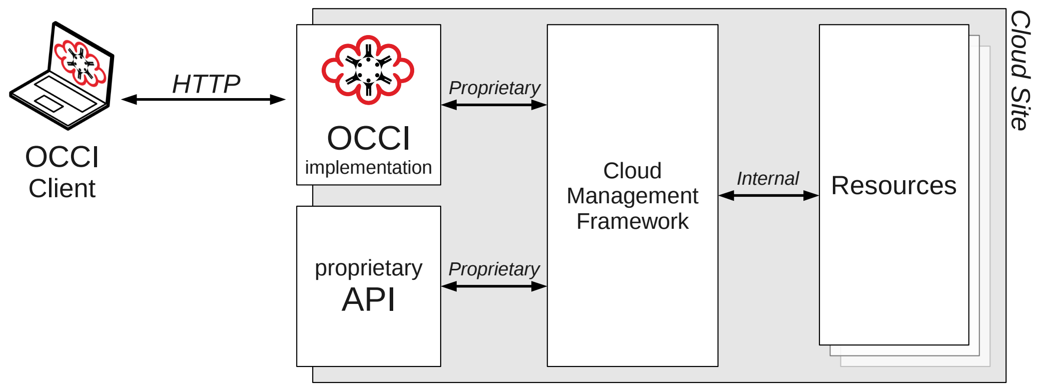

Figure 1: Accessing an arbitrary Cloud Management Framework through OCCI

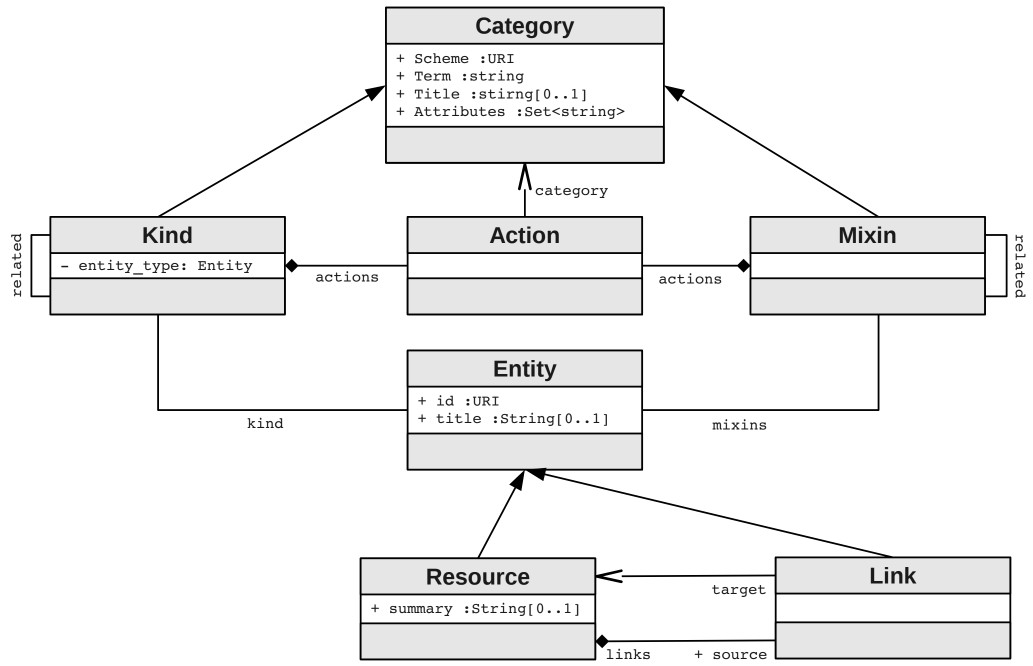

Figure 2: OCCI class diagram

2.2 OCCI Infrastructure

The OCCI Infrastructure model (Figure 3) details an OCCI Core extension to manage an infrastructure as a service (IaaS) platform. It is a collection of classes to provide storage, network and fundamental compute resources [14].

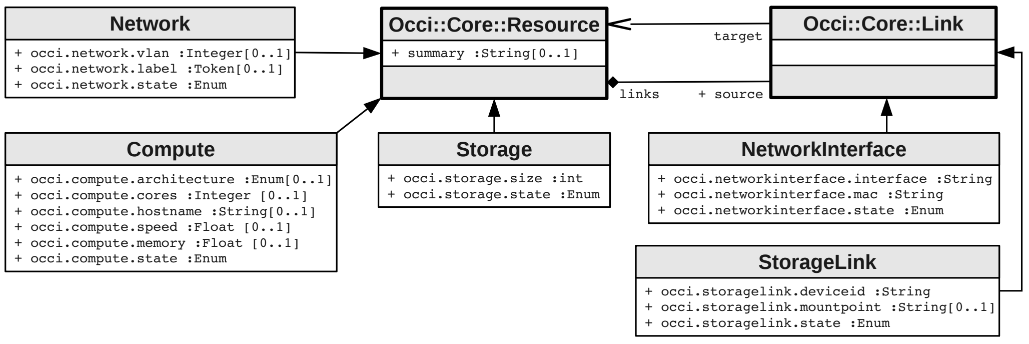
Network
+ occi.network.vlan :Integer[0..1]
+ occi.network.label :Token[0..1]
+ occi.network.state :Enum
Occi::Core::Resource
+ summary :String[0..1]
target
Occi::Core::Link
links
+ source
Compute
+ occi.compute.architecture :Enum[0..1]
+ occi.compute.cores :Integer [0..1]
+ occi.compute.hostname :String[0..1]
+ occi.compute.speed :Float [0..1]
+ occi.compute.memory :Float [0..1]
+ occi.compute.state :Enum
Storage
+ occi.storage.size :int
+ occi.storage.state :Enum
NetworkInterface
+ occi.networkinterface.interface :String
+ occi.networkinterface.mac :String
+ occi.networkinterface.state :Enum
StorageLink
+ occi.storagelink.deviceid :String
+ occi.storagelink.mountpoint :String[0..1]
+ occi.storagelink.state :Enum

Figure 3: OCCI infrastructure class diagram

2.3 OCCI HTTP Rendering

OCCI has been designed as a REST interface. Definition for the HTTP Transport is given in OCCI HTTP Rendering specification [15]. This interface allows OCCI clients to interact with OCCI services. Available REST commands correspond with common HTTP requests: POST, GET, PUT and DELETE. Messages can be rendered as plain text either into HTTP headers or into HTTP body. An example of an OCCI message rendered in plain text is shown in the following excerpt:

```
Category: compute;
          scheme="http://schemas.ogf.org/occi/infrastructure#";
          class="kind"
Link: </users/foo/compute/b9ff813e-fee5-4a9d-b839-673f39746096?action=start>;
      rel="http://schemas.ogf.org/occi/infrastructure/compute/action#start"
X-OCCI-Attribute: occi.core.id="urn:uuid:b9ff813e-fee5-4a9d-b839-673f39746096"
X-OCCI-Attribute: occi.core.title="My VM"
X-OCCI-Attribute: occi.compute.architecture="x86"
X-OCCI-Attribute: occi.compute.state="inactive"
X-OCCI-Attribute: occi.compute.speed=1.33
X-OCCI-Attribute: occi.compute.memory=2.0
X-OCCI-Attribute: occi.compute.cores=2
X-OCCI-Attribute: occi.compute.hostname="compute1.example.org"
```

2.4 Capabilities & Usage

OCCI, as described by the OGF standard, offers a well-rounded set of key capabilities for IaaS cloud management. These include, but are not limited to, the following:

- Advertising a model containing a list of all available features implemented by the given cloud management framework (kinds, mixins, actions, etc.)
- Creating compute, network and storage resources
- Querying compute, network and storage resources
- Triggering actions on compute, network and storage resources
- Destroying compute, network and storage resources
- Attaching & detaching disks and network interfaces

The aforementioned capabilities are sufficient for performing basic management tasks in IaaS-oriented clouds and can be easily extended to cover platform-specific functionality when necessary. To illustrate how OCCI is used to perform said management tasks, a simplified message flow example for creating a compute instance is available in Figure 4.

3. The rOCCI Project

The rOCCI project aims to provide general multi-purpose tools for improving interoperability in the cloud. rOCCI has been funded by the European Commission Information Society and, more recently, by the EGI-InSPIRE project as an open source project distributed under the Apache License, Version 2.0 [16].

rOCCI started as an internal project at the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), which is a university computing centre for the Georg-August-Universität Göttingen, and a computing and IT competence centre for the Max Planck Society. rOCCI was later adopted by the EGI Federated Cloud Task Force as the main tool for virtual machine management, providing interoperability and necessary abstraction in a heterogeneous federated environment. The project is currently lead and developed by CESNET, a research institute

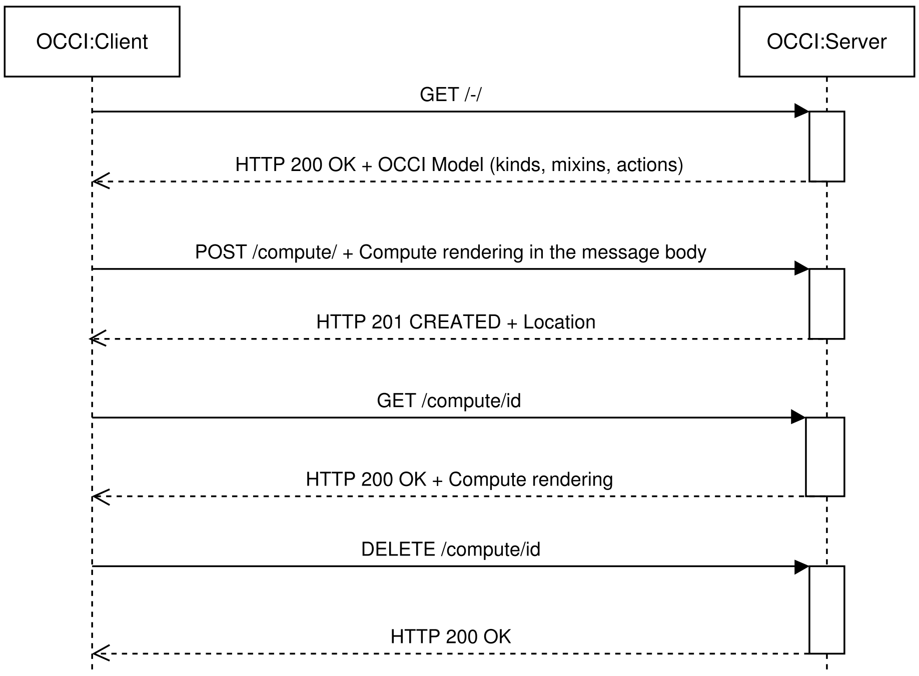
OCCI:Client
OCCI:Server
GET /-/
HTTP 200 OK + OCCI Model (kinds, mixins, actions)
POST /compute/ + Compute rendering in the message body
HTTP 201 CREATED + Location
GET /compute/id
HTTP 200 OK + Compute rendering
DELETE /compute/id
HTTP 200 OK

Figure 4: OCCI message flow example

acting as the operator of the Czech National Research Network (NREN) and the Czech National Grid Initiative (NGI). rOCCI development still receives assistance from GWDG.

The rOCCI framework, the main product of the rOCCI project, is a collection of libraries implemented in the Ruby programming language, and end-user products based on these libraries. It provides tools for both developers wishing to build new services or integrate OCCI support into existing applications, and end-users wishing to use production-ready tools. All components are written in the Ruby programming language.

3.1 Components of the rOCCI Framework

The rOCCI framework consists of two libraries: *rOCCI-core* and *rOCCI-api*, and two end-user products: *rOCCI-cli* and *rOCCI-server*. Together, they provide a fully featured solution adding OCCI support to IaaS-based clouds. Individual roles of the components are illustrated in Figure 5. A brief description of each component follows.

3.1.1 rOCCI-core

rOCCI-core is a central component of the framework. It implements classes representing entities defined by the OCCI standard, provides parsing and rendering capabilities from/to multiple message formats. Currently supported, as both input and output formats, are plain text, HTTP headers and JSON.

It also introduces various helper classes such as *Collection* or *Model,* simplifying the handling and rendering of complex OCCI messages, provides advanced logging and attribute validation facilities. Using Ruby's metaprogramming capabilities, the core is able to "extend itself" with new

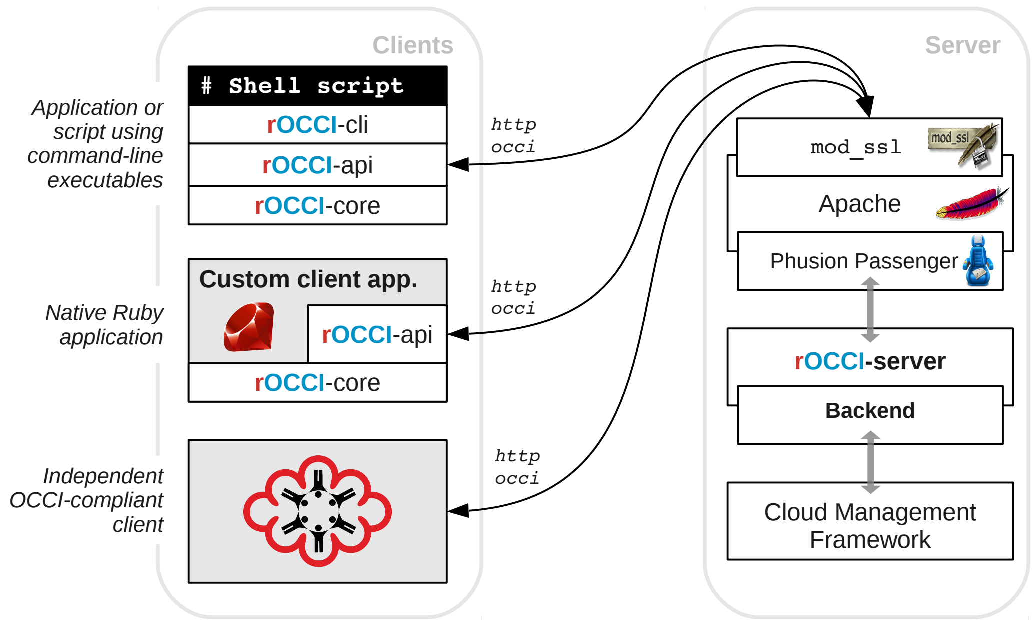

Figure 5: rOCCI components in use on the client side and server side

classes and definitions provided dynamically at runtime. This works well with OCCI's inherent extensibility.

By default, the core implements entities defined in *OCCI Core* [13] and *OCCI Infrastructure* [14], and message formats (i.e., renderings) defined in *OCCI HTTP Rendering* [15] and the *OCCI JSON Rendering* draft [17].

The code is organized in modules and classes closely resembling parts of the OCCI standard. This approach makes *rOCCI-core* easy to understand and also helps new developers with a pre-existing knowledge of OCCI. See for example the following interaction in an interactive Ruby shell:

```
2.0.0-p353 :001 > compute = Occi::Infrastructure::Compute.new
2.0.0-p353 :002 > compute.mixins << Occi::Infrastructure::OsTpl.mixin
2.0.0-p353 :003 > compute.cores = 4
2.0.0-p353 :004 > compute.memory = 1024
2.0.0-p353 :005 > compute.hostname = 'compute1.example.org'
2.0.0-p353 :006 > compute.to_text

Category: compute;scheme="http://schemas.ogf.org/occi/infrastructure#";class="kind"
Category: os_tpl;scheme="http://schemas.ogf.org/occi/infrastructure#";class="mixin"
X-OCCI-Attribute: occi.core.id="8c3da50f-6240-43cd-a50d-52362f754664"
X-OCCI-Attribute: occi.compute.cores=4
X-OCCI-Attribute: occi.compute.hostname="compute1.example.org"
X-OCCI-Attribute: occi.compute.memory=1024
```

3.1.2 rOCCI-api

rOCCI-api builds on top of *rOCCI-core* and implements support for transport protocols and corresponding authentication methods. It also provides an extended set of various helpers simplifying the use of *rOCCI-core* and targeting the development of client-side applications and tools. HTTP is currently the only supported transport protocol, support for AMQP is on the way.

rOCCI-api implements a pluggable authentication mechanism capable of fallbacks. Every authentication plug-in can declare a set of alternatives to be used in case of failure. This concept is capable of masking differences between various cloud frameworks implementing OCCI using their own authentication schemes. Currently supported authentication mechanisms are:

- Basic
- X.509
- VOMS
- OpenStack Keystone (as a fallback for basic, X.509 and VOMS)

rOCCI-api exposes methods which can be used to establish connections with the chosen end-points, list available resources, provide details about available resources, create instances, trigger actions on existing instances and destroy them. Aside from the basic functionality, it provides methods simplifying look-ups based on identifiers, schemes and terms, logging facilities etc.

To get a better understanding of *rOCCI-api*'s intended use and the basic concepts behind it, see the following interaction outlining a simple use case:

```
2.0.0-p353 :001 > extend Occi::Api::Dsl
2.0.0-p353 :002 >
2.0.0-p353 :003 > connect :http do |config|
2.0.0-p353 :004 >   config.endpoint = 'http://localhost:3000/'
2.0.0-p353 :005 > end
2.0.0-p353 :006 > compute = resource "compute"
2.0.0-p353 :007 > os_tpl = mixin "debian7", "os_tpl"
2.0.0-p353 :008 > resource_tpl = mixin "small", "resource_tpl"
2.0.0-p353 :009 > compute.mixins << os_tpl << resource_tpl
2.0.0-p353 :010 > compute.title = "My rOCCI VM"
2.0.0-p353 :011 > create compute

http://localhost:3000/compute/19465
```

3.1.3 rOCCI-cli

rOCCI-cli builds on top of *rOCCI-api* and provides a shell-based user interface suitable for advanced end-users. This component implements argument parsers and output formatters, provides basic documentation and usage examples, simplifies common tasks through a system of intelligent lookups and informed guesses. It also provides tools for integration with applications and systems unable to use the underlying libraries directly, such as providing machine-friendly JSON output:

```
occi --endpoint http://localhost:3000 --auth none --action create         \
     --resource compute --mixin os_tpl#debian7 --mixin resource_tpl#small \
     --attribute occi.core.title='TestVM'                                 \
     --context user_data='file:///home/cloud/context.yml'                 \
     --context public_key='file:///home/cloud/.ssh/id_rsa.pub'
```

3.1.4 rOCCI-server

The *rOCCI-server* extends cloud managers, which are not OCCI-compliant natively, with its own OCCI interface. It is based on the rOCCI (Ruby OCCI) Framework. In a typical installation, shown in Figure 6, the *rOCCI-server* is accessed through the Apache HTTP Server. Since

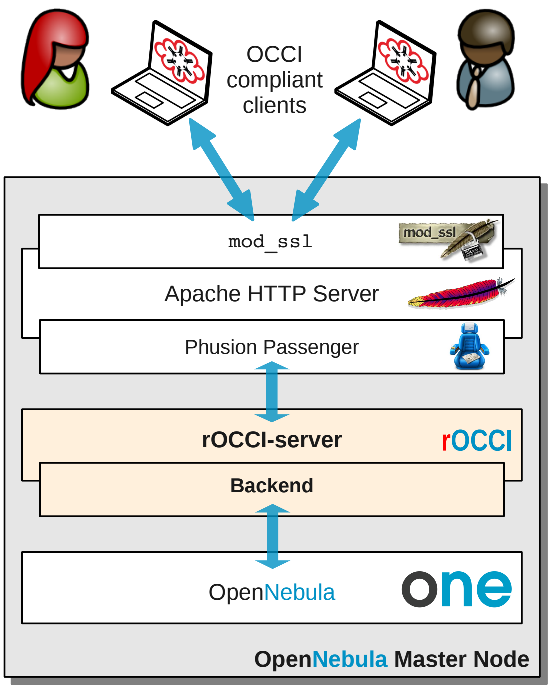

Figure 6: *rOCCI-server* in a typical setup with the OpenNebula Cloud Manager

the OpenNebula backend is the flagship of *rOCCI-server* backend development, *rOCCI-server* is typically found paired with the OpenNebula Cloud Manager.

The following differences from Figure 6, which shows a typical setup as envisioned by *rOCCI-server* developers, may appear in production if the site so prefers:

1. The HTTP server and the Cloud Manager may be located on separate machines. In that case, the *rOCCI-server* is installed with the HTTP server.

2. A different HTTP server can be used. NGiNX [18] is explicitly supported (including passenger). However, Apache is recommended as it provides advanced authentication mechanisms.

3. A different cloud manager can be used. This requires a corresponding backend. Note that at the time of this writing (Spring 2014) only the OpenNebula backend is available, but several others are in preparation.

Figure 7 shows the essentials of *rOCCI-server*'s internal structure. The tasks of the individual components are mostly obvious. Still here are a few notes to clarify:

- Rack middleware components wrap around the core of the application to act upon / augment the passing request and response data.

- In the MVC setup (Model / View / Controller), the Model is used to access CMF rather than traditional data storage, but apart from that, it acts similarly to other applications.

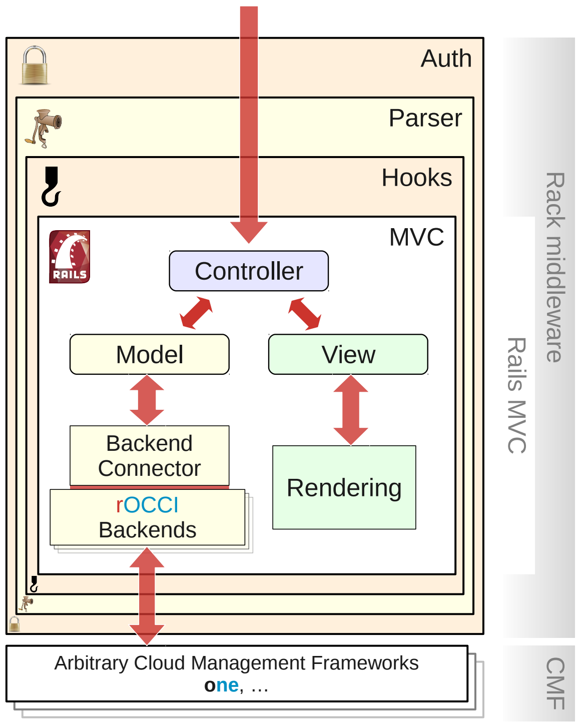

Figure 7: *rOCCI-server* internals

- Methods provided by the backend API map 1:1 on requests defined by the OCCI standard, which simplifies the model greatly.

4. Challenges

Whilst implementing the OCCI standard in the rOCCI framework, several compatibility tests with other implementations were performed, most notably with *OCCI-OS* for OpenStack [19] and *snf-occi* for Synnefo [20]. The rOCCI framework, specifically the *rOCCI-cli* component, played the role of the client, since both of the implementations mentioned above are designed to act in the role of the server.

Tests performed on both implementations can be divided into three groups according to their nature and aspects of the framework being tested:

Transport layer tests – checking supported transport layer protocols (e.g., HTTP, AMQP).

Authentication tests – checking supported authentication mechanisms (e.g., HTTP Basic, HTTP Digest).

Rendering tests – checking syntax of rendered OCCI messages, supported rendering formats (e.g., OCCI text/plain, OCCI text/occi, OCCI application/occi+json).

Feature tests – checking which parts of the OCCI standard are implemented (e.g., query interface, compute, network, storage).

Semantic tests – checking interpretation of complex sets of commands and their effect on resources (e.g., creating resource instances, actions on existing resource instances).

Simplified results and observations made during the testing process, within the scope of this paper, are as follows:

Transport layer tests – both implementations support HTTP, protocol defined by the OCCI standard; no issues were found.

Authentication tests – both implementations support HTTP-compatible authentication methods, however both transparently redirect to an internal authentication element requiring further negotiation using framework-specific APIs.

Rendering tests – both server-side implementations support OCCI text/plain and render valid messages, although minor improvements had to be made in the rOCCI framework, *OCCI-OS* and *snf-occi* alike to achieve that satisfactory result.

Feature tests – both implementations support basic features outlined by the OCCI standard, some advanced features are still in development on all sides; virtual machine life cycle management is possible.

Semantic tests – these have revealed conceptual problems between all implementations, mostly stemming from different underlying cloud management platforms and, at times, rather vague description in the standard; alignment effort is still in progress, in cooperation with the OCCI WG and their work on OCCI version 1.2.

With a considerable effort made on all sides, the rOCCI framework, *OCCI-OS* and *snf-occi* are now mostly compatible, as demonstrated in the use cases outlined below.

5. Use Cases

The rOCCI framework and its various components are actively used by several other projects and communities to facilitate interoperability between IaaS-based cloud management frameworks. These use cases can be divided into two basic groups based on their characteristic use of the framework:

1. Internal integration (the use of libraries)
2. External integration (the use of end-user components)

The first group represents projects and communities choosing to use parts of the rOCCI framework to build their own applications, connectors or other solutions. These include SixSq's Slipstream OCCI connector [21], JSAGA OCCI connector [22], VMDIRAC OCCI connector for the DIRAC interware [23] or the OCCI connector for COMPSs [24]. As a more complex example, the use of rOCCI in the Cloud4E project is explained below.

The second group represents communities using the rOCCI framework as a whole, i.e. as a complete solution for virtual machine management. This includes the use of *rOCCI-cli* and *rOCCI-server* in concert with third party OCCI implementations. The most prominent use case in this category is the EGI Federated Cloud activity.

5.1 EGI Federated Cloud

The EGI Federated Cloud [2] is a seamless grid of academic private clouds and virtualized resources, built around open standards and focusing on the requirements of the scientific community. The resulting infrastructure is a new type of research e-infrastructure, based on the mature federated operations services that make EGI a reliable resource for science.

This infrastructure integrates a wide variety of existing cloud management frameworks and the OCCI standard has been chosen to facilitate interoperability. It is implemented in several components, in different programming languages and by different people. Components from the rOCCI framework are being utilized both as clients and servers.

rOCCI-cli is used as a client for end users and third party developers. It demonstrates real-world interoperability by communicating with different server-side OCCI implementations running on top of various cloud management frameworks. Most notably *OCCI-OS* for OpenStack and *snf-occi* for Synnefo.

rOCCI-server is used as a server-side component providing OCCI support for the OpenNebula cloud management framework. It has been designed to support different framework-specific backends and can be used to provide OCCI support for other cloud management frameworks or IaaS cloud services. Backends for CloudStack, VMWare's vCloud Director and other platforms are being considered, while an experimental backend for Amazon's EC2 is already implemented. Other server-side components implementing OCCI used in EGI Federated Cloud, but independent of rOCCI, are *OCCI-OS* [19] and Synnefo's *snf-occi* [20]. OCCI is targeted as an open standard, not a specific implementation thereof.

5.2 Cloud4E

In the *Cloud4E* Project [25], interfaces are developed to allow vendors of Computer-aided engineering (CAE) software to implement a Software as a Service (SaaS) delivery model in the cloud. This approach lowers the barrier of entry for SMEs to take part in new cloud related business models without having to deal with technical challenges of cloud technology. Instead, only application-specific interfaces for the control of the underlying CAE software need to be implemented. These *connectors* provide the necessary, thin glue code layer to allow the integration of traditional applications, which are not cloud enabled, with an OCCI managed cloud framework.

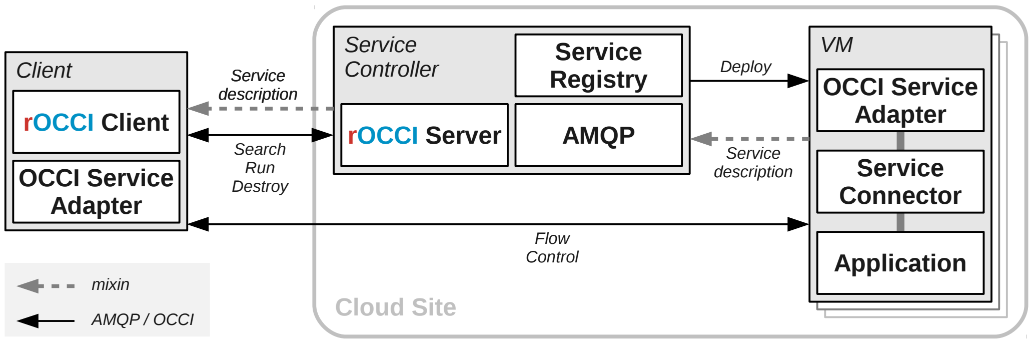

Figure 8: rOCCI in the Cloud4E use case

Furthermore, utilizing the generic model self-description facilities of the OCCI standard, the connector-embedded service descriptions are automatically translated into OCCI mixins and are thus available for discovery by OCCI client software (Figure 8). Regarding client software for service consumers, software providers can make use of generic solutions, which can be extended and adapted to any particular needs. As Cloud4E is not restricted to a specific cloud framework and makes use of rOCCI and AMQP, it provides for high portability of services and avoids vendor lock-in.

6. Future Work

Future developments of the rOCCI Framework are going to cover not only the evolution of the OCCI standard, but also extend interface and backend capabilities. The following extensions are currently being considered:

1. Implementation of emerging extensions and updates to the OCCI specification
    - Upcoming OCCI specification v. 1.2 – announced updated to the current OCCI specification
    - OCCI Accounting Specification – emerging standard to query CMFs for accounting information
    - OCCI Monitoring Specification – emerging standard to query CMFs for status monitoring
    - OCCI PaaS Specification – emerging specification of an interface for Platform as a Service clouds
2. Extending rOCCI interface capabilities
    - XML rendering – emerging standard for OCCI message rendering in XML format, as opposed to existing plain text and JSON rendering alternatives
    - OCCI transport over messaging – AMQP – an alternative to transport over HTTP
3. Implementing additional backends and extensions for popular cloud management frameworks and brokers. Currently considered are:
    - VMWare backend
    - Amazon EC2 backend (experimental implementation is already available)
    - MS Azure backend
    - CompatibleOne extensions
4. Interoperability testing and alignment in combination with other OCCI implementations, namely those available with OpenStack or Synnefo, which both feature their own OCCI interfaces (OCCI-OS [19] or SNF-OCCI [20], respectively).

7. Summary

As clouds become more and more popular and user requirements grow in size and complexity, interoperability is an increasingly important feature for all cloud management frameworks. Whether your local cloud is part of a larger heterogeneous infrastructure (e.g., a cloud federation) or you wish to provide standardized interfaces to local users, using a framework-agnostic protocol is always a good idea. The Open Cloud Computing Interface is an example of such a protocol.

The rOCCI framework provides a comprehensive solution for implementing OCCI compatibility on client's and server's side alike. It is suitable for either internal or external integration in both clients and servers, as evidenced by multiple practical use cases.

rOCCI currently provides essential OCCI compatibility for the OpenNebula Cloud Management Framework in a federated cloud environment, and further development will enable its use with other products over a variety of communication channels.

8. Acknowledgements

Development of the rOCCI Framework is partially funded by the EGI-InSPIRE project through the European Commission's 7th Framework Programme (contract # INFSO-RI-261323).

References

[1] *SAJACC Working Group Recommendations to NIST*, 2013, [Online] Available: http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/SAJACC/SAJACC_Group_Report_Feb_12_2013.pdf [Accessed: August 29, 2014].

[2] European Grid Infrastructure, *Federated Cloud*, [Online] Available: https://www.egi.eu/infrastructure/cloud/ [Accessed: August 29, 2014].

[3] Amazon Web Services, *Amazon EC2*, [Online] Available: https://aws.amazon.com/ec2/ [Accessed: August 29, 2014].

[4] The Distributed Management Task Force, *Cloud Management Initiative*, [Online] Available: http://dmtf.org/standards/cloud [Accessed: August 29, 2014].

[5] R. Nyrén, A. Edmonds, A. Papaspyrou, and T. Metsch, *OCCI specification*, OCCI-WG OGF, 2011. [Online] Available: http://occi-wg.org/about/specification. [Accessed: August 29, 2014].

[6] The Storage Networking Industry Association, *Cloud Data Management Interface (CDMI)*, [Online] Available: http://www.snia.org/cdmi [Accessed: August 29, 2014].

[7] *Apache jclouds*, [Online] Available: http://jclouds.apache.org [Accessed: August 29, 2014].

[8] *fog.io, the Ruby Cloud Services Library*, [Online] Available: http://fog.io [Accessed: August 29, 2014].

[9] *DeltaCloud*, [Online] Available: https://deltacloud.apache.org [Accessed: August 29, 2014].

[10] *CompatibleOne*, [Online] Available: http://www.compatibleone.org [Accessed: August 29, 2014].

[11] *Open Grid Forum*, [Online] Available: http://www.ogf.org [Accessed: August 29, 2014].

[12] OCCI Working Group, *An Open Community Leading Cloud Standards*, [Online] Available: http://occi-wg.org [Accessed: August 29, 2014].

[13] R. Nyrén, A. Edmonds, A. Papaspyrou, and T. Metsch, *Open Cloud Computing Interface – Core*, OCCI-WG OGF, 2011. [Online] Available: http://ogf.org/documents/GFD.183.pdf [Accessed: August 29, 2014].

[14] T. Metsch and A. Edmonds, *Open Cloud Computing Interface – Infrastructure*, OCCI-WG OGF, 2011. [Online] Available: http://ogf.org/documents/GFD.184.pdf [Accessed: August 29, 2014].

[15] T. Metsch and A. Edmonds, *Open Cloud Computing Interface – RESTful HTTP Rendering*, OCCI-WG OGF, 2011. [Online] Available: http://ogf.org/documents/GFD.185.pdf [Accessed: August 29, 2014].

[16] The Apache Software Foundation, *Apache License, Version 2.0*, [Online] Available: http://www.apache.org/licenses/LICENSE-2.0.html [Accessed: August 29, 2014].

[17] R. Nyrén and F. Feldhaus, *Open Cloud Computing Interface – JSON Rendering*, OCCI-WG OGF, 2013. [Online] Available: https://redmine.ogf.org/attachments/98/json_rendering.pdf [Accessed: August 29, 2014].

[18] *NGiNX* [Online] Available: http://nginx.org [Accessed: August 29, 2014].

[19] Metsch, T.; Edmonds, A., *OCCI*, [Online] Available: https://wiki.openstack.org/wiki/Occi [Accessed: August 29, 2014].

[20] GRNET, *snf-occi's documentation*, [Online] Available: http://www.synnefo.org/docs/snf-occi/latest/ [Accessed: August 29, 2014].

[21] SixSq, *Slipstream*, [Online] Available: http://sixsq.com/products/slipstream.html [Accessed: August 29, 2014].

[22] IN2P3, *JSAGA*, [Online] Available: http://software.in2p3.fr/jsaga [Accessed: August 29, 2014].

[23] DIRAC Project, *The DIRAC Interware*, [Online] Available: http://diracgrid.org [Accessed: August 29, 2014].

[24] Barcelona Supercomputing Center, *COMP Superscalar*, [Online] Available: http://www.bsc.es/computer-sciences/grid-computing/comp-superscalar [Accessed: August 29, 2014].

[25] The Cloud4E Project, *Trusted Cloud Computing for Engineering*, [Online] Available: http://www.cloud4e.de [Accessed: August 29, 2014].