YANG++ Project

Andy Bierman
andy@yumaworks.com

IETF 121

V 0.1.0

Agenda

- Reuse problems with groupings
  - Background: yang-next issue #148
  - Focus: yang-next issue #140
- Next steps

YANG++ project

- YANG++ is a work-in-progress open-source project to create more powerful YANG modeling mechanisms and tools
  - Github: https://github.com/YumaWorks/yangpp
  - Docs: https://yangpp.yumaworks.com/en/latest/
- A YANG++ “class” is a cross between a YANG grouping and a YANG Schema Mount Point
  - XPath can be relative to the class root node (like a mount point) or other classes
  - All definitions automatically inherited from the parent class
- A YANG++ “uses-class” is a “uses’ statement that works like an identityref type
  - The grouping match is “derived-from-or-self” not exact match
  - The YANG library contains class bindings for the implementation

Groupings primer

- YANG grouping contents are only accessible in a data model if an accessible ‘uses’ statement references the grouping
- A Final Schema Tree is derived from the contents of the YANG Library
  - Modules, Revisions, and Features are all processed to produce the base schema tree
  - “augment” statements contents are expanded within the target node
  - “uses” statements are replaced with the referenced grouping contents
- Only the final schema tree is needed for protocol implementation
  - Groupings and other metadata may not be saved or even used

Problems with groupings

- P1) Not a real abstraction layer
- P2) Groupings are not always relocatable
- P3) Groupings cannot be augmented
- P4) Groupings are fixed and non-replaceable

P1) Not a real abstraction layer

- A grouping is just a collection of YANG statements that get expanded in the schema tree like a C #define
- There are no “boundaries” or structure in machine readable statements
- The type of layered model shown in RFC 6095 Extending YANG With Language Abstractions is difficult to create in YANG
- Goal: Make it easy to use groupings as more than a collection of statements
- Solution Paths:
  - Improve documentation guidelines (description-stmt only tool now)
  - New TBD YANG 2.0 statements
  - YANG++ Proposal: Virtual Objects

Abstraction layer example 1

Empty Mandatory Choice

Example: A virtual case within a concrete choice

```
class options {
  virtual {
    augment "config/mand-choice" {
      case <mand-case>;
    }
  }

  container config {
    choice mand-choice {
      mandatory true;
    }
  }
}
```

```
class my-options {
  parent-class options {
    map-virtual mand-case {
      map-path "config/mand-choice/my-case";
    }
  }

  augment "config/mand-choice" {
    case my-case {
      leaf my-leaf { type string; }
    }
  }

}
```

Abstraction layer example 2

Service Template

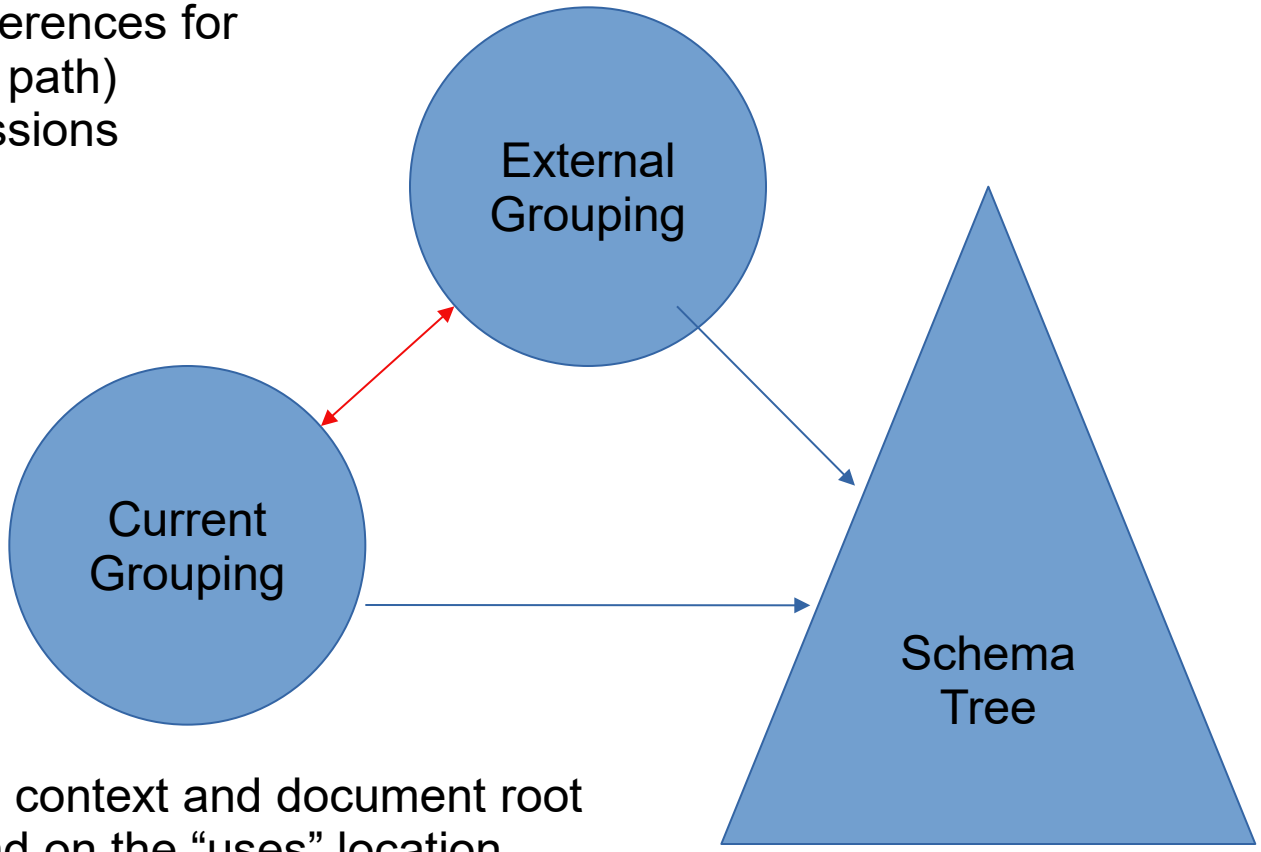
```
module base-service {
  // ..

  class base-service {
   virtual {
     // concrete class expected to map these 2 statements
     // and provide names for these actions
     action <reset>;
     action <status>;

     // concrete class expected to fill in this config container
     // as needed and use the provided name 'config'
     container config;
   }
 }

}
```

YANG++ Examples

```
class mybase-service {
 parent-class base:base-service {
   map-virtual reset {
     map-path my-reset;
   }
   map-virtual status {
     map-path my-status;
   }
 }

 // local groupings work the same since the class root
 // is a real container or list node
 grouping status-parms {
    leaf status {
      type string;
    }
    leaf last-error {
      type string;
    }
 }

 // no virtual sections in this class makes it a concrete class
 // a concrete definition for each 'virtual' definition is expected.
 // if missing then a deviate (not-supported) is implied
 action my-reset {
   input {
     leaf myparm1 { type string; }
   }
 }

 action my-status {
   output {
     uses status-parms;
   }
 }

 // this is a replacement of the virtual config
 container config {
   list list1 {
     key name1;
     leaf name1 { type string; }
     leaf my-leaf2 { type string; }
   }
 }
}
```

Grouping cross references

3 types of references for
(must, when, path)
XPath expressions



XPath context and document root
depend on the “uses” location

P2) Groupings are not always relocatable

- All XPath cross references are for the final schema tree, even if relative
- There is no easy way to use the conceptual schema tree(s) in a position-independent manner
- Goal: Make it easy to re-locate a grouping in a different part of the schema tree
- Solution Paths:
  - Improve documentation guidelines (description-stmt only tool now)
  - New TBD YANG 2.0 statements
  - New TBD XPath variables and/or functions
  - YANG++ Proposal: Position Independent YANG

Relocatable groupings example

Abstract push capabilities

```
class push-caps {
  leaf min-interval {
    type uint32;
    units centiseconds;
  }
  leaf max-segment-size {
    type uint32;
    units bytes;
  }
}
```

```
class push-settings {
  classref "sys:push-caps::syscaps" {
    description "System capabilities to use";
  }

  leaf min-interval {
    type uint32;
    units centiseconds;
    must ". >= /sys:push-caps::syscaps/min-interval";
  }
  leaf max-segment-size {
    type uint32;
    units bytes;
    must ". <= /sys:push-caps::syscaps/max-segment-size";
  }
}
```

Real /system

```
container system {
  uses-class push-caps {
    root-name "push-capabilities";
  }
}
```

```
container settings {
  uses-class push-settings {
    bind-classref sys:push-caps::syscaps {
      path "/sys:system/sys:push-capabilities";
    }
  }
}
```

P3) Groupings cannot be augmented

- Only the initial and unchangeable grouping can be used or else rewrite the module with the “uses”
- All augmentations must be made on the data structures at each “uses” expansion
  - This can be error-prone and very difficult to maintain
- This could be very complicated to implement
  - Augment-grouping was considered for YANG 1.1 and rejected
  - All grouping augmentations may need to be collected before a grouping can be expanded

Augmenting groupings solution path

- Recent standards improvements could help
  - The YANG Library “augmented-by” solution could apply here
  - The YANG Packages work may also help make this more feasible now
  - YANG Conformance issues are now addressed with the YANG Semver work
- YANG 2.0 TBD augment-grouping
  - Ad-hoc just like augment
    - Very flexible but most complex
- YANG++ Proposal: Classes
  - Single inheritance too simplistic but may be easier to implement

Augmenting grouping example

```
grouping address {
  // … 4 leafs
}

grouping us-address {
  uses address;
  leaf zipcode { … }
}

container address {
  uses address;
}

// YANG++
uses-class address;
```

The 'address' class represents one generic address:

```
class address {
  leaf last-name {
    type string;
    description
      "Last name of the person who is associated with this address";
  }
  leaf first-name {
    type string;
    description
      "First name of the person who is associated with this address";
  }
  leaf street {
    type string;
    description "street address";
  }
  leaf city {
    type string;
    description "City address";
  }
}
```

The 'us-address' class has all the leafs as the parent class plus a zipcode, ordered after the last parent child node.

```
class us-address {
  parent-class address;

  leaf zipcode {
    type string { length "5 | 10"; }
    description "zipcode";
  }
}
```

P4) Groupings are fixed and non-replaceable

- The only way to upgrade a grouping is to change each uses-stmt
- Solution Approach
  - YANG 2.0 derived-from-or-self “uses”
    - uses-stmt declares 1 or more bases
    - Grouping declares 1 or more bases to match 1 or more uses bases
    - YANG Library identifies implemented-to-specified grouping mappings
    - YANG Packages could specify standard or vendor grouping replacements
  - YANG++ Proposal: Classes
    - Single inheritance too simplistic but may be easier to implement

Replaceable groupings example

Example: Envelope with from-address and to-address sub-trees.

```
class envelope {
  uses-class address {
    root-name from-address;
  }
  uses-class address {
    root-name to-address;
  }
}
```

If the YANG Library binds class ‘us-address’ to ‘address’ then each ‘uses-class address’ will expand to have the base leafs and the zipcode leaf.

Next steps

- Decide which issues (if any) should be addressed in YANG 2.0
- Evaluate solution paths
- Pick solutions
- Write the YANG specifications