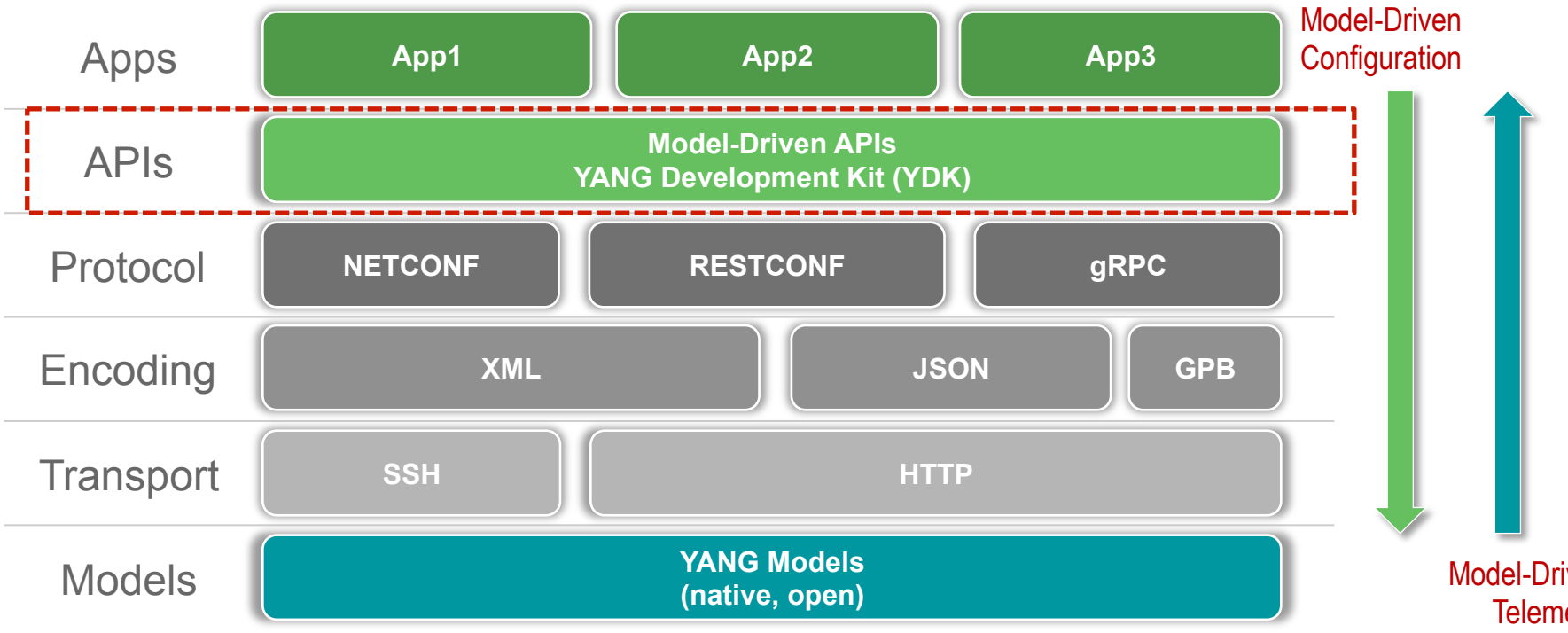
Ok, We Got Data Models, Now What?

Santiago Álvarez

@111pontes

Model-Driven Programmability Stack



 saalvare@cisco.com

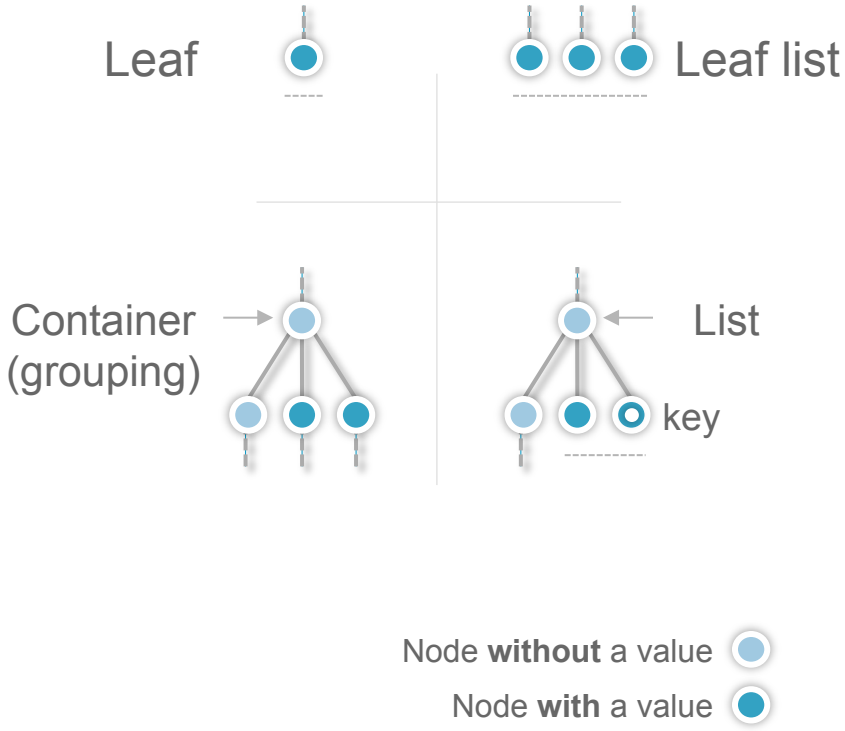
Benefits of Model-Driven Programmability

- Model based, structured, computer friendly
- Multiple model types (native, OpenConfig, IETF, etc.)
- Models decoupled from transport, protocol end encoding
- Choice of transport, protocol and encoding
- Model-driven APIs for abstraction and simplification
- Wide standard support while leveraging open source

saalvare@cisco.com

YANG

- Modeling language for network devices
- Main node types
  - **Leaf** – node with name and value of certain type (no children)
  - **Leaf list** – sequence of leafs
  - **Container** – groups nodes and has no value
  - **List** – Sequence of records with key leafs
- Augmentations extend a model
- Deviations specify divergence from model



YANG Model Example

YANG

```
container community-sets {
       description “Container for community sets";
       list community-set {
         key community-set-name;
         description "Definitions for community sets";
         leaf community-set-name {
           type string;
           description "name of the community set";
         }
         leaf-list community-member {
           type string {
             pattern '([0-9]+:[0-9]+)';
           }
           description "members of the community set";
         }
       }
     }
```

CLI

```
community-set C-SET1
  65172:1,
  65172:2,
  65172:3
end-set
!
community-set C-SET10
  65172:10,
  65172:20,
  65172:30
end-set
!
```


saalvare@cisco.com

Model Data Example

XML

```
<community-sets>
  <community-set>
    <community-set-name>C-SET1</community-set-name>
    <community-member>65172:1</community-member>
    <community-member>65172:2</community-member>
    <community-member>65172:3</community-member>
  </community-set>
  <community-set>
    <community-set-name>C-SET10</community-set-name>
    <community-member>65172:10</community-member>
    <community-member>65172:20</community-member>
    <community-member>65172:30</community-member>
  </community-set>
</community-sets>
```

CLI

```
community-set C-SET1
  65172:1,
  65172:2,
  65172:3
end-set
!
community-set C-SET10
  65172:10,
  65172:20,
  65172:30
end-set
!
```


saalvare@cisco.com

Model Data Example

JSON

```
{   "community-sets": {
        "community-set": [
            {   "community-set-name": "C-SET1",
                "community-member": [
                    "65172:1",
                    "65172:2",
                    "65172:3" ]
            },
            {   "community-set-name": "C-SET10",
                "community-member": [
                    "65172:10",
                    "65172:20",
                    "65172:30" ]
            }
        ]
    }
}
```

CLI

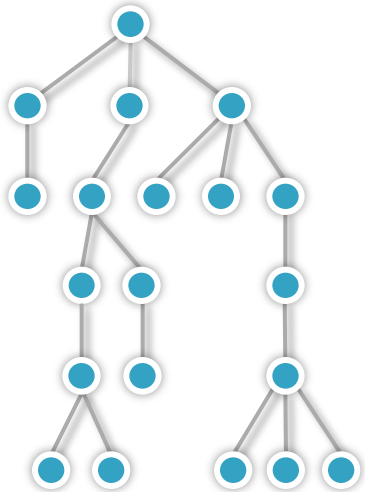
```
community-set C-SET1
  65172:1,
  65172:2,
  65172:3
end-set
!
community-set C-SET10
  65172:10,
  65172:20,
  65172:30
end-set
!
```


saalvare@cisco.com

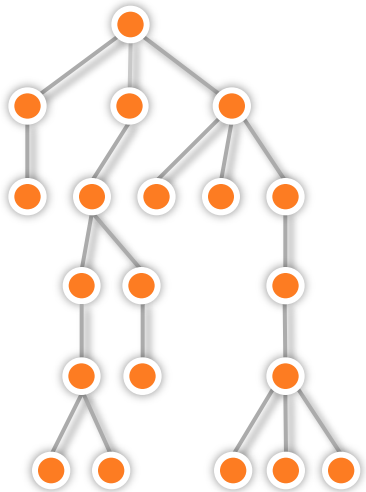
Model-Driven APIs

- Simplify app development
- Abstract transport, encoding, model language
- API generated from YANG model
- One-to-one correspondence between model and class hierarchy
- Multi-language (Python, C++, Go, Ruby, etc.)

YANG Model

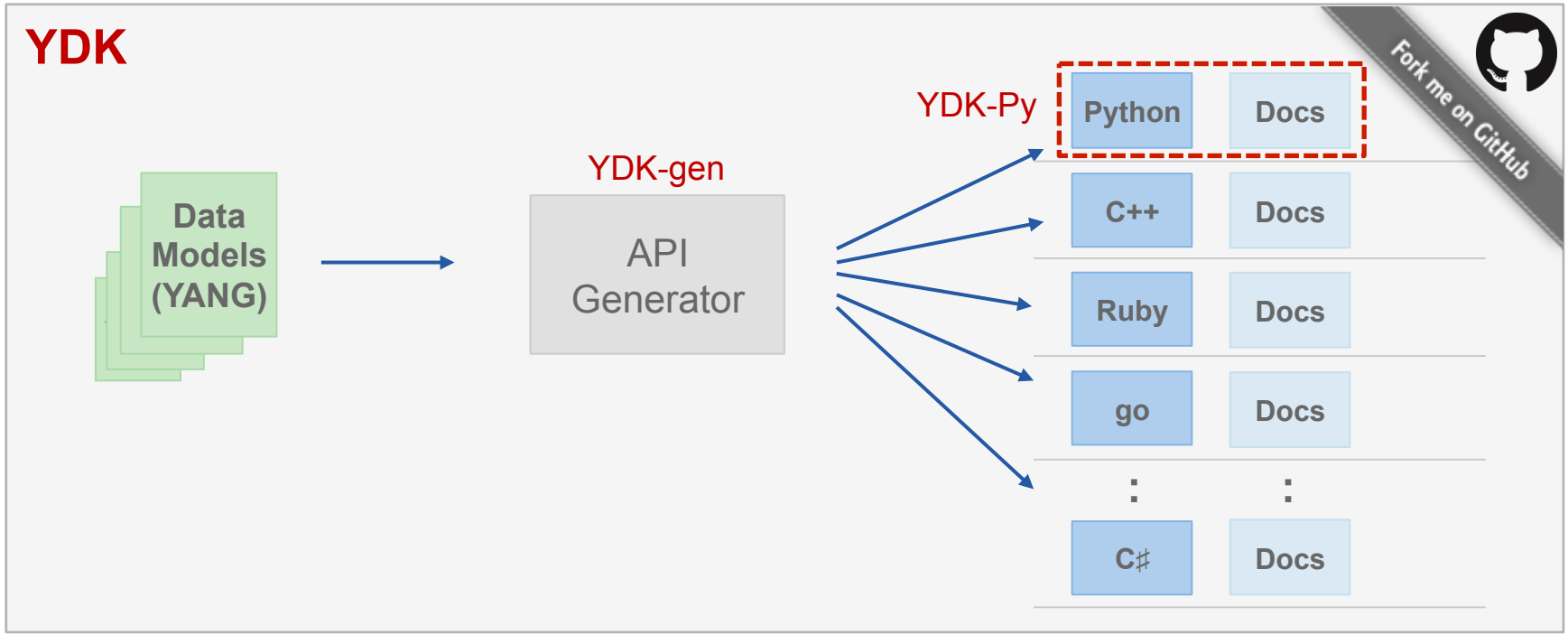


Class Hierarchy (Python, C++, Ruby, Go)

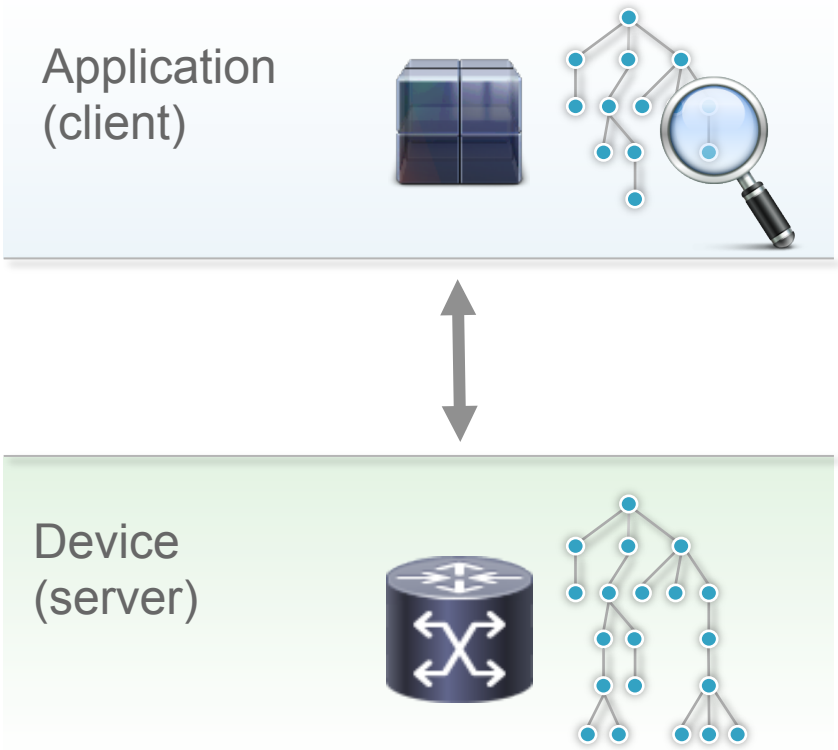


saalvare@cisco.com

Generation of Model-Driven APIs Using YANG Development Kit (YDK)



Client-Side Validation



- Model constraints used during API generation
- YDK service will automatically perform local (client-side) validation
- Config (read-write) vs. state (read-only)
- Type check (enum, string, etc.)
- Value check (range, pattern, etc.)
- Semantic check (key uniqueness/presence, mandatory leafs, etc.)
- Model deviation check (unsupported leaft, etc.)

saalvare@cisco.com

A YDK-Py “Hello World” Using OpenConfig BGP

```
# Cisco YDK-Py OC-BGP “Hello world”
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models.openconfig import openconfig_bgp as oc_bgp

if __name__ == "__main__":
    provider = NetconfServiceProvider(address=10.0.0.1,
                                      port=830,
                                      username=“admin”,
                                      password=“admin”,
                                      protocol=“ssh”)
    crud = CRUDService()  # create CRUD service
    bgp = oc_bgp.Bgp()  # create oc-bgp object
    bgp.global_.config.as_ = 65000  # set local AS number
    crud.create(provider, bgp)  # create on NETCONF device
    provider.close()
    exit()
# End of script
```

```
module: openconfig-bgp
   +--rw bgp
      +--rw global
      |  +--rw config
      |  |  +--rw as
      |  |  +--rw router-id?
      |  +--ro state
      |  |  +--ro as
      |  |  +--ro router-id?
      |  |  +--ro total-paths?
      |  |  +--ro total-prefixes?
...
```


saalvare@cisco.com

A YDK-Py Routing Policy Example

Python

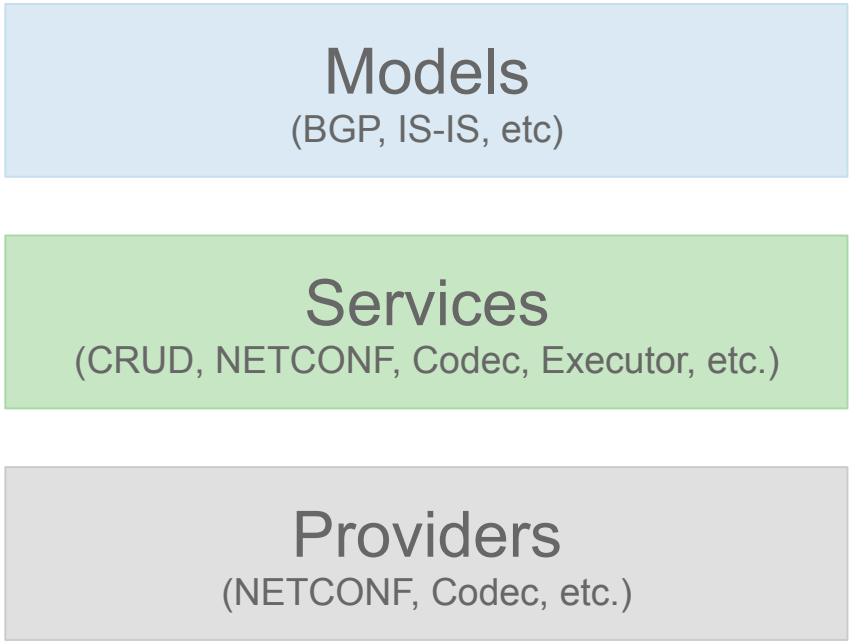
```
# community set configuration
c_set = bgp_defined_sets.community_sets.CommunitySet()
c_set.community_set_name = "C-SET1"
c_set.community_member.append("65172:1")
c_set.community_member.append("65172:2")
c_set.community_member.append("65172:3")
bgp_defined_sets.community_sets.community_set.append(c_set)

# community set configuration
c_set = bgp_defined_sets.community_sets.CommunitySet()
c_set.community_set_name = "C-SET10"
c_set.community_member.append("65172:10")
c_set.community_member.append("65172:20")
c_set.community_member.append("65172:30")
bgp_defined_sets.community_sets.community_set.append(c_set)
```

CLI

```
community-set C-SET1
  65172:1,
  65172:2,
  65172:3
end-set
!
community-set C-SET10
  65172:10,
  65172:20,
  65172:30
end-set
!
```

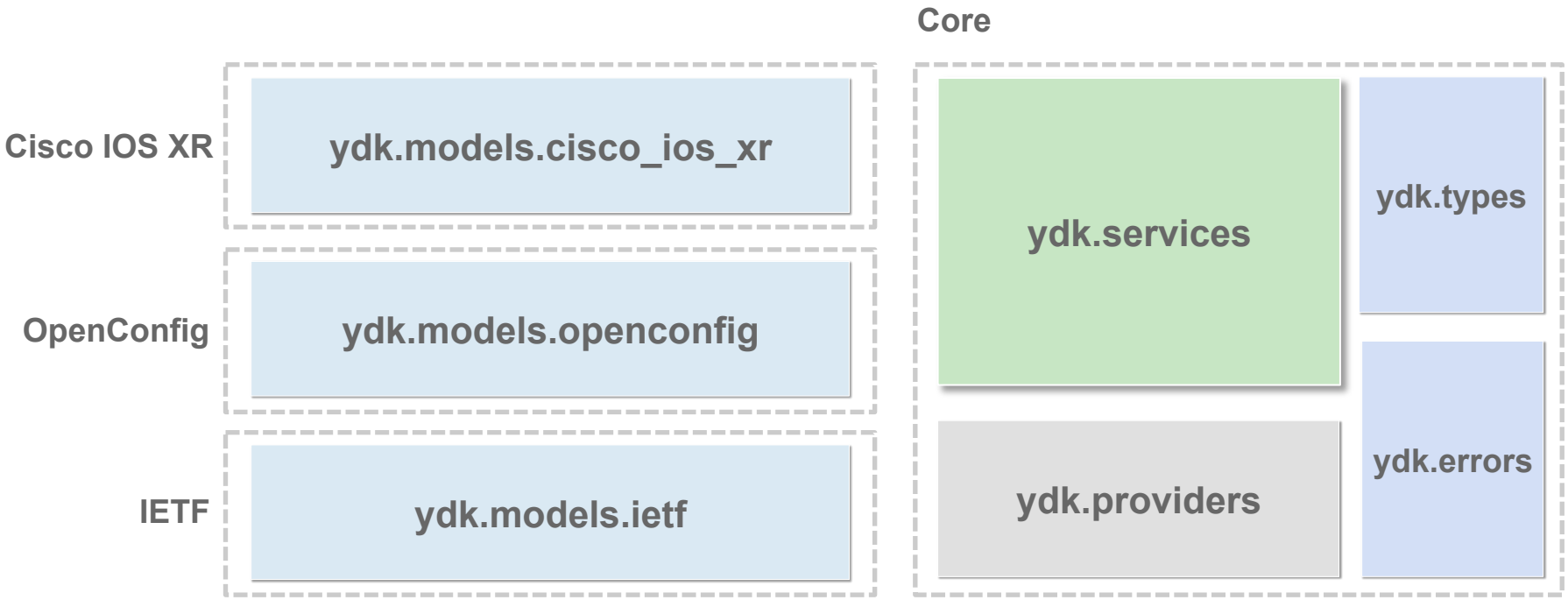

saalvare@cisco.com

YDK API Structure

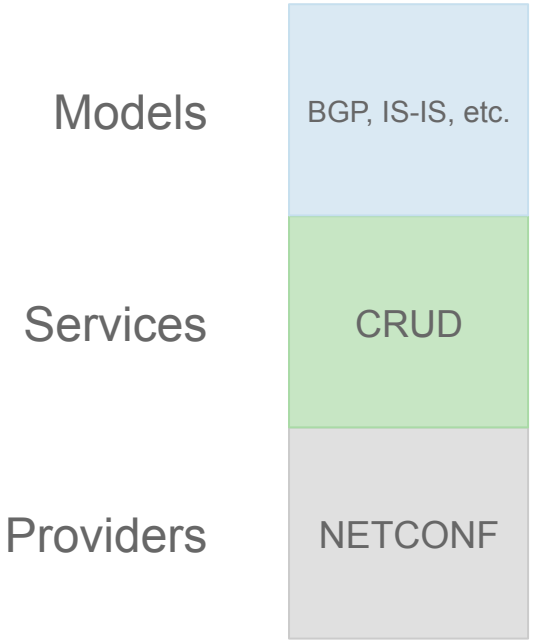


- **Models** group Python APIs created for each YANG model
- **Services** perform operations on model objects (interface)
- **Providers** implement services (implementation)

YDK-Py 0.5.0 Package Structure (Bundles)



CRUD Service



- Create / Read / Update / Delete interface (abstracts NETCONF operations)
- All operations can be invoked on configuration data
- Only read operation can be invoked on operational data
- Create/merge operation merge configuration
- Data validation performed when CRUD operation invoked
- Uses NETCONF provider

YDK-Py CRUD Service

- Class

  ydk.services.CRUDService

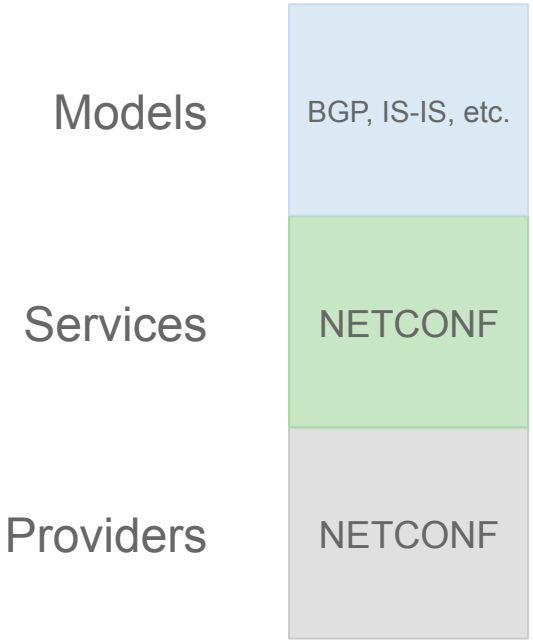
- Methods:

```
create(provider, entity)
read(provider, read_filter, only_config=False)
update(provider, entity)
delete(provider, entity)
```

Example for CRUD Service

```
# import providers, services and models
from ydk.services import CRUDService
from ydk.providers import NetconfServiceProvider
from ydk.models.openconfig import openconfig_bgp as oc_bgp

# create NETCONF provider
provider = NetconfServiceProvider(address=device.hostname,
                                  port=device.port,
                                  username=device.username,
                                  password=device.password,
                                  protocol=device.scheme)

crud = CRUDService() # create CRUD service
bgp = oc_bgp.Bgp()  # create config model object
config_bgp(bgp)  # add object configuration
crud.create(provider, bgp)  # create object on NETCONF device
provider.close()
```

NETCONF Service



- Direct interface to NETCONF operations
- Data validation performed when NETCONF operation invoked
- Uses NETCONF provider

YDK-Py NETCONF Service

- Class

  ydk.services.NetconfService

- Methods:

```
get_config(provider, source, get_filter, with_defaults_option=None)
edit_config(provider, target, config, default_operation=None,
            error_option=None, test_option=None)
get(provider, get_filter, with_defaults_option=None)
commit(provider, confirmed=False, confirm_timeout=None, persist=False,
       persist_id=None)
lock(provider, target)
unlock(provider, target)
close_session(provider)
```

YDK-Py NETCONF Service (cont.)

- Methods (cont.):

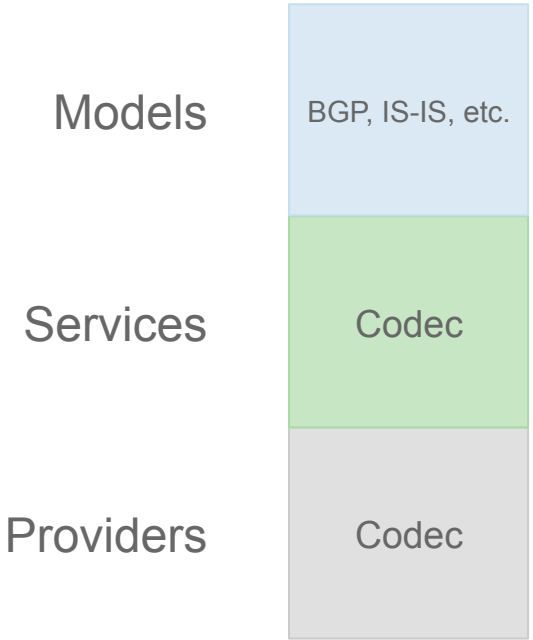
```
cancel_commit(provider, persist_id=None)
copy_config(provider, target, source, with_defaults_option=None)
delete_config(provider, target)
discard_changes(provider)
kill_session(provider, session_id)
validate(provider, source=None)
```


saalvare@cisco.com

Example for NETCONF Service

```
# import providers, services and models
from ydk.services import NetconfService, Datastore
from ydk.providers import NetconfServiceProvider
from ydk.models.openconfig import openconfig_bgp as oc_bgp

# create NETCONF provider
provider = NetconfServiceProvider(address=device.hostname,
                                  port=device.port,
                                  username=device.username,
                                  password=device.password,
                                  protocol=device.scheme)
netconf = NetconfService()  # create NETCONF service
bgp = oc_bgp.Bgp()  # create config model object
config_bgp(bgp)  # add object configuration
netconf.edit_config(provider, Datastore.candidate, bgp)  # edit config on NETCONF device
netconf.commit(provider)  # commit config on NETCONF device
provider.close()
```

Codec Service



- Interface to encode and decode model data
- Encoding converts model object to encoded string
- Decoding converts encoded string to model object
- Uses codec provider initialized for a specific encoding type (e.g. XML)

YDK-Py Codec Service

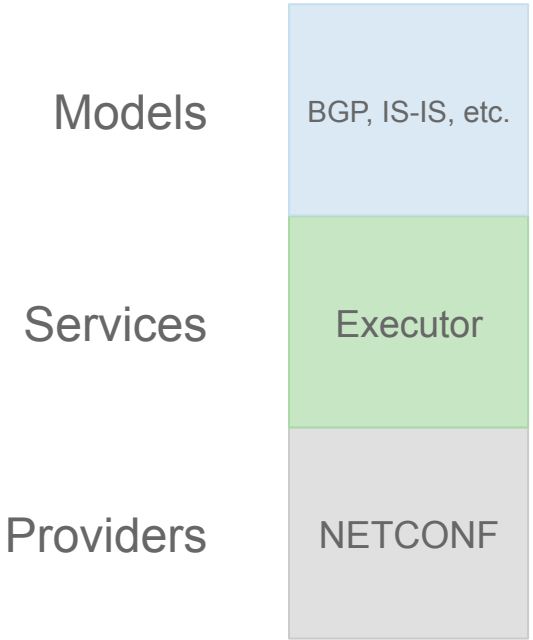
- Class
  `ydk.services.CodecService`
- Methods:
  `encode``(provider, entity)`
  `decode``(provider, payload)`

saalvare@cisco.com

Example for Codec Service

```
# import providers, services and models
from ydk.services import CodecService
from ydk.providers import CodecServiceProvider
from ydk.models.openconfig import openconfig_bgp as oc_bgp

# create codec provider
provider = CodecServiceProvider(type='xml')

codec = CodecService() # create codec service

bgp = oc_bgp.Bgp()  # create config model object
config_bgp(bgp)  # add object configuration

print(codec.encode(provider, bgp))  # print model object encoded in XML

provider.close()
```

Executor Service



- Interface to execute model RPCs
- Input/output validation performed when RPC execution operation invoked
- Uses NETCONF provider

YDK-Py Executor Service

- Class
  `ydk.services.ExecutorService`
- Methods:
  `execute_rpc``(provider, rpc)`

saalvare@cisco.com

Example for Codec Service

```
# import providers, services and models
from ydk.services import ExecutorService
from ydk.providers import NetconfServiceProvider
from ydk.models.cisco_ios_xr_act import Cisco_IOS_XR_syslog_act as xr_syslog_act

# create NETCONF provider
provider = NetconfServiceProvider(address=device.hostname,
                                  port=device.port,
                                  username=device.username,
                                  password=device.password,
                                  protocol=device.scheme)
executor = ExecutorService()  # create executor service
syslog_act = xr_syslog_act.LogmsgRpc()  # create RPC object
syslog_act.input.message = "My very own syslog message!"
syslog_act.input.severity = xr_syslog_act.SeverityEnum.CRITICAL
executor.execute_rpc(provider, syslog_act)  # execute rpc
provider.close()
```

YDK-Py Logging

- Uses Python logging facility

  Handlers – send log records to a given destination(stream, file, socket, etc.)

  Filters - determine which log records to output

  Formatters - specify layout of log records

- Logger names follow package hierarchy and determine granularity
- Sample logger names from less to more specific

  `ydk`

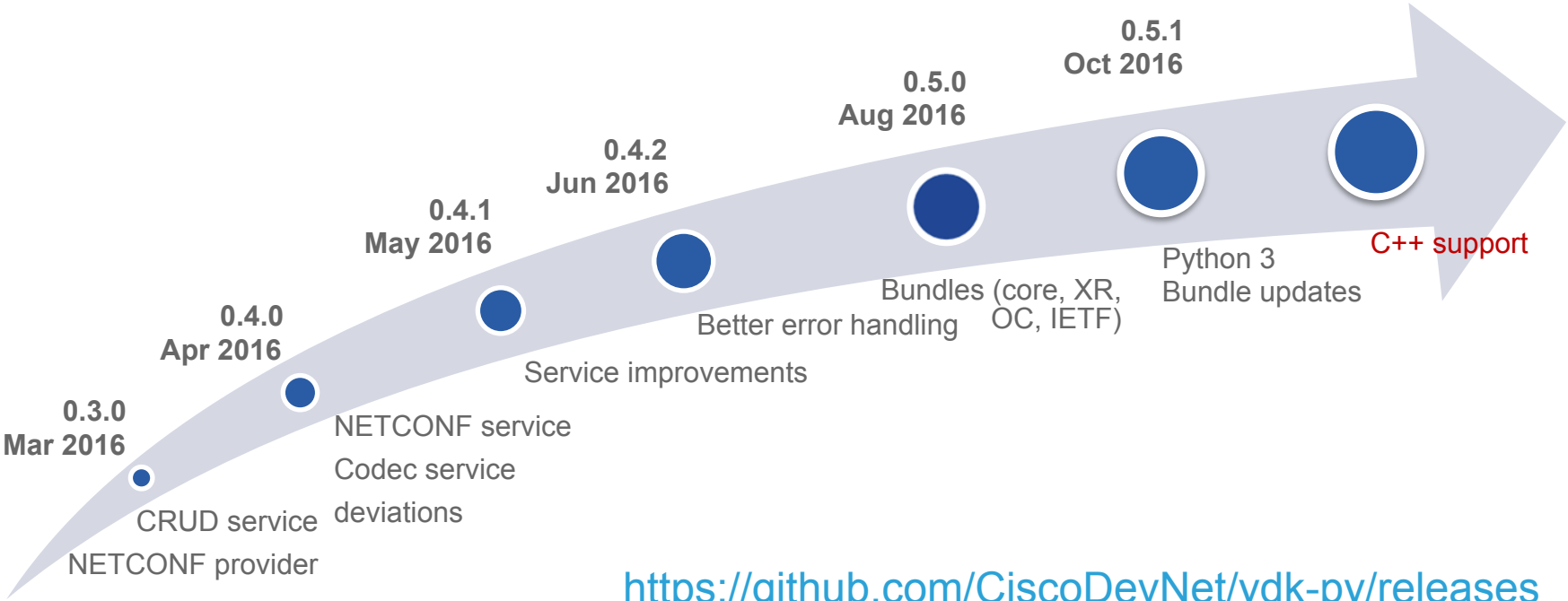
  `ydk.services`

  `ydk.services.CRUDService`

Logging Example

```
logger = logging.getLogger("ydk.services.CRUDService")  # get logger

logger.setLevel(logging.DEBUG)  # one of CRITICAL, ERROR, WARNING, INFO, DEBUG

handler = logging.StreamHandler()  # send log records to sys.stderr

# set record format to show time, severity and log message
formatter = logging.Formatter(("%(asctime)s - %(name)s - "
                               "%(levelname)s - %(message)s"))

handler.setFormatter(formatter)  # associate formatter with handler
logger.addHandler(handler)  # associate handler with logger
```

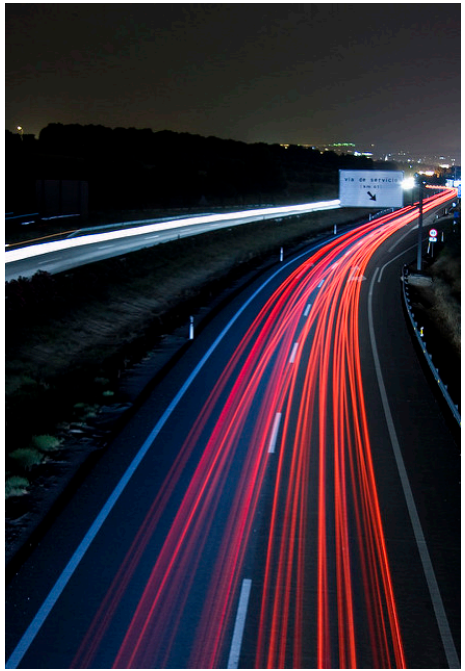
Releases



https://github.com/CiscoDevNet/ydk-py/releases

Future Work Items

- Path API
- Dynamic API generation
- Additional providers (Google RPC, RESTCONF)
- Telemetry support
- Additional services (e.g. validation)
- Additional languages (e.g. C++, Go, Ruby, Java)
- Additional encodings (e.g. JSON, GPB)
- Model-mapping support



Resources

GitHub

- YDK Python API (https://git.io/vaWsg)
- YDK-Py sample apps (https://git.io/vaw1U)
- YDK Generator (https://git.io/vaw1M)
- XR Docs (https://xrdocs.github.io/programmability/)
- Cisco IOS XR YANG models (https://git.io/vg7fk)
- YANG Explorer (https://git.io/vg7Jm)

DevNet

- YDK at DevNet (https://goo.gl/Wqwp3C)

@111pontes

Resources (cont.)

YDK Sandbox

- Ubuntu YDK-PY Vagrant box (https://git.io/vaw1U)

YDK Support

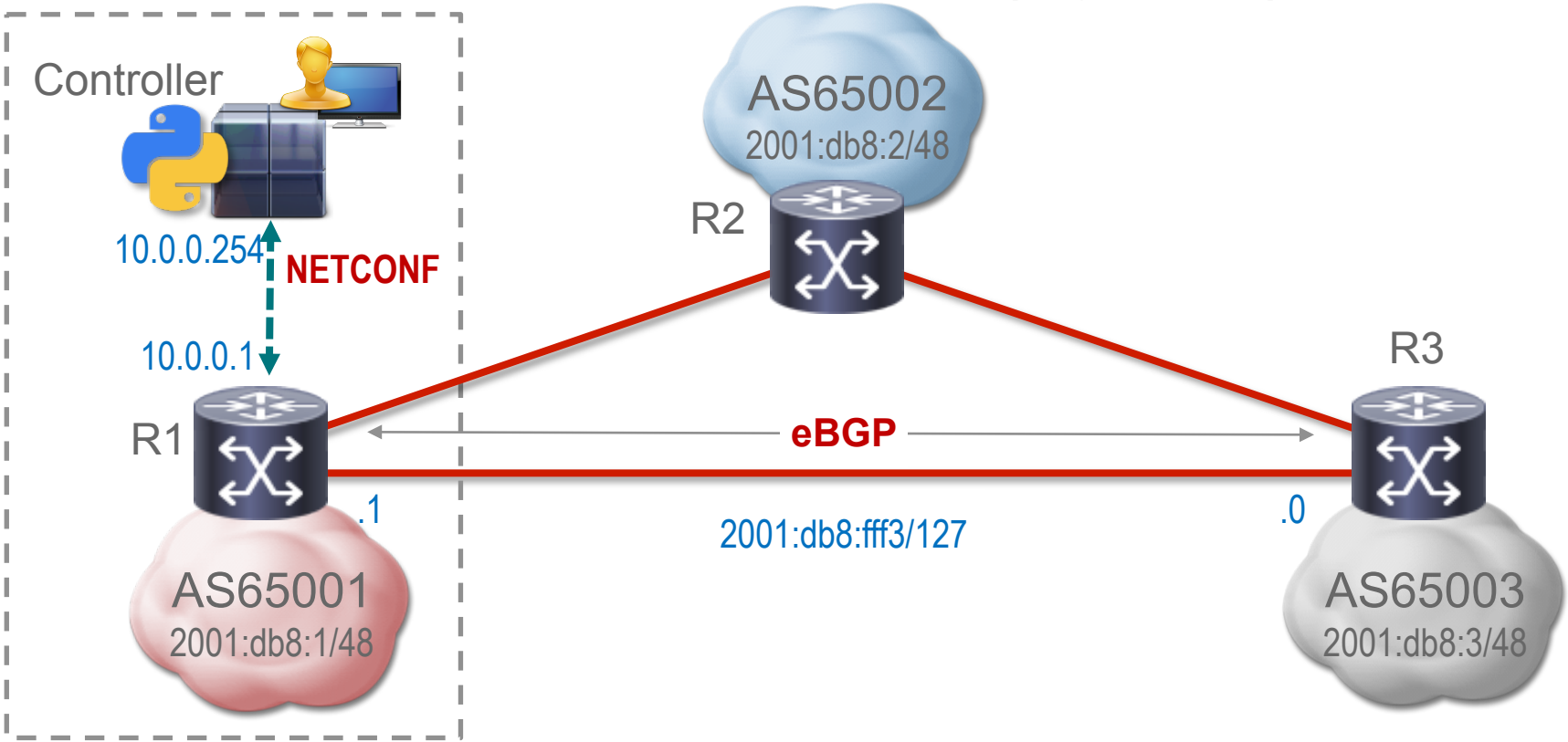
- Cisco support community (https://communities.cisco.com/community/developer/ydk)

Other

- Device programmability blog (http://blogs.cisco.com/author/santiagoalvarez)

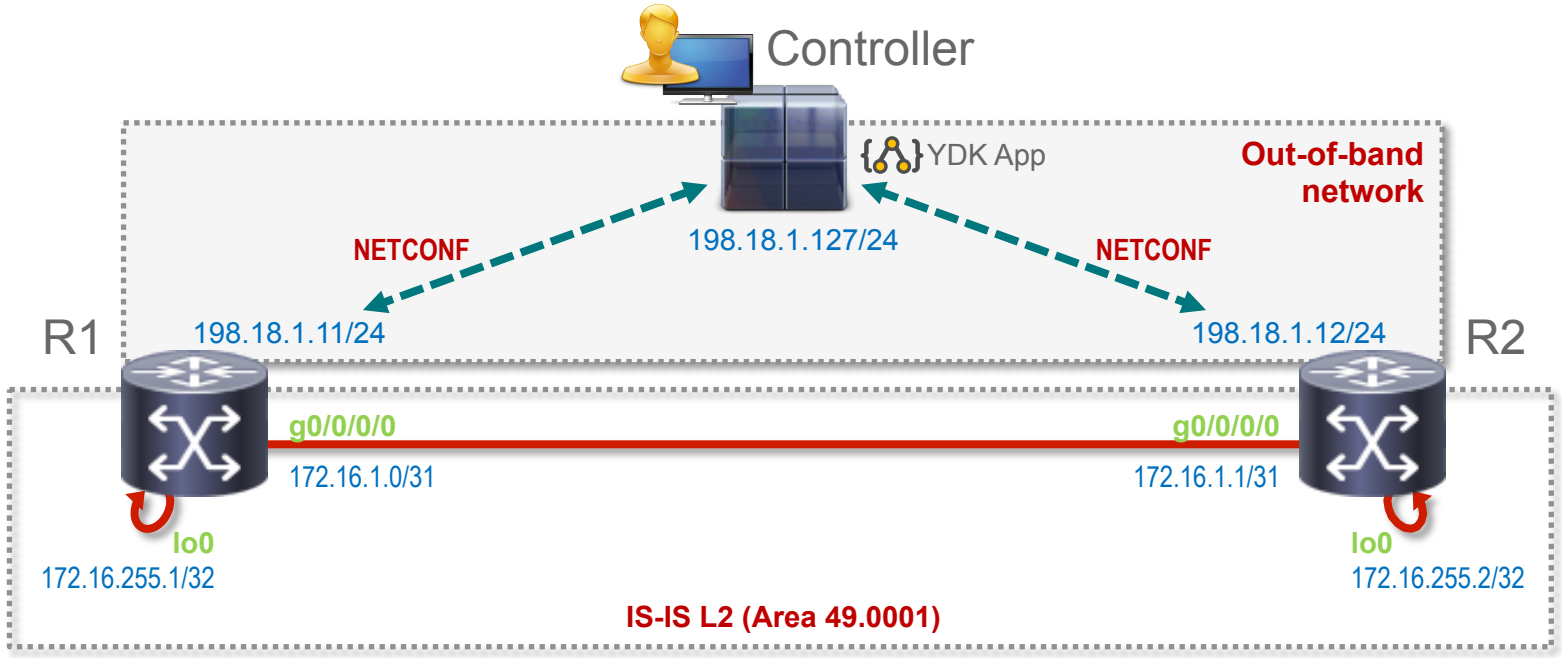
Demo

OC-BGP IPv4/IPv6 Unicast (Python)



Thanks

Testbed Topology



 saalvare@cisco.com

Backup

Motivations for Network Programmability

- Speed and scale demand software automation and data analytics
- Rapid innovation as competitive advantage
- One network operator per 1000s / 10000s of complex network devices

saalvare@cisco.com

TOMORROW starts here.