IEEE YANG
Some observations

Don Fedyk LabN Consulting L.L.C.

dfedyk@labn.net

Background

- During the last 6 months I've been looking at YANG and following YANGsters discussions.
- As a newcomer one comment I made was it seemed like IEEE has a more complicated YANG files than some others that are public.
- The other question I had was about automated programming and how much detail we put in a YANG model
- Here is some more input on these two points – perhaps not complete but a couple of insights Yangsters might consider.

Many good reasons do we do YANG

- Replace SNMP
- Give management objects standard names
- Give the objects a standard type and syntax
- Limit the values of an object to valid values, create defaults
- Re-Use standard other Models definitions
- Create dependencies and hierarchies based on component models
- Provide a Model that can be Validated and “Compiled”
- Publish in public

Comparison of YANG models

IEEE

- Build models to cover the full superset of 802.1 functions
- Follow IEEE bridge component models
- Use YANG conditionals (when .. must) etc. to enable functions for permutations of the components
- Reference IEEE Specifications

Some others:

- Use a leaner style (Based on MACsec/VLAN Models I have observed)
  - Smaller descriptions
  - No or minimal references
- Make heavy use of groupings
- Other – there may be a bit more here

Auto programming and initialization

Discussions around YANG:

- Originally a discussion about defaults
  - One argument is initialization code is outside of YANG so don’t worry about YANG defaults
  - On the other hand YANG allows defaults and ranges why not use them?
- This led to a discussion about how simple should YANG models be.
- This led to a discussion of being able to compile YANG so why not put in as much detail as possible.
- Where is YANG compiled? – next charts

YANG compiling to code

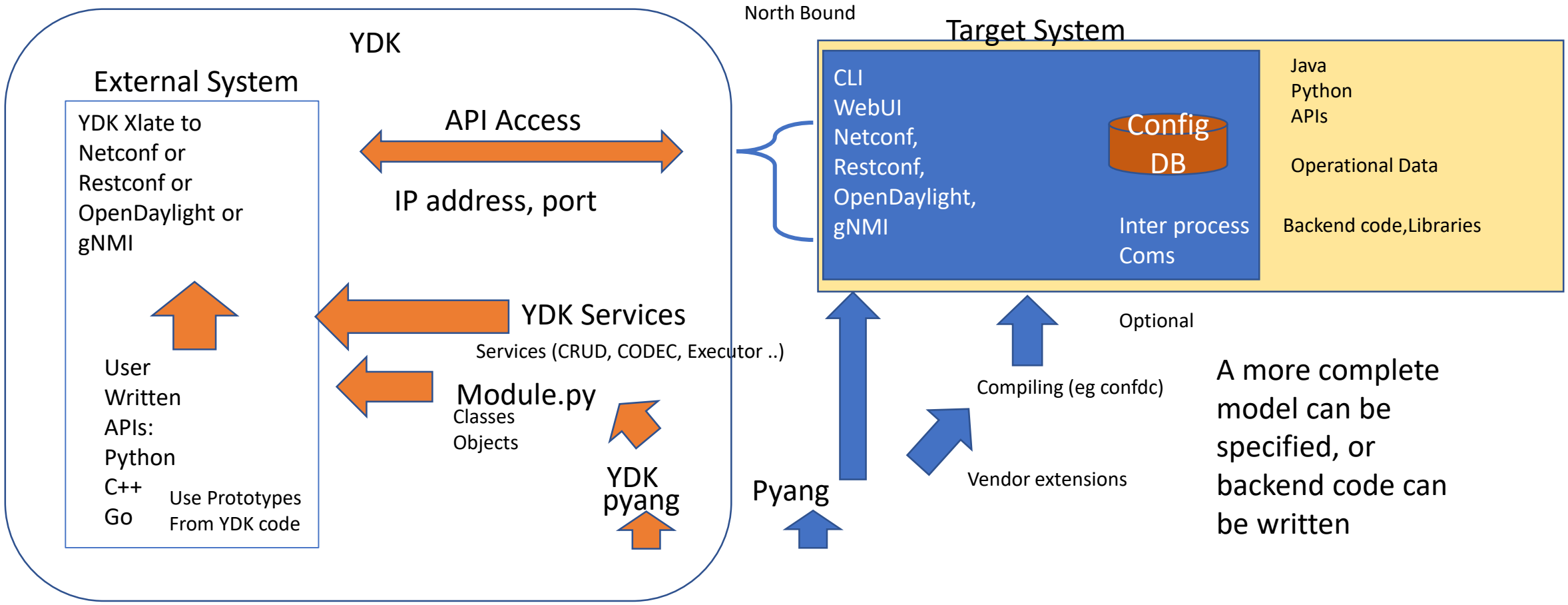
There are two areas YANG is being compiled today

1. For the Northbound Interface (Netconf, Restconf etc.)
   - In this case - augment the models and produce code for initialization, verification, interface to management functions
     - Highly dependent on the tool chain
   - Alternative is to code this manually
     - This is where, for example, defaults get initialized and configuration is validated before being committed.
2. For APIs to interface to the North bound interface from external
   - YANG Development Kit (YDK) produce class objects and code from YANG.

The common link for these is the YANG modle naming and typing of objects.

Note a Validated YANG model can be used by Netconf – I don't count this as compiling

YANG - YANG Development Kit - Ecosystem



Names and types are important

We are defining the starting point not the whole

Conclusions – For Discussion

- IEEE models support IEEE component model
  - This does add dependencies – typically to the bridge model
  - We could use more groupings to expose reusable pieces.
  - Don’t worry about code generation
    - yang validation – a must
    - confdc yuma123 etc helps if you can run the model
  - Don’t get hung up on defaults – go for functionally and readability.
- If you want code generation
  - Use the standard models as a base.
  - Augment these models for additional code generation

Thank You, Questions?

Tables Definition

Leaf-list – no defaults

```
list user-priority-tc {
      key "user-priority";
      description
        "Each entry in the Traffic Class Table is a
        traffic class, represented by an integer from
        0 through 7 that also comprises
        the numeric value of the four most
        significant bits of the Port Identifier
        component of the SCI for the selected SC";
      reference
        "IEEE 802.1AE-2018 Clause 10.7.17";
      leaf user-priority {
        type uint8 {
          range "0..7";
        }
        description
          " Deleted for example ";
      reference
        "IEEE 802.1AE-2018 Clause 10.7.17";
      }
      leaf traffic-class {
        type uint8 {
          range "0..7";
        }
        description
          " Deleted for example ";
        reference
          "IEEE 802.1AE-2018 Clause 10.7.17";
      }
    }
```

Container – with defaults

```
container user-priority-0 {
  description
    "Each entry in the Traffic Class Table is a
    traffic class, represented by an integer from
    0 (default) through 7 that also comprises the
    numeric value of the four most significant bits
    of the Port Identifier component of the SCI for
    the selected SC.";
  reference
    "IEEE 802.1AE-2018 Clause 10.7.17";
  leaf traffic-class {
    type uint8 {
      range "0..7";
    }
    default 0;
  }
}
```

●●●

```
container user-priority-7 {
  description
    "Each entry in the Traffic Class Table is a
    traffic class, represented by an integer from
    7 (default) through 7 that also comprises the
    numeric value of the four most significant bits
    of the Port Identifier component of the SCI for
    the selected SC.";
  reference
    "IEEE 802.1AE-2018 Clause 10.7.17";
  leaf traffic-class {
    type uint8 {
      range "0..7";
    }
    default 7;
  }
}
```

Simpler and functionally the same except for defaults

Tables Tree

Leaf-List

```
|  +--rw user-priority-tc* [user-priority]
|  |  +--rw user-priority    uint8
|  |  +--rw traffic-class?   uint8
```

Container

```
|  +--rw user-priority-0
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-1
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-2
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-3
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-4
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-5
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-6
|  |  +--rw traffic-class?   uint8
|  +--rw user-priority-7
|  |  +--rw traffic-class?   uint8
```