

OpenNaaS Tutorial

CYCLONE



S.E.G.

04/05/2015

Agenda

- Introduction to OpenNaaS
 - Key concepts
 - Main features
 - Interaction paradigms
- OpenNaaS for developers
 - Technological framework
 - Licencing
- Lab session
 - Project source modules distribution
 - Example code review
 - Extending the example on your own
- OpenNaaS in the real world
 - Managing an OpenDaylight controller
- Q&A



OpenNaaS

Key Concepts



S.E.G.

04/05/2015

Resource

Physical or virtual network component manageable by OpenNaaS capabilities.

- Root Resources
 - Router, Switch, Network, CPE, etc.
- Resources
 - Port, Link, Slice, etc.

Service

Software component providing the ability of configuring or managing a resource (or a set of them).

Capability

Set of services bound to a specific resource.

- Management Capability: Provides new resources to the system.
 - RootResourceManagement, PortManagement, LinkManagement, etc.
- Regular Capability: Provides services to configure the resource it's bound to.
 - LinkAdministration: Source port, destination port...
 - SliceAdministration: Slice units, slice cubes...

Capability example

```
public interface IRootResourceProvider extends ICapability {
```

```
    List<IRootResource> getRootResources();
```

```
    IRootResource getRootResource(String id) throws ResourceNotFoundException;
```

```
}
```

Service

Service

Resource

Defining a resource: Resource Descriptor

- Contains technical and specific information of the Root Resource.
- Specification
 - Type
 - Router, Switch, Network, etc.
 - Model
 - JunOS, Cisco, ...
 - Version
 - 10.04, 1.0, Helium, Icehouse, etc.,
- Endpoints
 - http, https, netconf, etc.
- Credentials
 - username/password, certificates, ...

Example

```
...  
Specification specification = new Specification("network", "odl", "helium");  
Endpoint endpoint = new Endpoint("http://10.10.0.4:8080");  
UsernamePasswordCredentials credentials = new UsernamePasswordCredentials("admin", "admin");  
  
RootResourceDescriptor rd = RootResourceDescriptor.create(specification, Arrays.asList(endpoint),  
credentials);  
  
...
```

OpenNaaS

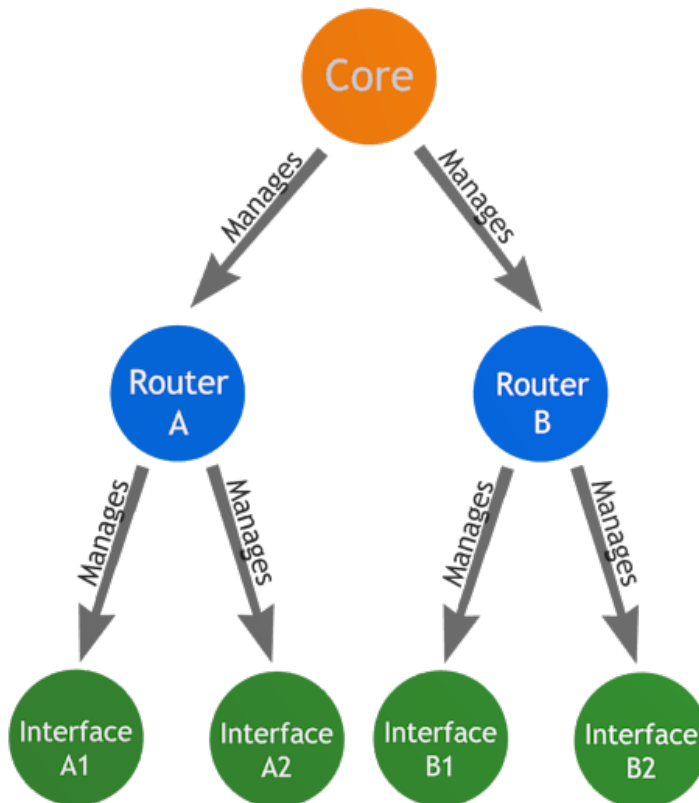
Main features



S.E.G.

04/05/2015

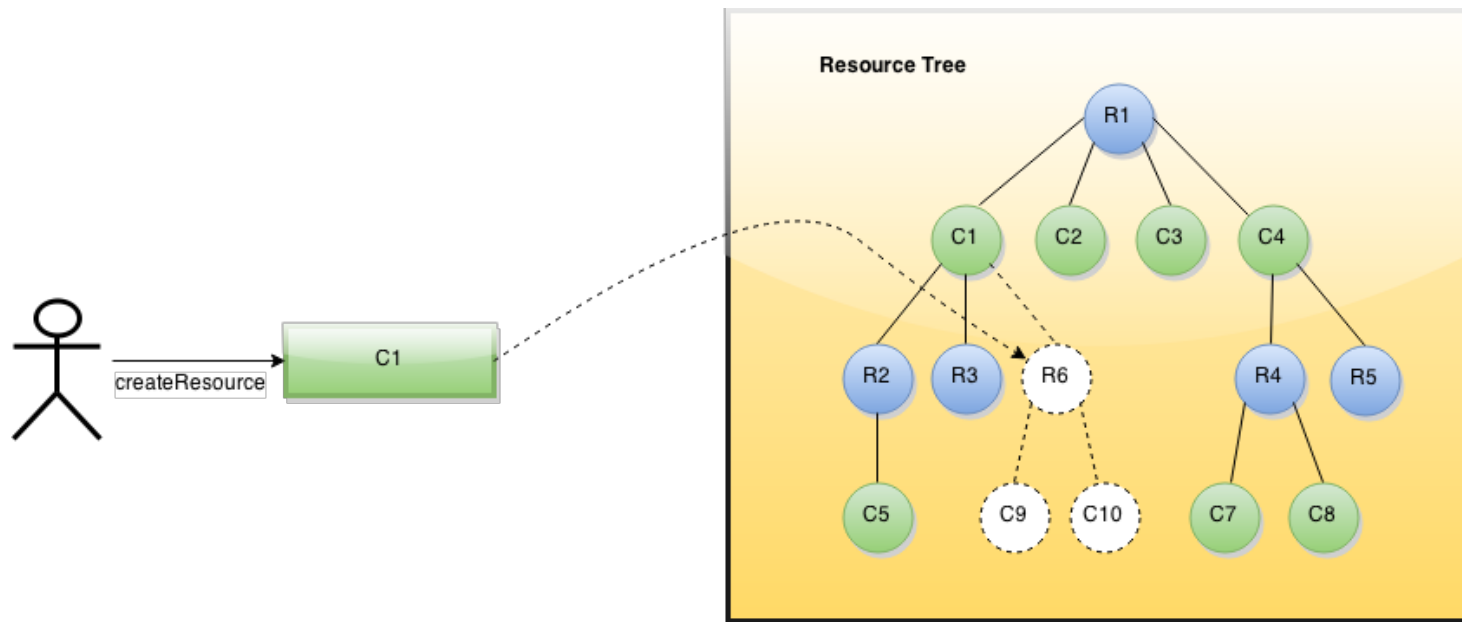
Dynamic resources and services: Resource Tree



- Internal data structure reflecting the available Resources and their services.
- Define and manages the relations between parent and child resources using “resource management” capabilities.
- Used by various modules, e.g. the REST API, to publish services endpoints with HATEOAS architecture.

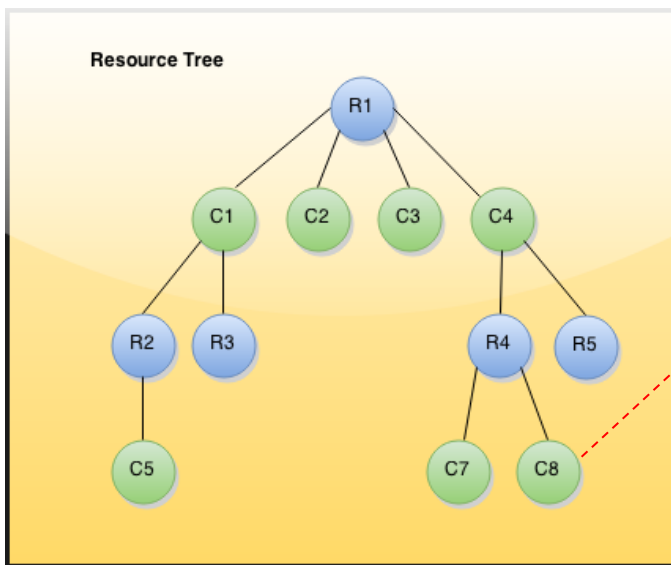
Binding Management

- Decides which capabilities are bound to a resource.
- Monitors resources and capabilities lifecycle.
- Interacts with Resource Tree once resources/capabilities are added/removed.



REST API

- Services automatically published.
- Built from Resource Tree.



<http://localhost:9000/mqnaas/R1/C4/R4/C8>

Example:

<http://localhost:9000/mqnaas/mqnaas-core/IRootResourceProvider/router1/IIPortManagement>

Dependency Injection

- Capabilities can depend on services of other capabilities.
- Dependent services are accessed through the capabilities they belong to.
- Dependent capabilities are automatically injected.

```
public ODLNetworkTopology implements INetworkTopology {  
  
    @DependingOn  
    ILinkManagement linkManagement;  
  
    ...  
}
```



Available Capabilities

Interaction Paradigms

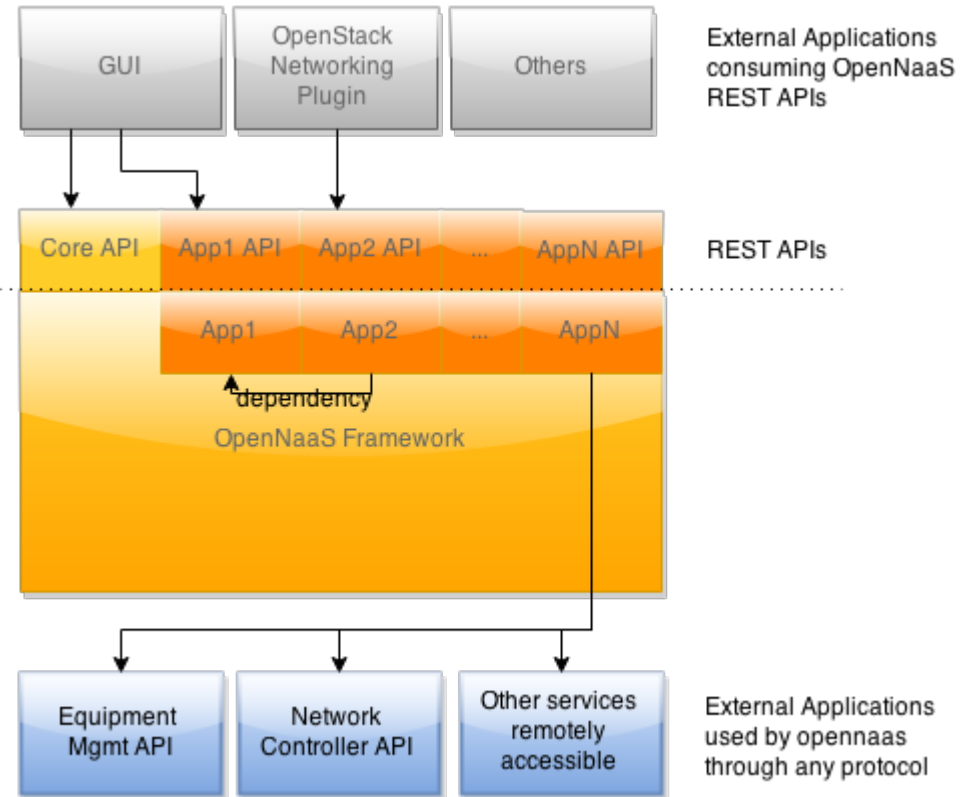


S.E.G

04/05/2015

Interaction paradigms

1. Consume an existing OpenNaaS app remotely
2. Create your app inside OpenNaaS
 - a. Use other OpenNaaS apps
 - b. Use external apps remotely through a client.



Accessing remote devices

Applications may make use of any protocol to interact with external devices or services. Commonly, a protocol client is used, specially when protocols are session oriented.

Accessing OpenNaaS apps

Remote access to OpenNaaS apps

Accessible through the remote REST API.

Invoking a method results in a service being executed.

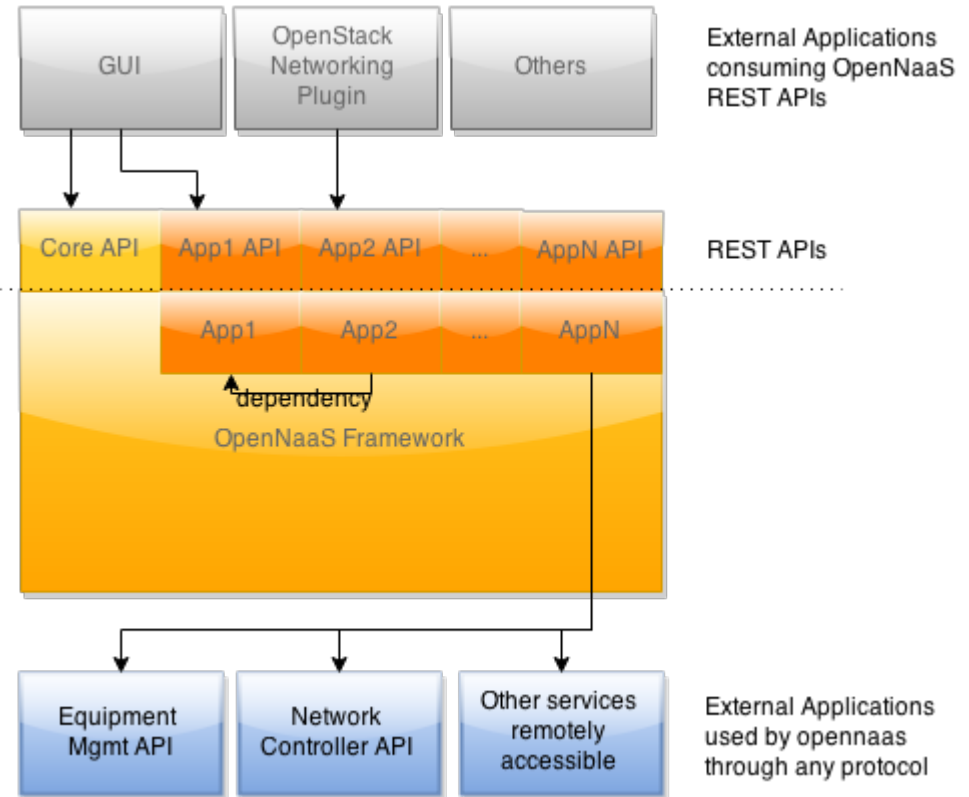
Static environment. Needs polling for being up to date.

Local access to OpenNaaS apps

Resolved through dependency injection mechanism, or by using OpenNaaS ServiceProvider.

Calling a method results in a service being executed.

Dynamic environment. Apps may come and go whenever resources are created.



OpenNaaS for developers



S.E.G

04/05/2015

Base Technologies

- Java 7 (Oracle)
- Apache Maven 3 - Software project management tool. <https://maven.apache.org/>
- OSGi 4.3 - Modular system implementing a dynamic component model, where modules are divided in bundles offering/consuming services to/from the rest of modules. <http://www.osgi.org/Technology/HomePage>

Platform

- Apache Karaf 3.0.x. Karaf is a small OSGi based runtime which provides a lightweight container onto which various components and applications can be deployed. <https://karaf.apache.org/>
 - Provides UI (shell)
 - Support for OSGi 4.3

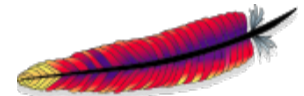


Web services

- Apache CXF 2.7.14 - Web services framework supporting multiple protocols (SOAP, XML/HTTP, RESTful HTTP, or CORBA) over a variety of transports (HTTP, JMS or JBI). <https://cxf.apache.org/>

Licensing

- OpenNaaS core
 - LGPL v3 - <https://www.gnu.org/licenses/lgpl.html>
- OpenNaaS extensions
 - Contributor can choose any license
 - i2cat provided extensions ASF v2.0 - <https://www.apache.org/licenses/LICENSE-2.0.html>



*Developing
your first extension*



S.E.G

04/05/2015

Tutorial environment: VM Content

- Development tools
 - Oracle Java 7
 - Git
 - Maven 3
 - Eclipse Luna

- MQNaaS source code
 - `$HOME/soft/mqnaas`

- Apache Karaf 3.0.3
 - `$HOME/soft/apache-karaf-3.0.3/`

- Mininet

- Opendaylight Helium

HelloWorld Sample

- Package distribution
- Bundle architecture
 - Definition of bundle.
 - Pom file: packaging, dependencies, plugins, etc.
 - Karaf feature.
- Capability Interface
 - Java Interface extending ICapability.
 - Declared methods compose the capability services.
- Capability Implementation
 - “IsSupporting” method.
 - “Activate” and “Deactivate” methods.
 - Services implementation.
 - Dependency and resource injection

Detailed info: <https://github.com/dana-i2cat/mqnaas/tree/master/examples/hello-world>

HelloWorld Sample (2)

- Run the app:
 - Run OpenNaaS: Just click the icon
 - Follow the instructions in <https://github.com/dana-i2cat/mqnaas/tree/master/examples/hello-world#building>
 - Install feature:
 - Create Hello-world resource.
 - Launch request to hello world service

Hands on exercise:

Extending HelloWorld Sample

Goal:

Create a capability depending on HelloWorld, and available through the API.

Tips:

- Create a new Capability
- Make this capability depend on the HelloWorld one.
- Test it through the API :)



Real sample:

OpenDaylight Network



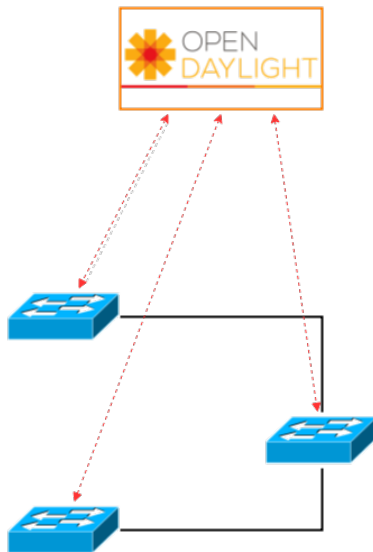
S.E.G

04/05/2015

OpenDaylight extension

Physical view

OpenDaylight instance:
Openflow Controller of the
physical switches.

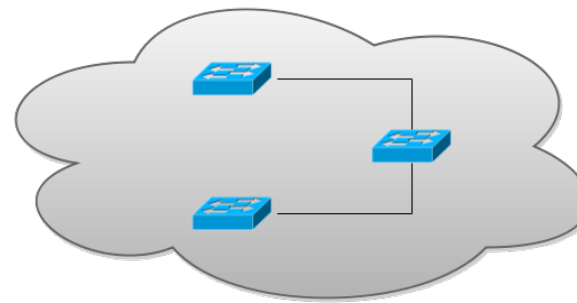


Network of OpenFlow switches
in tree topology.

Logical view



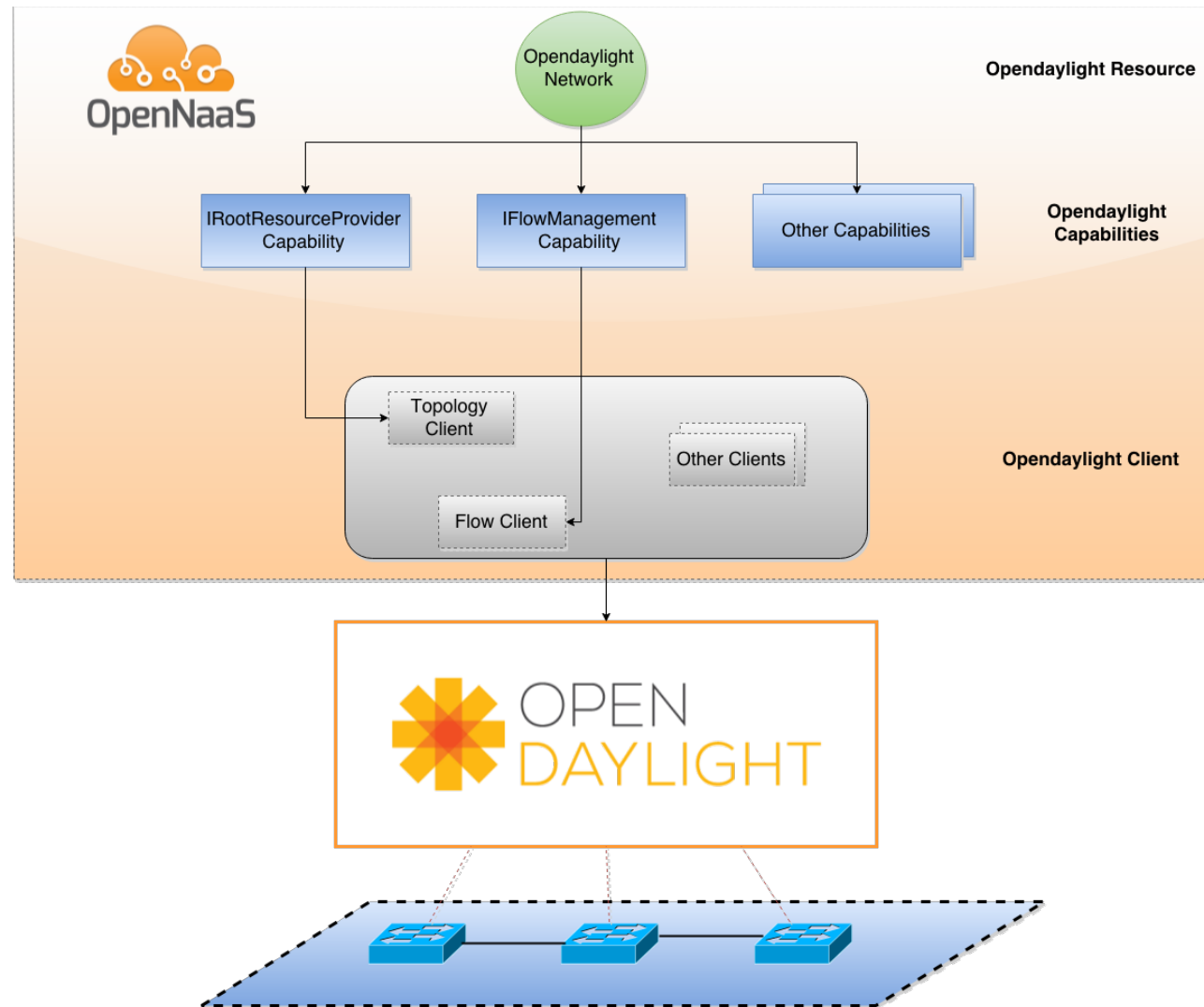
OpenNaaS instance



Virtual representation of
OpenDaylight as a Network
resource managing three
switches.

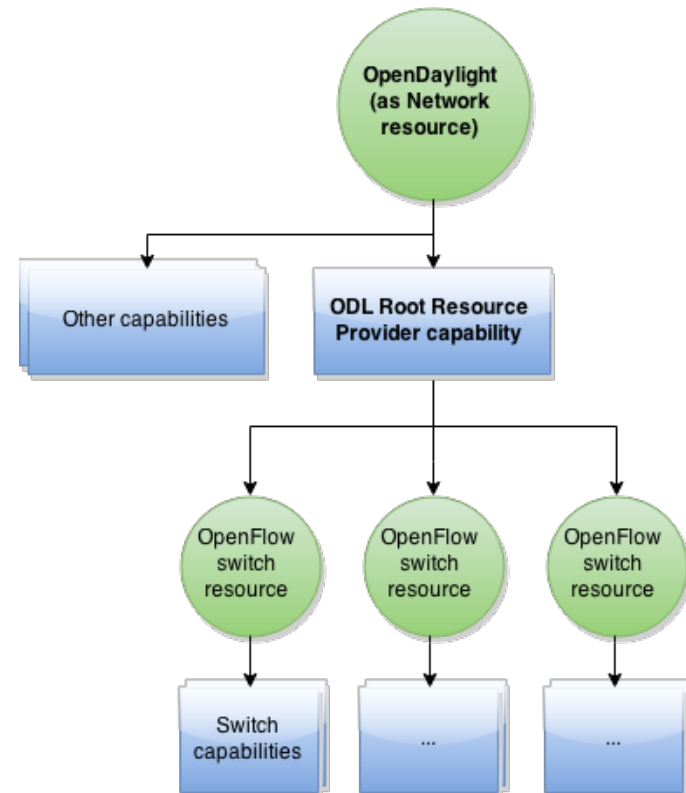
OpenDaylight extension: developer view

- NetworkResource:
 - Model =.opendaylight,
 - EP = controller URL
- OpenDaylight specific implementations for network capabilities:
 - Use client to expose services generically defined in their interface (HAL)
- OpenDaylight client: Communicates with a real instance of OpenDaylight controller



OpenDaylight extension: user view

- OpenDaylight represented as Network Resource.
- Underlying resources automatically initialized by IRootResourceProvider implementation for OpenDaylight networks.
- Resources model automatically initialized by same capability.
- Openflow specific capabilities:
 - FlowManagement



GUI

<http://84.88.41.171:8080/#!/network>

Questions & Answers




S.E.G

04/05/2015

Open question:

- How to integrate OpenNaaS into CYCLONE overall system?

Cyclone

The OpenNaaS logo, which consists of an orange cloud shape with four white circles and lines representing nodes or connections.

OpenNaaS

