OpenConfig Reply
draft-bjorklund-netmod-openconfig-reply-00

Jürgen Schönwälder, Jacobs University

OpenConfig Discussion Background

- OpenConfig proposals at IETF 92 (March 2015)
  - draft-openconfig-netmod-model-structure-00
- Virtual Interim Meetings (June 2015)
  - draft-openconfig-netmod-opstate-01
- Response to OpenConfig proposal (July 2015)
  - draft-bjorklund-netmod-openconfig-reply-00

#1: "asynchronous" implementations

- OpenConfig proposal:
  - All configuration nodes are duplicated in a separate config false branch of the data tree, in order to support "asynchronous" implementations.
  - The config true part of the tree is called "intended configuration”, and is always modeled under a NP container "config", and the duplicate part is called "actual configuration", and is always modeled under a NP container "state".

#1: "asynchronous" implementations

- Cons:
  - Data model duplication. This reduces the readability of the models, uses additional memory in implementations, and adds to the complexity of the overall solution.
  - Fragile solution, since it relies on a convention; if a module doesn't follow this convention, that module cannot be used in a such a system.
  - Adding semantics to node names instead of using YANG statements.
  - Relies on groupings and therefore the YANG constraints will be applied to the config=false copy, even if the operational state does not have the same restrictions as the config.
  - Mandatory for all platforms even if they are not synchronous.
  - Leafrefs will be broken - if a leafref has the "config" node in its path, then the "state" version of the leafref still refers to the "config" node.

#1: "asynchronous" implementations

- Alternate proposal:
  - We propose that the actual config is handled as a separate conceptual datastore. In NETCONF, this can be done by using get-config with a new datastore name. Other protocols would do something similar.

#1: "asynchronous" implementations

- Pros:
  - This works without rewriting any modules already published. It works with vendor models and models from other SDOs that do not follow these conventions.
  - Doesn't cost anything for synchronous platforms.
- Cons:
  - Needs a datastore, a concept that not all protocols support.
  - Note: A protocol does not have to expose datastores in the same way as NETCONF in order to support this concept.

#2: relate config to state

- OpenConfig proposal:
  - All data models always add two special NP containers "config" and "state" which contains all leafs, in order to have a simple and deterministic way to related config to state.

#2: relate config to state

- Cons:
  - Too simplistic. Doesn't solve the problem that the state may contain different instances than config. Configuration data and operational instances are really separate entities, and should be modeled as such.
  - Adds noise to the data model.
  - Fragile solution, since it relies on a convention; if a module doesn't follow this convention, that module cannot be used in a such a system.
  - Adds semantics to node names instead of using YANG statements.

#2: relate config to state

- Alternate proposal:
  - Follow the solution used in RFC 7223 (branch at the root, /interfaces and /interfaces-state)
  - It should be noted that in both the open config proposal and the scheme from RFC 7223, there is no formal way to know the relationship between the configuration and its related operational state. It might be worthwhile to try to define YANG statements to solve this problem instead.

#3: top-level node

- OpenConfig proposal:
  - All data models intended for devices are rooted under a top-level node called "device".
- Cons:
  - Doesn't solve any problem. It just adds 7 characters to all paths. Whatever problem there is today with "/FOO" will now be a problem with "/device/FOO".
  - All existing models need to be rewritten, including vendor models.

#3: top-level node

- Alternate proposal:
  - Use "/" as the top level.

Discussion

- How do we continue this discussion and conclude it in a timely manner?