jOCCI – General-Purpose OCCI Client Library in Java

Michal Kimle, Boris Parák, Zdeněk Šustr
Cloud Standards
  OCCI – Introduction, principles, underlying ideas

jOCCI
  Motivation
  Overview

rOCCI & jOCCI side by side
Standards for *Infrastructure as a Service* cloud delivery model

- Relatively late in coming
- **CDMI** (Cloud Data Management Interface, SNIA, 2010)
  - Cloud storage
- **OCCI** (Open Cloud Computing Interface, OGF, 2011)
  - VM Management
- **CIMI** (Cloud Infrastructure Management Interface, DMTF, 2012)
  - VM Management
Standards for *Infrastructure as a Service* cloud delivery model

- Relatively late in coming
- **CDMI** (Cloud Data Management Interface, SNIA, 2010)
  - Cloud storage
- **OCCI** (Open Cloud Computing Interface, OGF, 2011)
  - VM Management
- **CIMI** (Cloud Infrastructure Management Interface, DMTF, 2012)
  - VM Management

Compare with Amazon WS – y. 2006
- Standard of choice for the EGI Federated Cloud
  - CIMI not yet released at design time
  - no discussion of OCCI vs. CIMI here
  - CDMI for object storage

- Text-based protocol and API focusing on cloud interoperability
  - Primarily for IaaS, extensible to other areas
  - Currently v. 1.1, published in three parts:
    - Core
    - Infrastructure
    - HTTP Rendering
  - v. 1.2 to be released for public comment on 23 March 2015
    - See insider info in the “Future” Section
OCCI Core Class Structure

Category
- Scheme: URI
- Term: string
- Title: string[0..1]
- Attributes: Set<string>

Kind
- entity_type: Entity

Action
- kind: mixed

Mixin
- mixins

Entity
- id: URI
- title: String[0..1]

Resource
- summary: String[0..1]

Link
- source
- target
- links

20 March 2015
ISGC '15, Taipei, Taiwan
EGI
www.egi.eu
**Example:** Rendering a simple compute resource, i.e., describing a virtual machine in OCCI:

```
Category: compute; scheme="http://schemas.ogf.org/occi/infrastructure#"; class="kind"
X-OCCH-Attribute: occi.core.id="ee13808d-7708-4341-a4ba-0e42e4818218"
X-OCCH-Attribute: occi.core.title="exampleVM"
X-OCCH-Attribute: occi.compute.cores=1
X-OCCH-Attribute: occi.compute.memory=1.7
X-OCCH-Attribute: occi.compute.architecture="x86"
X-OCCH-Attribute: occi.compute.speed=1
X-OCCH-Attribute: occi.compute.state="active"
```
Certain limitations in **text** rendering

- Inability to describe multiple resources in a single message
  - will be eventually overcome by JSON rendering specs
  - or XML
- Difficult parsing
- Size limitations when transported in HTTP headers
  \(\text{(text/occi)}\)
<table>
<thead>
<tr>
<th><strong>Client</strong> must be able to ...</th>
<th><strong>Server</strong> must be able to ...</th>
</tr>
</thead>
</table>
| receive, parse and understand **model** | render **model**:
  - local capabilities
  - mixins
  - resource categories |
| render **request**, validate against model and send | parse **request** and act on it |
| receive **response**:
  - location
  - rendered object
  - HTTP return code | return **response** |

**Table:** Simplified schema of a single-request workflow
jOCCI – What Led Here

Mid-2014

• Numerous infrastructures already using OCCI in their workflows...
• often even extending OCCI with custom specifications...
• but very limited choice of general-purpose clients
  • command-line client: rOCCI-cli
    • lets you wrap around CLI calls
  • client libraries: rOCCI-core + rOCCI-api
    • Ruby – native \(\Rightarrow\) OK
    • JRuby – not so good, but demand for Java support apparent
Mid-2014

- Numerous infrastructures already using OCCI in their workflows...
- Often even extending OCCI with custom specifications...
- But very limited choice of general-purpose clients
  - Command-line client: rOCCI-cli
    - Lets you wrap around CLI calls
  - Client libraries: rOCCI-core + rOCCI-api
    - Ruby – native $\Rightarrow$ OK
    - JRuby – not so good, but demand for Java support apparent

**September 2014: EGI** FedCloud funds development of an independent OCCI client library in Java: jOCCI.
- Taken up by CESNET – developers of the existing rOCCI framework
jOCCI v. 0.1.0 released on 3 March 2015

- 1st release comes in two parts
  - jOCCI-core
    - representation of OCCI Class structure
    - https://github.com/EGI-FCTF/jOCCI-core
  - jOCCI-api
    - transport routines
    - https://github.com/EGI-FCTF/jOCCI-api

- Libraries available from maven central
- More in following slides
Java-based implementation of OCCI:
- **classes** defined by OCCI Core and Infrastructure specifications, and methods for manipulating them
- methods for **rendering** plain-text or HTTP header representations of OCCI class instances
- methods for basic **parsing** of such renderings
- methods for **validation** of OCCI requests with respect to known models on client-side

... in other words...
- Java-based implementation of OCCI:
  - classes defined by OCCI Core and Infrastructure specifications,
  - methods for manipulating them
  - methods for rendering plain-text or HTTP header representations of OCCI class instances
  - methods for basic parsing of such renderings
  - methods for validation of OCCI requests with respect to known models on client-side

... in other words...

OCCI translated to Java

Think in OCCI

work with OCCI concepts hands-on in Java

Remember this?

jOCCI-core
• Java-based implementation of OCCI:
  • **classes** defined by OCCI Core and Infrastructure specifications, and methods for manipulating them
  • methods for **rendering** plain-text or HTTP header representations of OCCI class instances
  • methods for basic **parsing** of such renderings
  • methods for **validation** of OCCI requests with respect to known models on client-side
  
  ... in other words ... **OCCI translated to Java**
  • **Think** in OCCI
  • work with OCCI concepts hands-on in Java
• Transport functions for rendered OCCI queries
  • HTTP
  • Authentication
    • Basic
    • X.509 & VOMS
    • Keystone tokens
  • *Client* *(HTTPRequest)* class implementing basic resource management methods
    • list
    • describe
    • create
    • delete
    • trigger
• EntityBuilder class to construct OCCI class instances
  … while *immediately* verifying the new instance against model
Creating a Client instance:

```java
Client client = new HTTPClient(URI.create("https://remote.server.net"), new BasicAuthentication("username", "password"), MediaType.TEXT_OCCI, true);
```

Creating a compute resource (having skipped discovery for brevity):

```java
Model model = client.getModel();
EntityBuilder entityBuilder = new EntityBuilder(model);
Resource resource = entityBuilder.getResource("compute");
resource.addMixin(model.findMixin("debian7", "os_tpl"));
resource.addMixin(model.findMixin("small", "resource_tpl"));
resource.addAttribute(Compute.MEMORY_ATTRIBUTE_NAME, "2048");
URI location = client.create(resource);
```

Works with any cloud platform implementing OCCI!

- rOCCI (OpenNebula, AWS, MS Azure\textsuperscript{in progress}, openstack, synnefo, ...
Where does jOCCI fit into the world?

(Admittedly from the point of view of an EGI person)
OCCI – the Cloudscape Today

Clients

Native Java application
Native Ruby application
Application or script using command-line executables

Custom client app.

Native Ruby application

# Shell script

rOCCI-cli
rOCCI-api
rOCCI-core

http
occi

Server

mod_ssl
Apache
Phusion Passenger

rOCCI-server

Backends

OpenNebula
MS Azure

http
occi

http
occi

http
occi

http
occi

http
occi

http
occi

http
occi

http
occi

Server

Independent OCCI-compliant server

NEW! NEW! NEW! NEW!
OCCI – the Cloudscape Today

**Clients**

- Native Ruby application
- Application or script using command-line executables
- Custom client app.

**Server**

- Independent OCCI-compliant server
- mod_ssl
- Apache
- Phusion Passenger
- Backends
- OpenNebula
- MS Azure
- Amazon Web Services

**NEW!**

- Native Ruby application
- Application or script using command-line executables
- Custom client app.

**Clients**

- jOCCI-core
- jOCCI-api
- # Shell script
- rOCCI-cli
- rOCCI-api
- rOCCI-core

**Server**

- Independent OCCI-compliant server
- mod_ssl
- Apache
- Phusion Passenger
- Backends
- OpenNebula
- MS Azure
- Amazon Web Services
OCCI – the Cloudscape Today

Clients

- Custom client app.
  - Native Java application
    - jOCCI-api
    - jOCCI-core
  - Application or script using command-line executables
    - # Shell script
      - rOCCI-cli
      - rOCCI-api
      - rOCCI-core
  - Native Ruby application
    - rOCCI-api
    - rOCCI-core

Server

- Server
  - mod_ssl
  - Apache
  - Phusion Passenger
  - rOCCI-server
  - Backends
  - OpenNebula
  - MS Azure
  - Amazon Web Services

Server

- Independent OCCI-compliant server

NEW! NEW! NEW! NEW!

20 March 2015 ISGC ’15, Taipei, Taiwan

EGI www.egi.eu
Multiple advantages of separation from rOCCI:

- multiple implementation attempts are crucial to the development of open standards
- getting a new take on an old problem
- not building on old bugs and design mistakes
- getting an opportunity to test real-world interoperability across different implementations
  - one solution testing the other
- a chance to apply language-specific design patterns, optimizations and use native project maintenance & distribution mechanisms
- no need for emulation, cross-compilation or embedded runtime environments for other languages
1.2
No, seriously:

- OCCI 1.2 is the greatest development currently expected
  - Proper Attribute definition in the standard
  - New resource state (error)
  - JSON rendering – now preferred
  - Pagination syntax & semantics defined by the standard
  - New extensions for billing, SLAs and monitoring
  - Separation of rendering and protocol specs

- Missing from 1.2, i.e., plans beyond:
  - XML rendering
    - … but JSON is a 100-% capable alternative
  - Filtering mechanism redesign
Aside of OCCI 1.2:

- Interoperability tests
  - Cloud Plugfests
  - EGI FedCloud testsuites
- Continued development of rOCCI backends
  - MS Azure – in progress, pole in the spokes
  - AWS – already out but you may have missed it
— THAT IS ALL —

Questions?