OPEN DAYLIGHT

YANG Modeling: The Good, The Bad, and The Ugly

Colin Dixon

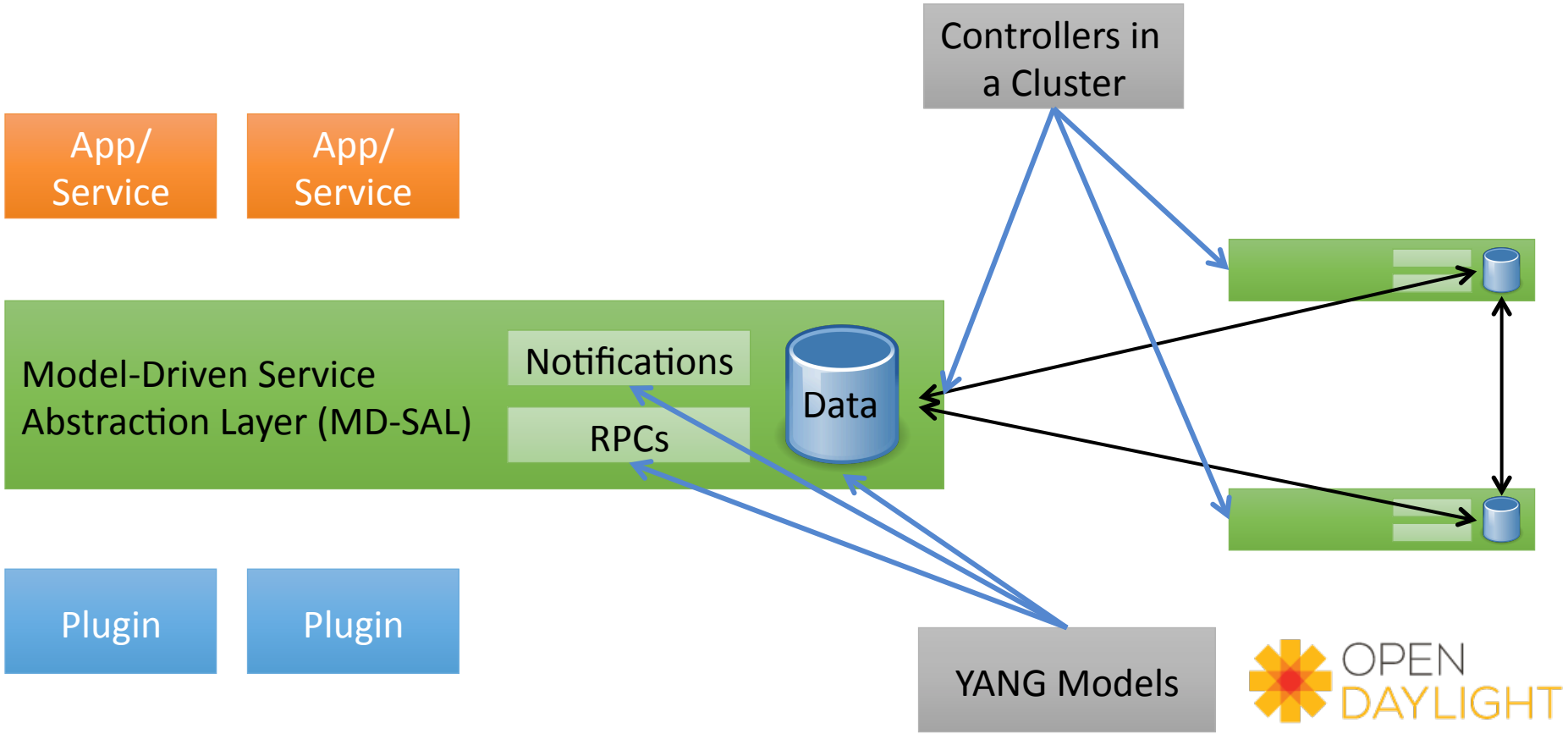
Technical Steering Committee Chair, OpenDaylight
Principal Engineer, Brocade

Talk Outline

- *Really* fast intro to the OpenDaylight Architecture
- What is YANG?

- **The Good:** Things that make life better
- **The Bad:** Things that are frustrating
- **The Ugly:** Things that need to care to get right

Core Architecture



What is YANG?

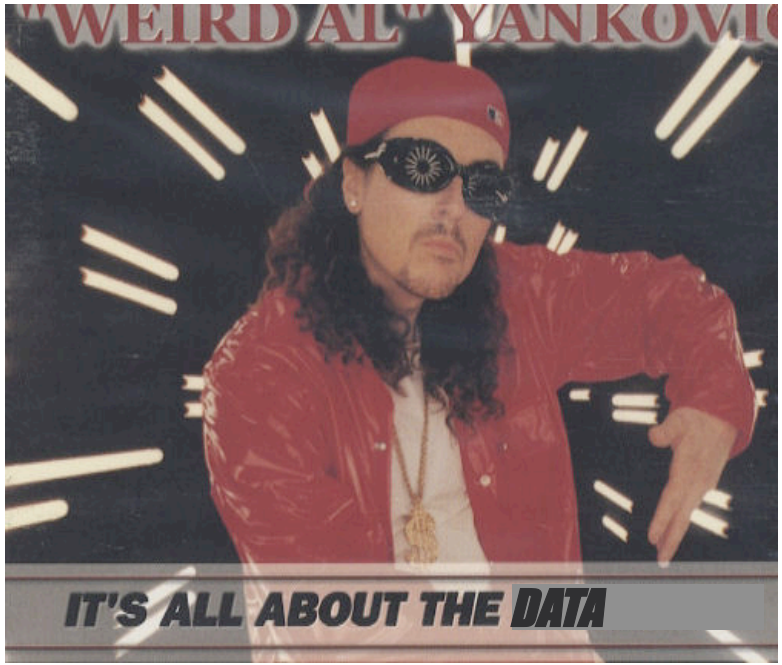
- Data modeling language for NETCONF
  - RFC 6020
- Great, what is NETCONF?
  - Think of it as an SNMP replacement with nice features
  - YANG models ~= SNMP MIBs
- OK, fine, but what *is* YANG?

OPEN DAYLIGHT

What is YANG?

- Three core abstractions
  - Data
  - RPCs (just data in and data out)
  - Notifications (just data out)

- So, it's really all about the data

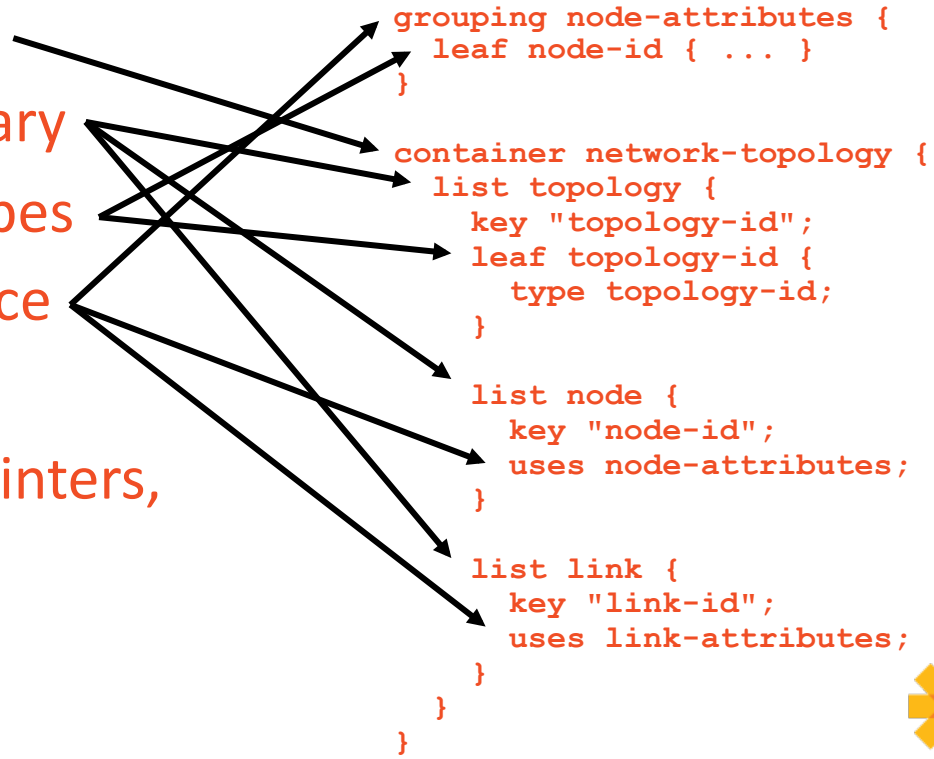


OPEN DAYLIGHT

What does YANG data look like

- container ~= struct
- list ~= map/dictionary
- leaf ~= primitive types
- grouping ~= interface

- Others: typedef, pointers, constraints, etc.



```
grouping node-attributes {
  leaf node-id { ... }
}

container network-topology {
  list topology {
    key "topology-id";
    leaf topology-id {
      type topology-id;
    }

    list node {
      key "node-id";
      uses node-attributes;
    }

    list link {
      key "link-id";
      uses link-attributes;
    }
  }
}
```

OPEN DAYLIGHT

The Good

YANG in OpenDaylight: The Good

- We have tons of cool tooling to make working with YANG easy
  - Auto-generated code
    - REST APIs
    - Java “Bindings”
    - Web User Interfaces
  - Tools to convert modeled objects to/from JSON and XML
  - Extensibility

- Essentially, all the benefits of model-driven programming

Great, let's play with it

- Idea: model to populate DMAC-based forwarding of a switch
  - Take a switch DPID, port, and MAC address
  - Produce an OpenFlow rule on that switch to forward traffic to that MAC out the given port

```
module multilevelmodeling {
  namespace "urn:opendaylight:mlm";
  prefix mlm;

  import ietf-yang-types {prefix yang;}
  import openflow-types {prefix oft;}

  revision 2015-06-12;

  grouping dmac-entry-attributes {
    leaf dmac { type yang:mac-address; }
    leaf port { type oft:port-number; }
    leaf switch-dpid { type uint64; }
  }

  container dmac-table {
    list dmac-entry {
      uses dmac-entry-attributes;
    }
  }
}
```

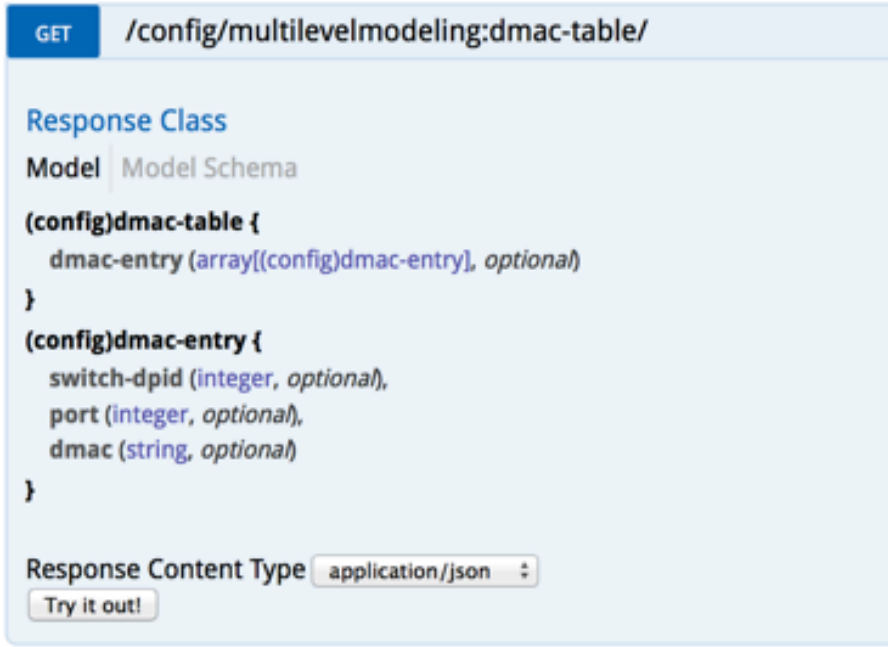
OPEN DAYLIGHT

Demo

OPEN DAYLIGHT

Auto-generated APIs and UI

- Auto-generated REST APIs
  - Just nice, no futzing
- Auto-generated UI for models
  - Takes advantage of the fact that humans can interpret strings
  - e.g., if you see “IP address” next to a dotted quad…



OPEN DAYLIGHT

Extensibility

- You can augment an existing model with new information
  - Here, we add two new fields
    - source MAC
    - source port
  - This can be done after the fact
  - It doesn't require recompilation

```
module multilevelaug {
  namespace "urn:opendaylight:mlm:aug";
  prefix mlmaug;

  import ietf-yang-types {prefix yang;}
  import openflow-types {prefix oft;}
  import multilevelmodeling {prefix mlm;}
  import yang-ext {prefix "ext";}

  revision 2015-06-14;

  grouping dmac-source-entry-attributes {
    leaf smac { type yang:mac-address; }
    leaf sport { type oft:port-number; }
  }

  augment "/mlm:dmac-table/mlm:dmac-entry" {
    ext:augment-identifier "smac-entry";
    uses dmac-source-entry-attributes;
  }
}
```

Demo

OPEN DAYLIGHT

The Bad

OPEN DAYLIGHT

The Bad (I'd like to see fixed)

- Major limitation
  - Missing recursive self inclusion
- ODL-specific issues
  - All data is "rooted" in the tree, makes it annoying to have "free-floating" data even though it would be hugely useful
- Annoyances
  - Can't tell from encoded data which "choice" was taken
  - Typing isn't perfect, e.g., can't constrain instance identifiers
  - Can't have true maps
  - Can't have keyless lists
  - Auto-populated keys in lists are missing

No Recursive Self Inclusion

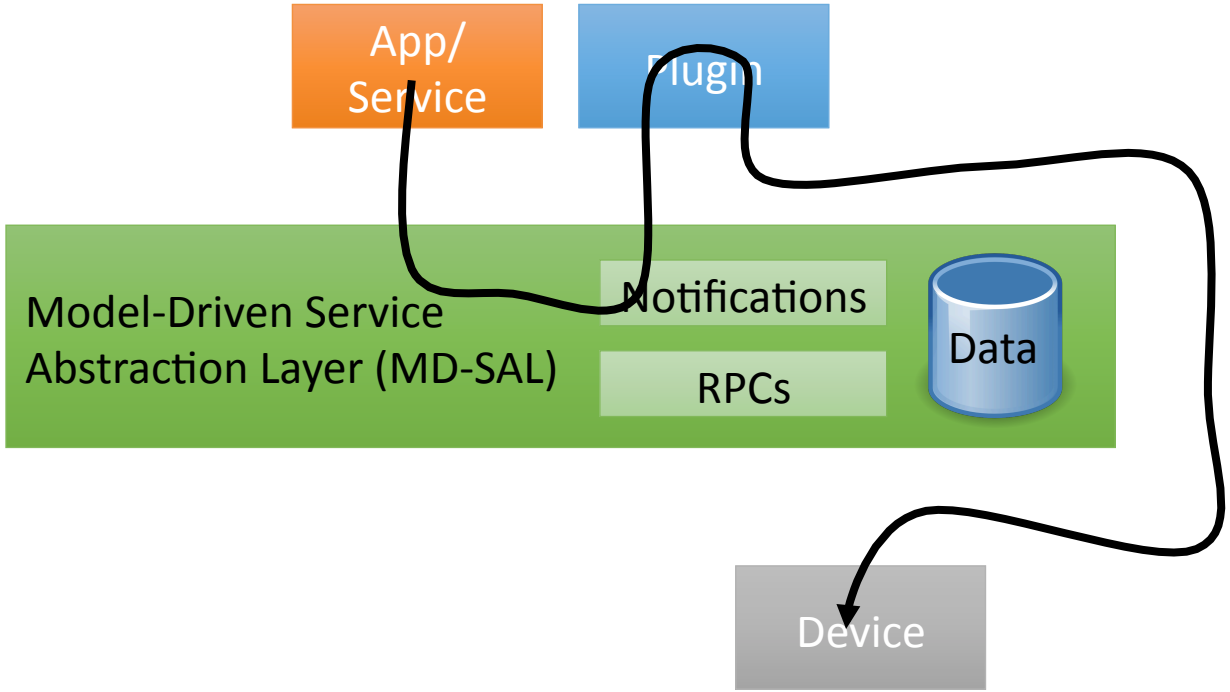
- Grouping can’t contain itself
- Thus, can only finitely recurse
  - even then awkwardly

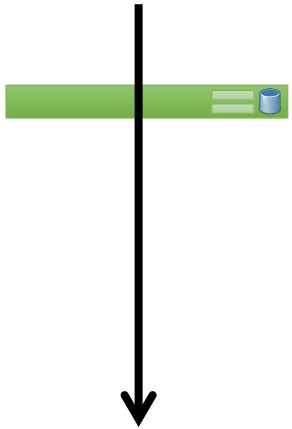
```
// WHAT WE WANT:
grouping router-attributes {
  // ...
  list VRF {
    uses router-attributes; //illegal!
  }
}

// WHAT WE CAN DO:
grouping router-attributes{
  //...
}
container router{
  uses router-attributes;
  list VRF{
    uses router-attributes; //only one level
  }
}
```

OPEN DAYLIGHT

The Ugly

Core Architecture (Logical)

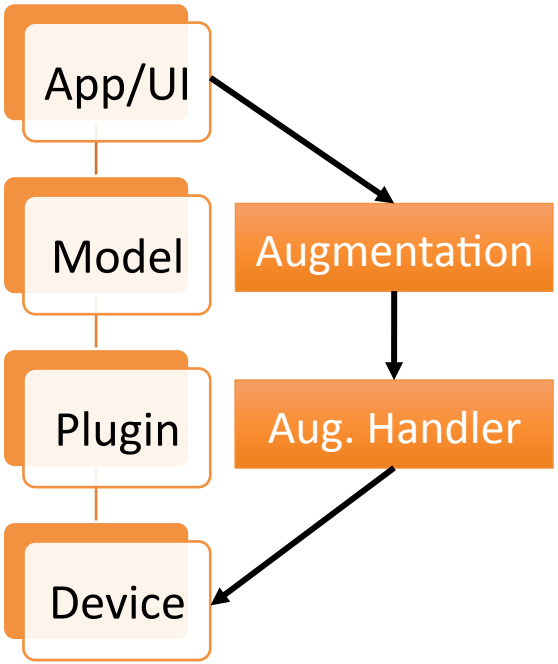




OPEN DAYLIGHT

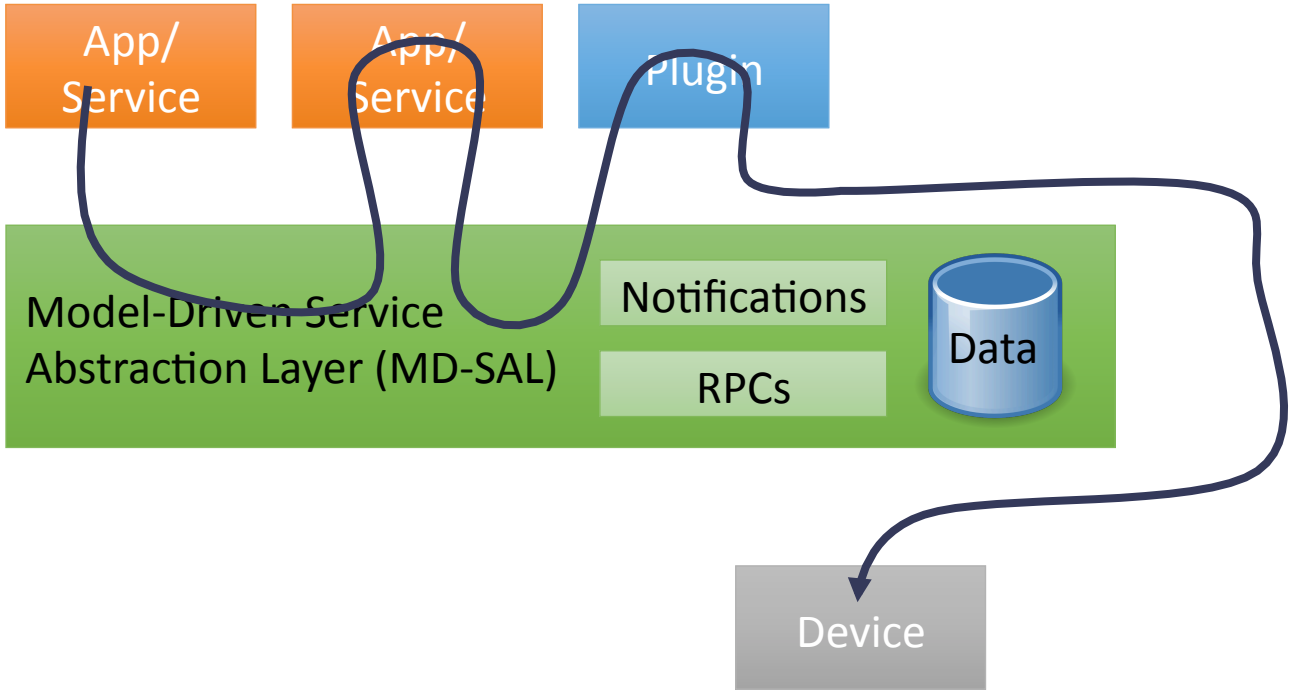
Simple Modeling + Augmentations

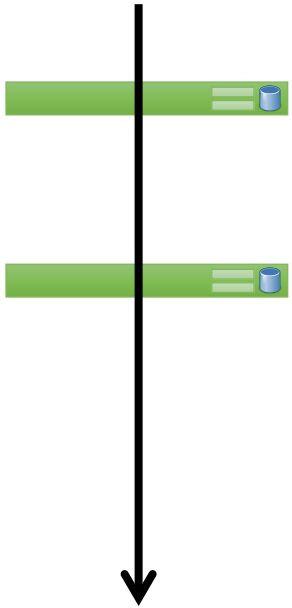
- We have one model
  - Plugin translates it to a device
  - App/UI consumes/presents it
- Augmentation
  - Model-based UI is automatic
  - Apps can ignore the new part
  - Need some “augmentation handler” in the plugin
    - Model augmentation => device operations



OPEN DAYLIGHT

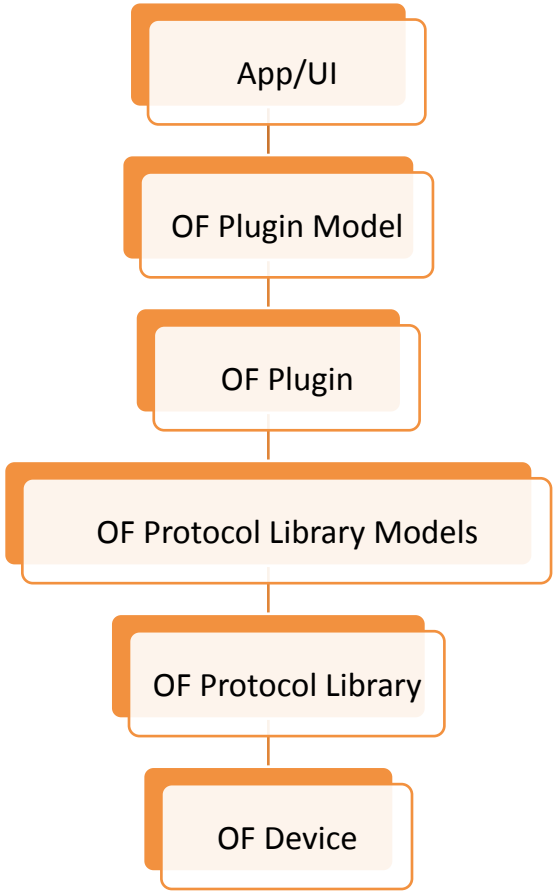
Core Architecture (Logical)





OPEN DAYLIGHT

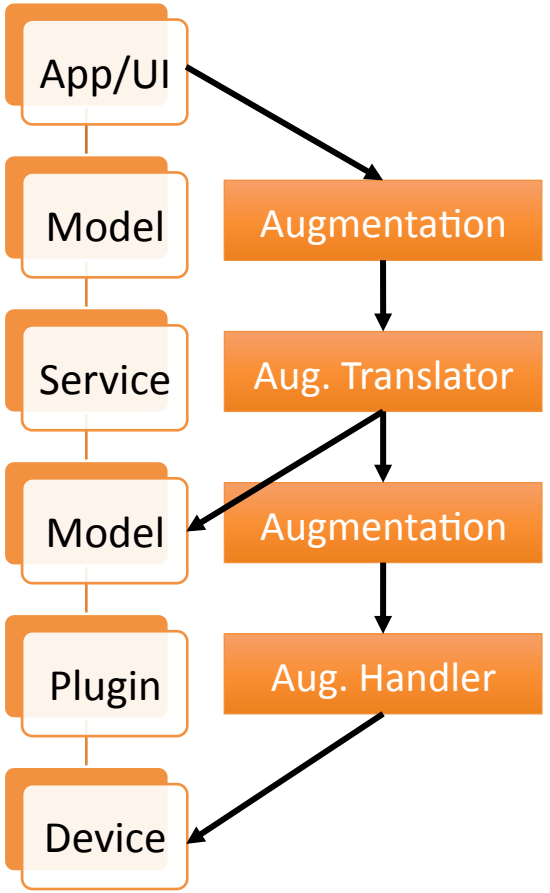
Layered Modeling Exists



OPEN DAYLIGHT

Layered Modeling is Hard

- Multiple models in a stack
  - Augmentations need translators
  - Translate into another model
  - Translate into an augmentation of another model
  - Might also need another “augmentation handler” to control the device



OPEN DAYLIGHT

Conclusions

- OpenDaylight is based on YANG modeling
- YANG models are mostly good
  - Auto-generation of code, APIs, UIs
  - Extensibility
  - Have some rough edges
- When writing a model provider…
  - …make sure to allow for augmentation handlers
  - …make sure to allow for augmentation translators

OPEN DAYLIGHT