OCCI rendering and resource interpretation

Boris Parák, CESNET

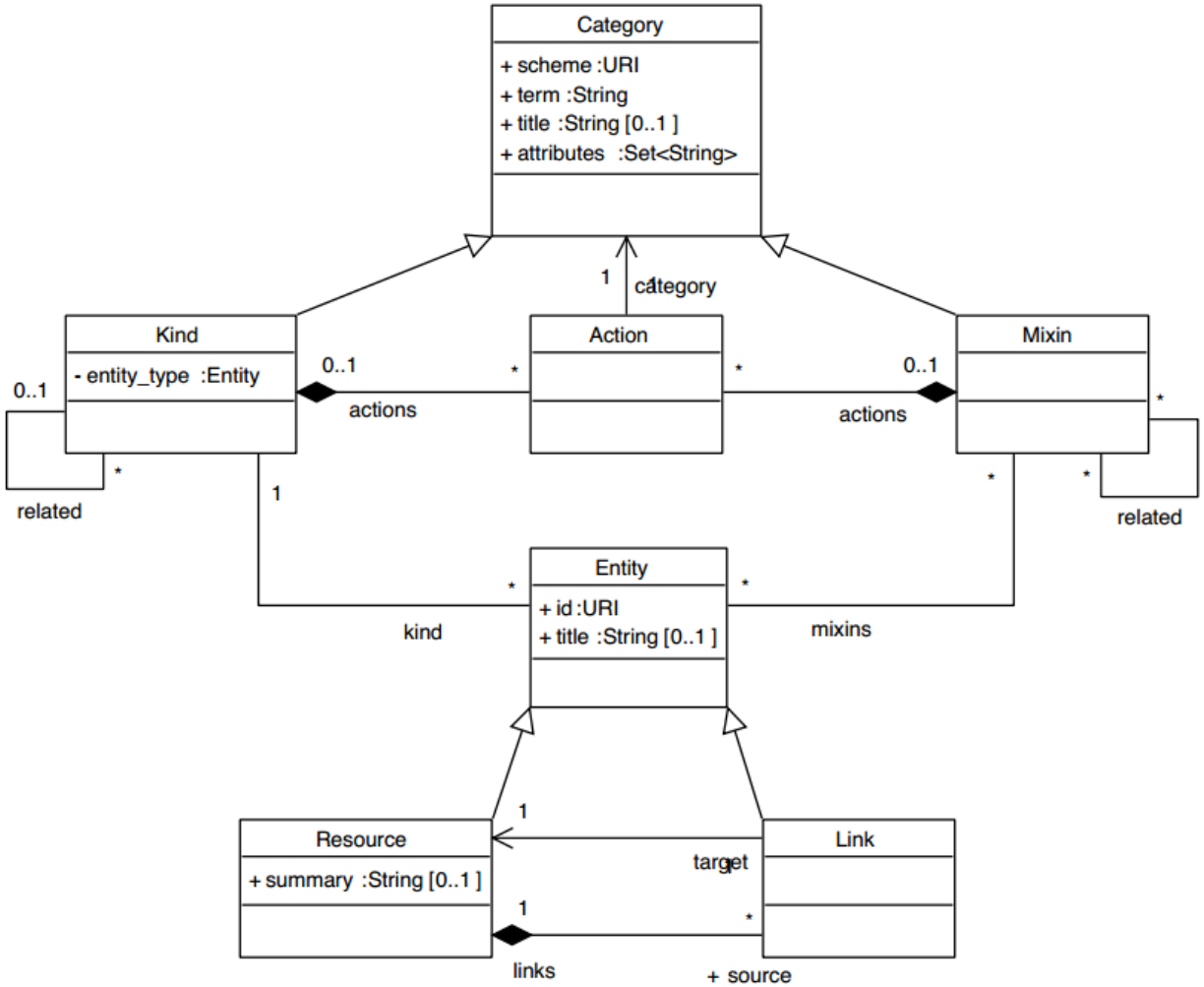
Overview

- Introduction
- Platforms & Implementations
- Rendering
- Semantics
  - Resources
  - Mix-ins

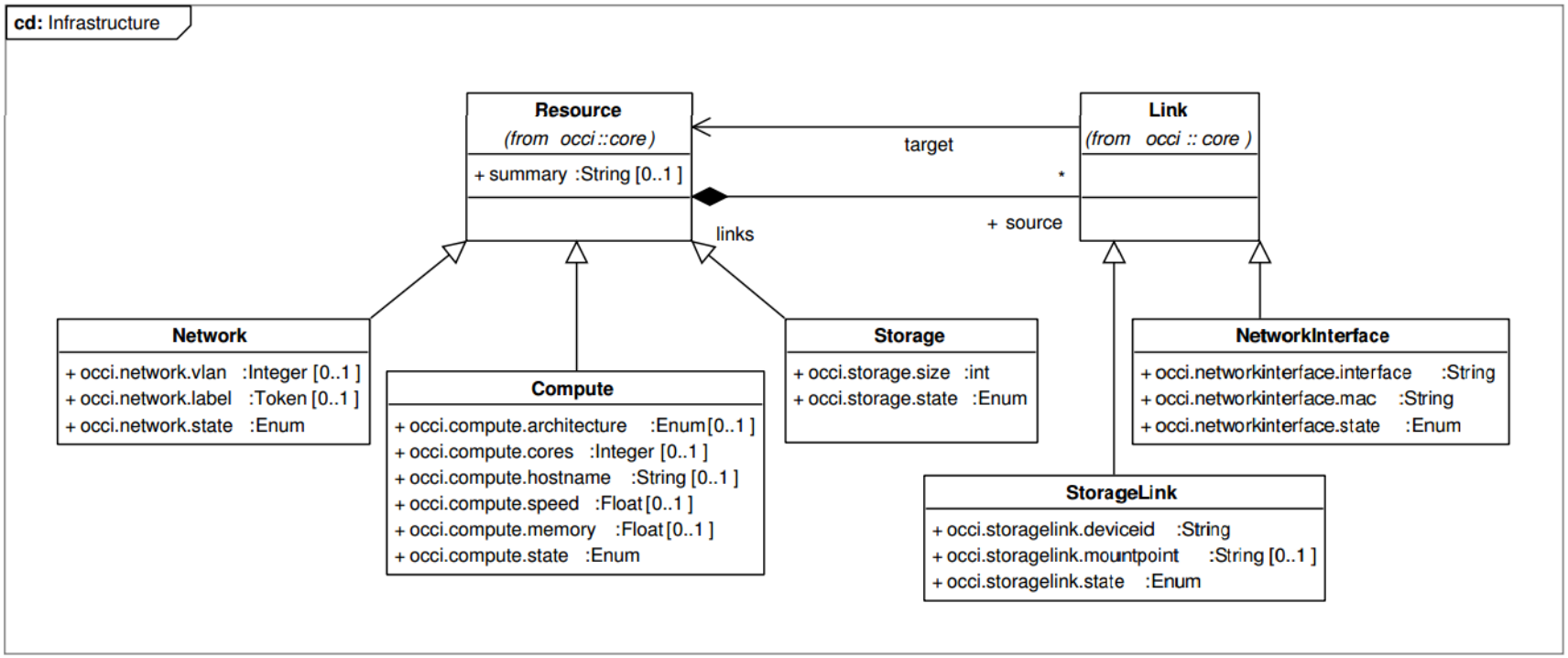
Introduction – OCCI

- OCCI $\rightarrow$ Open Cloud Computing Interface
- text-based protocol and API focusing on interoperability in the cloud
- originally designed for IaaS clouds, but is extensible
- works with resources, links and mixins

Introduction – OCCI Core



Introduction – OCCI Infra.



Platforms & Implementations

- OpenNebula, OpenStack, Synnefo, WNoDeS, ...
- each platform uses its own OCCI implementation
- issues with interpretation, rendering and semantics of some elements
- the following focuses on rOCCI (w/ OpenNebula) and OCCI-OS (w/ OpenStack)

Rendering – Formats I.

- **text/occi**
  - rendering purely into HTTP headers
  - using *Category*, *Location* and $X - OCCI - *$
  - requires a specialized parser

- **text/plain**
  - the same rendering syntax as **text/occi**
  - using HTTP body
  - requires a specialized parser

Rendering – Formats II.

- **application/occi+json**
  - preferred, rendered as pure JSON
  - easily parsable, with generic JSON parsers
  - standard not yet officially released

- **text/uri-list**
  - alternative for the *Location* header
  - multiple URIs rendered in the HTTP body

Rendering

- Problems with rendering formats:
  1. headers grow exponentially in production
  2. not all HTTP clients and proxies work properly with headers
  3. JSON rendering is still evolving and changing

- Problems with compliance:
  1. implementations breaking rendering restrictions, mostly issues with *Category* terms containing forbidden characters

Semantics – Resources

- problem with storage resources
- in rOCCI (w/ OpenNebula):
  – every attachable disk-like element is considered a "storage"
  → instances can be launched directly by specifying *storage* and *network* resource
- in OCCI-OS (w/ OpenStack):
  – images are considered a separate entity from ordinary block devices or snapshots
  → instances have to be launched using a template

Semantics – Mix-ins I.

- mix-ins are used to extend existing resources
- as other categories, they are identified by a *term* and the *title* attribute is optional
- only a few are mentioned in the specification, the rest is provider/implementation specific

- **os_tpl**
  – virtual machine templates
- **resource_tpl**
  – resource templates

Semantics – Mix-ins II.

We need to:

1. compile widely used mix-ins
2. push for standardization across implementations
3. first attempt $\rightarrow$
   `https://github.com/occi/occi-extensions`

References

What to read if you want to know more?

- `http://occi-wg.org`
- `https://github.com/gwdg/rOCCI-cli`
- `https://github.com/gwdg/rOCCI-server`
- `https://github.com/tmetsch/occi-os`

Do you have any questions?

- ask directly at parak@cesnet.cz
- ask in the mailing lists rocci@gwdg.de or inspire-mp-rocci@mailman.egi.eu

Appendix A

```
$ occi --auth basic --username a --password b \
       --action create --resource compute \
       --mixin os_tpl#demo --mixin resource_tpl#small \
       --attributes title='rOCCI_VM' \
       --context public_key='file:///tmp/id_rsa.pub'

$ occi --auth basic --username a --password b \
       --action create --resource compute \
       --link /storage/<id> --link /network/<id> \
       --attributes title='rOCCI_VM' \
       --context public_key='file:///tmp/id_rsa.pub'
```

Appendix B

```
Category: omserver;
          scheme="https://.../occi/infrastructure/os_tpl#";
          class="mixin";
          title="OMServer";
          rel="http://.../occi/infrastructure#os_tpl";
          location="/mixins/omserver/"
```

Figure: Example – OCCI Category