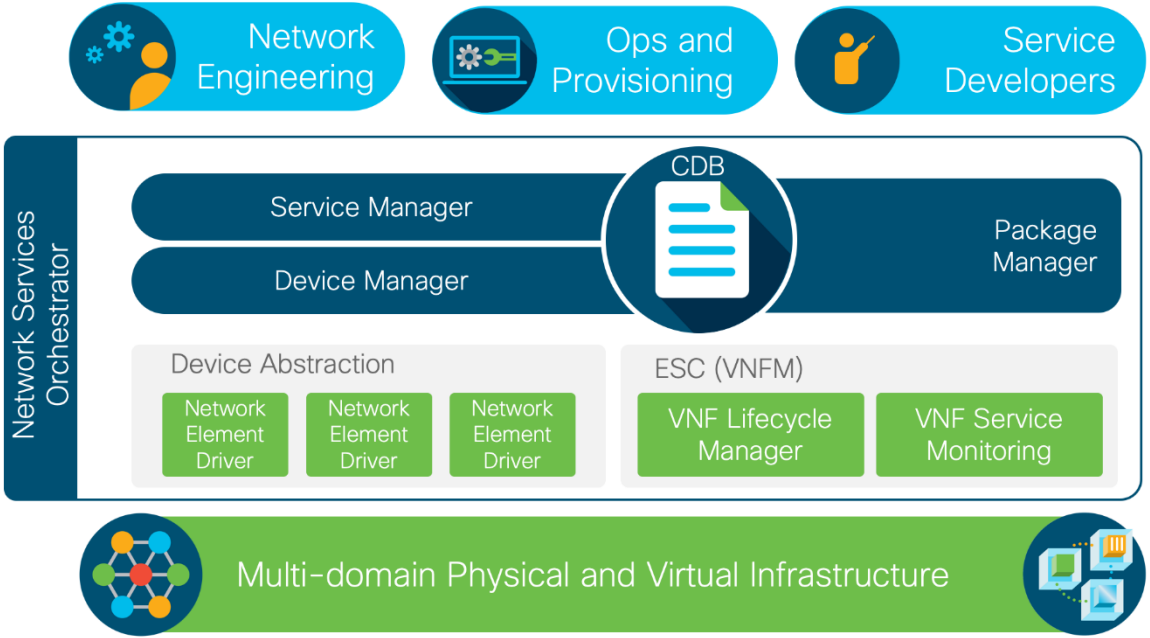
Developer Days
Network Services Orchestrator

Jinja2 Templates and Model Independence

The joys of model driven code

Viktor Leijon
Principal Engineer, Mobility and Automation

NSO Architecture



One small word

Model-driven, end-to-end service lifecycle and customer experience focused

Seamless integration with northbound tooling

Loosely-coupled and modular architecture leveraging open APIs and standard protocols

Orchestration across multi-domain and multi-layer for network-wide, centralized policy and services

NSO does it with models

- All data in CDB is described by a YANG model
- All the APIs in NSO are driven by the model
- The CLI and WebUI changes with the model

Why can't your code be the same?

```
list nodes {
    key name;
    leaf name {
      type string;
    }
    leaf info {
      type uint16;
    }
    ...
}
```

Striving for modularity

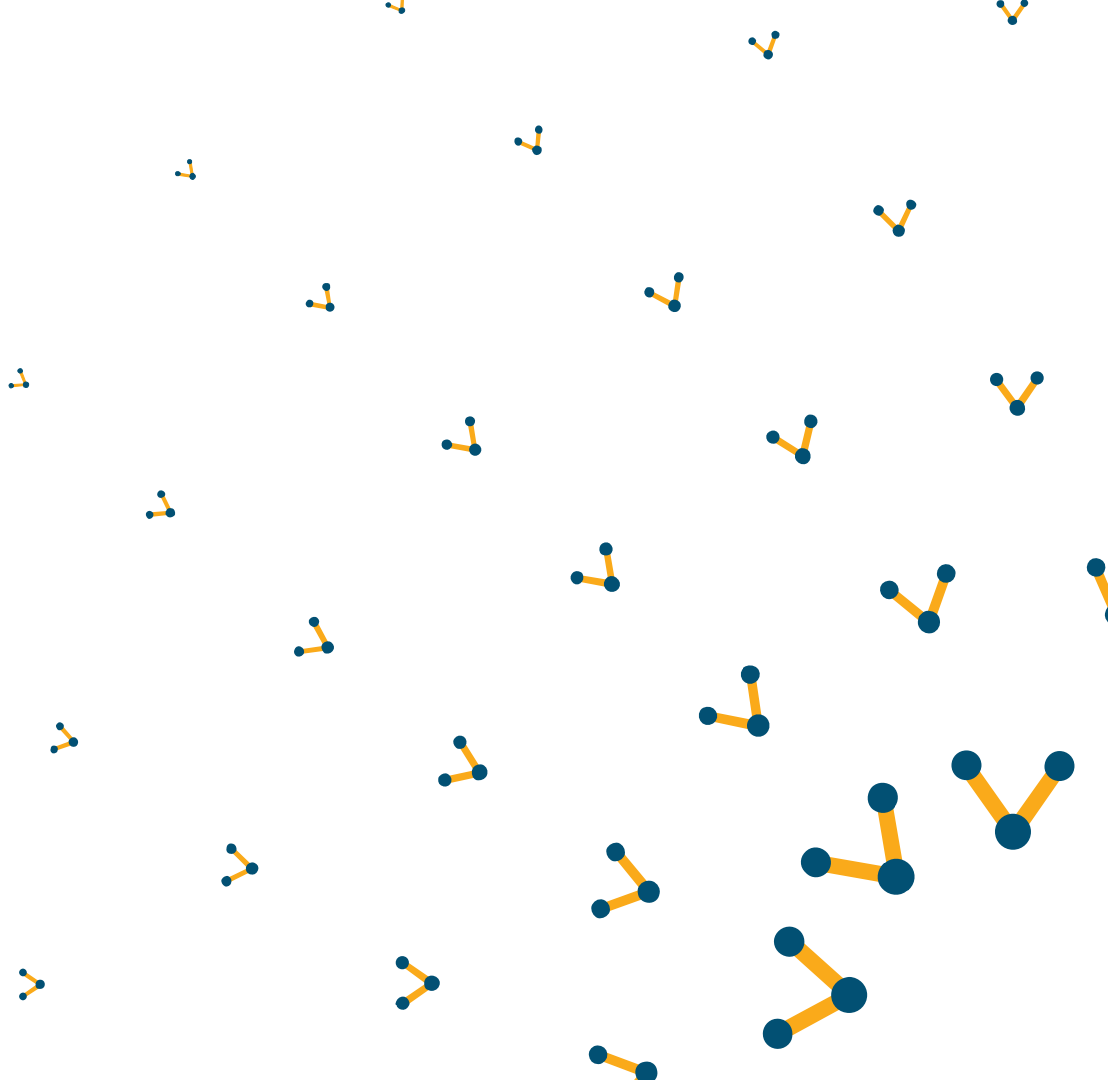
- Key/Value lists injected into custom templates
- Leaves pointing to device templates that are applied
- Embedding a JSON document in a string leaf
- Other forms of trickery
- Hides the structure of the data

```
list params {
    key name;
    leaf name {
      type string;
    }
    leaf value {
      type string;
    }
}
```

Luckily NSO provides a schema API!

- You don’t have to know your model in advance
- Possible to dynamically react to changes in the model
- But, who changes the model then?
  - Your code might be a library
  - Customers might do customization
  - Maybe you want to auto-generate from templates

Basic Schema



What is the schema API?

- Available in all development APIs
  - C, Java, Python, Erlang, JSON-RPC
  - NETCONF/RESTCONF supports RFC8525 (YANG library)
- We will be talking Python today.
- The `CsNode`/`CsNodeInfo` is the same in all APIs
  - Directly reflects NSO-internal structures
  - Corresponds to the fxs-files you build from YANG
- Fairly primitive

A tiny example

```
def is_list(path):
  with maapi.single_read_trans('islist', 'system') as t:
    n = maagic.get_node(t, path)
    return n._cs_node.is_list()
```

Question:
Is there a better way to do this?

```
>>> islist.is_list('/devices')
False
>>> islist.is_list('/devices/device')
True
```

`pydoc _ncs.CsNode`

CsNode

- Represents a node in the schema tree
- Compact representation
  - Use `_ncs.hash2str()` to decode tags.
  - Most other fields are integers mapping to constants: `_ncs.C_INT16` et c. (see `pydoc _ncs` for a full list)
- Represents the **kind** of the current node
- Contains the tree structure of the model

```
pydoc _ncs.CsNodeInfo
```

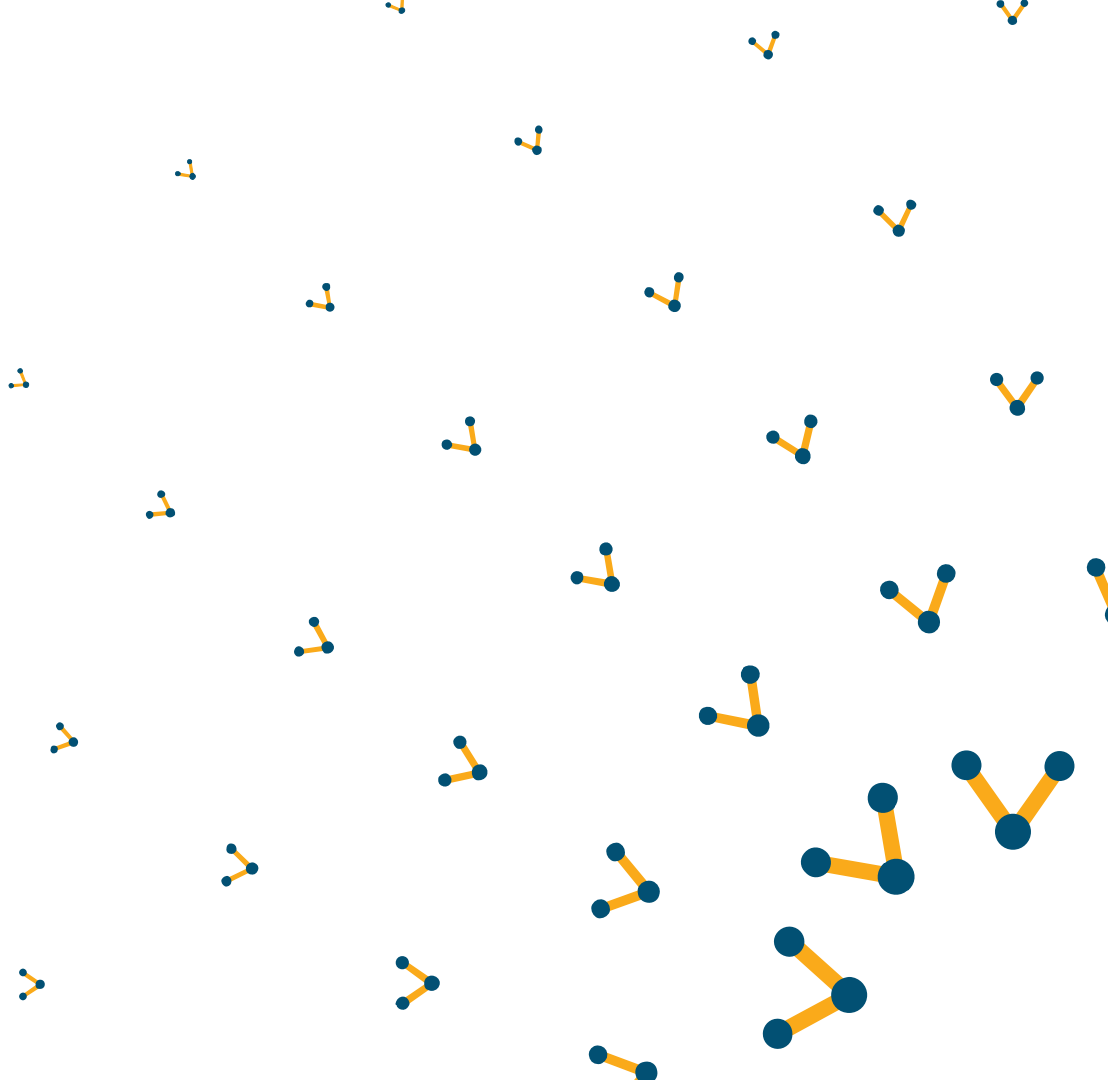
CsNodeInfo

- Detailed information about a node
- List information
  - Cardinality
  - Keys
- Actual data-type for a leaf
- Meta-data nodes for the node
  - `tailf:meta-data` – can be used for extra-deep magic.

A larger example – dumptree.py

```
def dumpNode(cs, recursive=True, depth = 0):
  cstype = cs2type(cs)
  name = _ncs.hash2str(cs.tag())
  tval = gettype(cs)
  print('{}{} {} {}'.format(' '*depth, cstype, name, tval))
  c = cs.children()
  while c:
    dumpNode(c, depth=depth+1)
    c = c.next()
```

maagic Schema



Free schema with your python

- Iterate over your children
- Built in type information
- Extract `_cs_node` for extra information.

```
pydoc ncs.maagic.Node
```

```
def is_list_maagic(path):
  with maapi.single_read_trans('islist',
       'system') as t:
    n = maagic.get_node(t, path)
    return isinstance(n, maagic.List)
```

Showing all children

```
def children(path):
  with maapi.single_read_trans('getchildren','system') as t:
    n = maagic.get_node(t, path)
    for child in n:
      (prefix, name) = child.split(':')
      childnode = n[name]
      print("Child: {}, Type: {}".format(name, type(childnode)))
```

NSO ❤️ Jinja2

Detour: `load-native-config`

- Requires NED support
- Not *true* native - limited to what the NED supports
- Mostly relevant for CLI NEDs
- Action in config mode

```
# devices device ios0 load-native-config verbose
data "interface GigabitEthernet0/0\nip address
192.168.1.1 255.255.255.0"
info
  Number of lines parsed             :      4
  Number of lines skipped            :      0

admin@ncs(config)# show conf
devices device ios0
 config
  interface GigabitEthernet0/0
   ip address 192.168.1.1 255.255.255.0
  exit
 !
!
```

Demo: Jinja2 templates

- Use the schema to extract config values
- Feed the values into a jinja2 template
- Use `load-native-config` to load the rendered template
- https://gitlab.com/nso-developer/example-model-driven-jinja2

Conclusion

- The schema API is cool
- All the NSO advantages:
  - Nice CLI
  - Input validation
  - Structured payloads
- The possibilities are endless – what can you drive with a model?


CISCO