Cisco Network Service Orchestrator

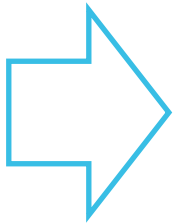
enabled by Tail-f technology

Carl Moberg <camoberg@cisco.com>
Technology Director, Cloud and Virtualization Group (CVG)

Agenda

- An introduction to the YANG data modeling language
- How NSO uses YANG data models to represent the syntax, structure and semantics of devices and services
- The moving parts of Network Equipment Drivers (NEDs) that provides the southbound multi-vendor, multi-protocol capabilities of NSO
- The moving parts of an examples service model for MPLS-VPN including the YANG module and the decomposition code in Java and templates

Current networks are challenging...



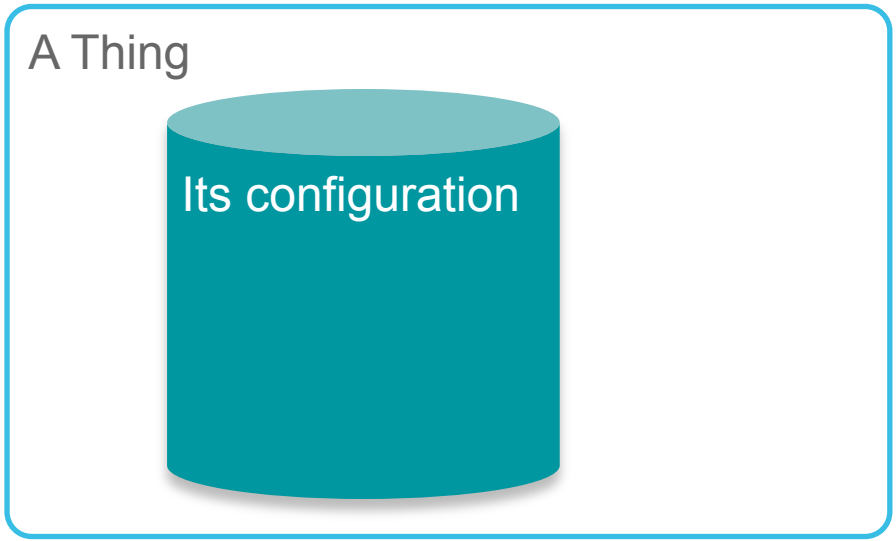
*Gartner: Telecom Operations Management Systems (BSS, OSS and SDP), Worldwide, 3Q13 Update

A look at some problem commonalities

- Vendors have interfaces – ways of interacting with their "thing"
  - Invented long time ago, no resources to refactor – also it is hard
  - Not really developed for integration – lock in-implementations!
  - Openness are commonly superficial – "that's how we do REST!"

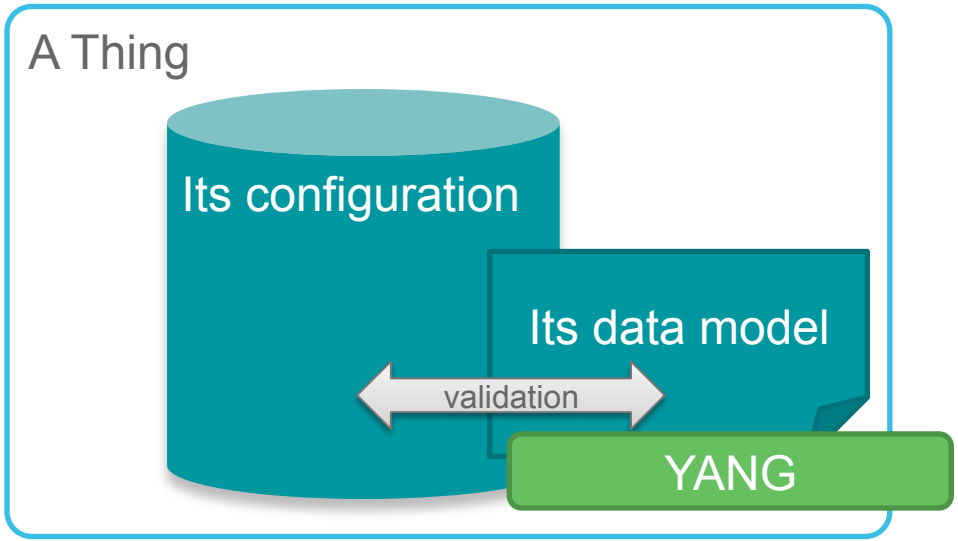
- One step deeper into details
  - Interfaces lack common structure, semantics – "that's how we define VLAN!"
  - Protocols and interfaces lack basic features – "what do you mean *rollback*?"
  - Disagreement on layerings – "You can only reach my devices through an EMS"

Configuration, Models and Protocols



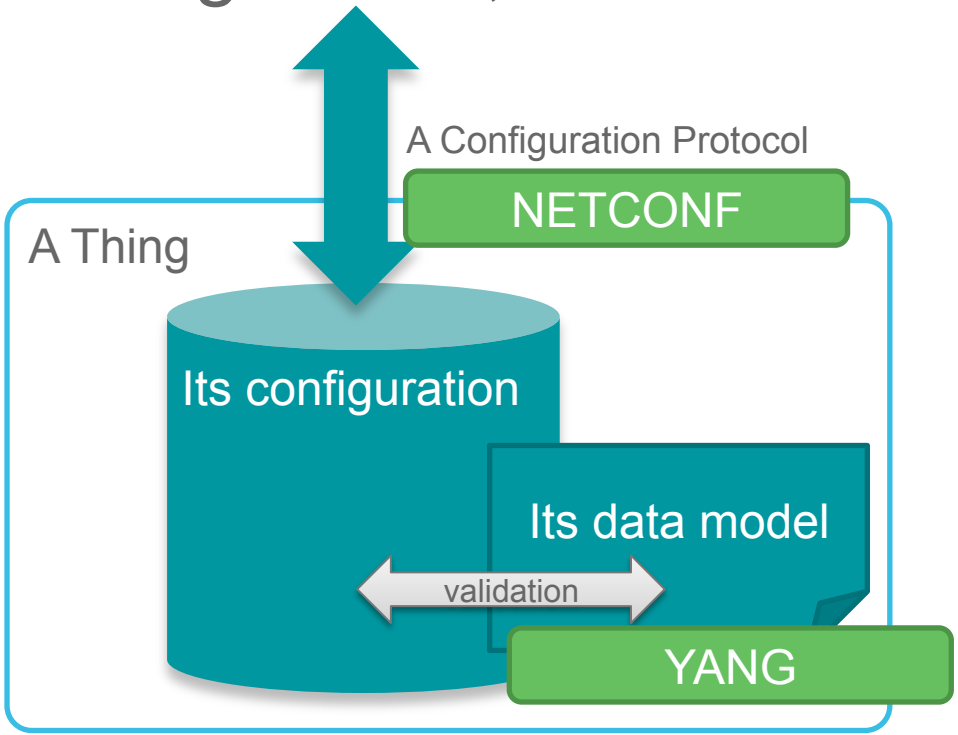
- Examples of A Thing includes:
  - Physical routers, switches
  - Virtual Networking Functions (VNFs)
  - Controller applications
- The *configuration* of the thing is the representation of it's intended state

Configuration, **Models** and Protocols



- Examples of *A Thing* includes:
  - Physical routers, switches
  - Virtual Networking Functions (VNFs)
  - Controller applications
- The *configuration* of the thing is the representation of it's intended state
- The *data model* is the set of constraints applied to the intended state through validation

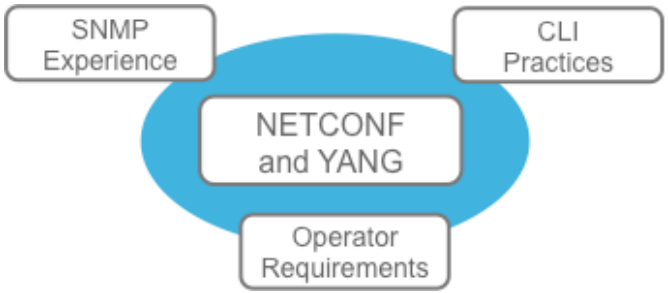
Configuration, Models and **Protocols**



- Examples of *A Thing* includes:
  - Physical routers, switches
  - Virtual Networking Functions (VNFs)
  - Controller applications
- The *configuration* of the thing is the representation of it's intended state
- The *data model* is the set of constraints applied to the intended state through validation
- The *protocol* is the means by which to manipulate valid configuration

Origins of NETCONF and YANG

- Several meetings at events in 2001 (NANOG-22, RIPE-40, LISA-XV, IETF 52)
  - Operators expressing opinion that the developments in IETF do not really address requirements configuration management.
- June of 2002, the Internet Architecture Board (IAB) held invitational workshop on Network Management (RFC3535) to
  - Identify a list of technologies relevant for network management with their strengths and weaknesses
  - Identify the most important operator needs.



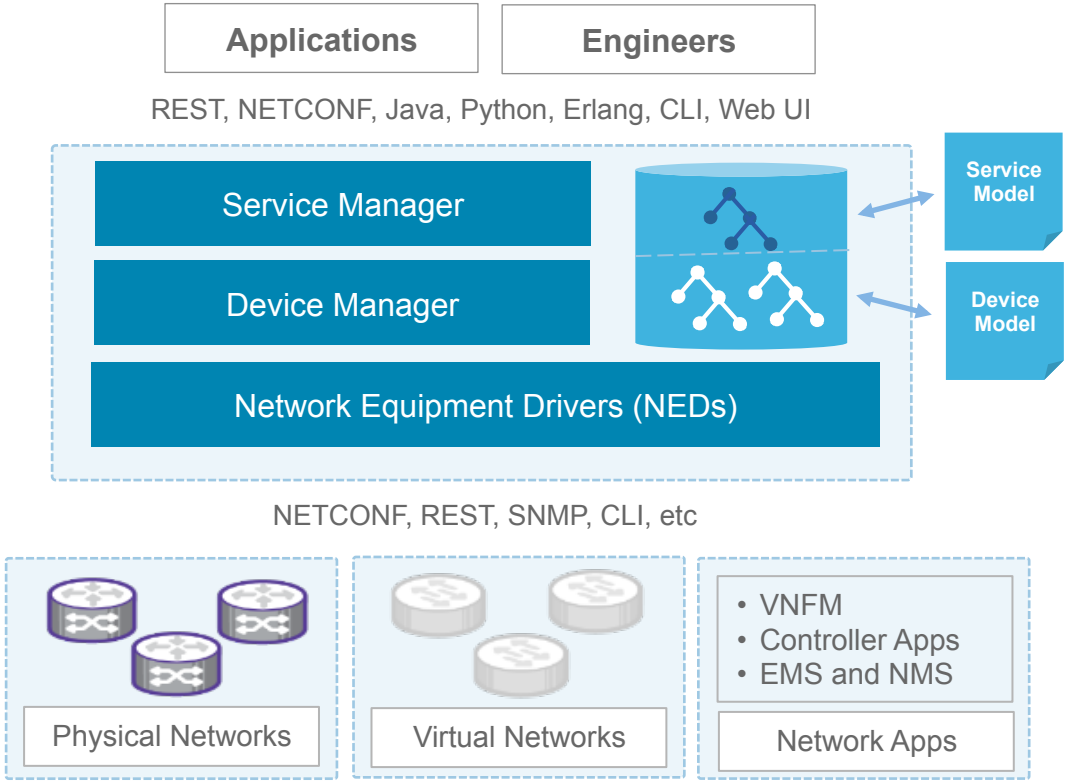
YANG – A Data Modeling Language for Networking

- Human readable and easy to learn
- Hierarchical configuration data models
- Reusable types and groupings (structured types)
- Extensibility through augmentation
- Formal constraints for configuration validation
- Data modularity through modules and sub-modules
- Well defined versioning rules

```
grouping mpls-global {
  description "global level definitions for MPLS protocol
  operation";
  list mpls-interfaces {
    key interface-name;
    description "interfaces for which MPLS is enabled";
    leaf interface-name {
      type string;
      description "reference to interface name";
      // TODO: add ref to interface model
    }
    leaf-list interface-admin-groups {
      type leafref {
        path "/mpls:mpls/mpls:global/mpls:mpls-admin-groups/"
        + "mpls:admin-group-name";
      }
      description
        "list of configured admin-groups on the interface";
    }
  }
  leaf mpls-lsp-install-delay {
    type uint16 {
      range 0..3600;
    }
```

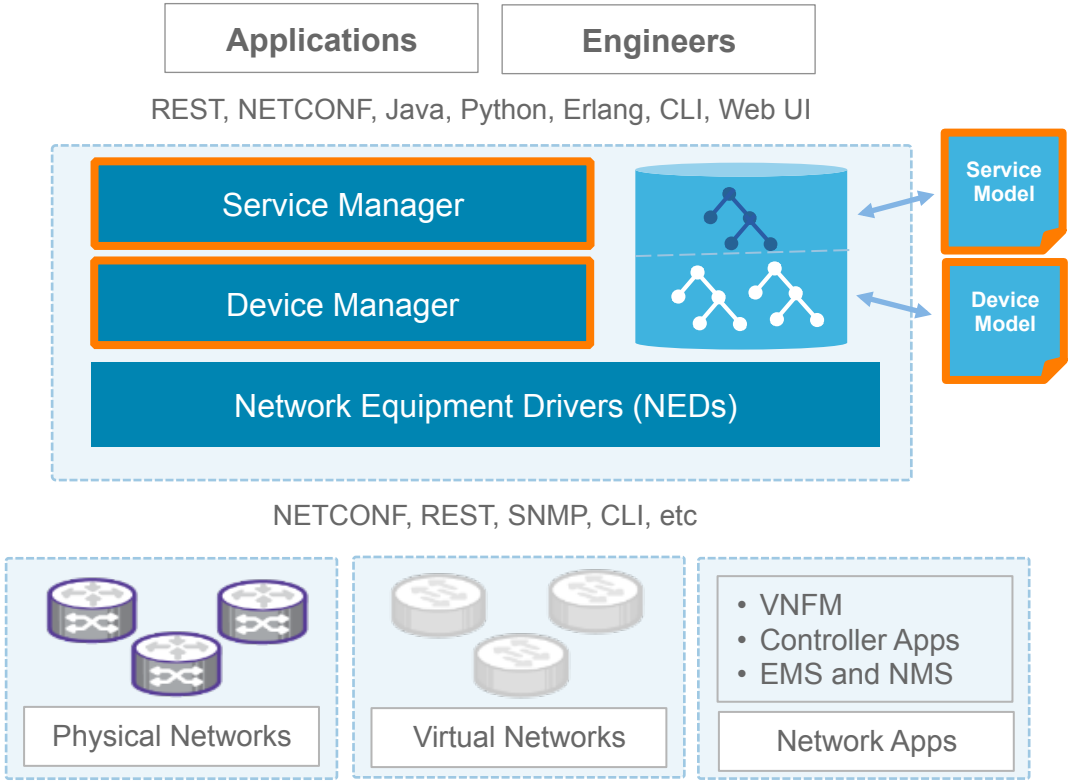
Why you should care: YANG is a full, formal contract language with rich syntax and semantics to build applications on

Introducing Network Service Orchestrator (NSO)



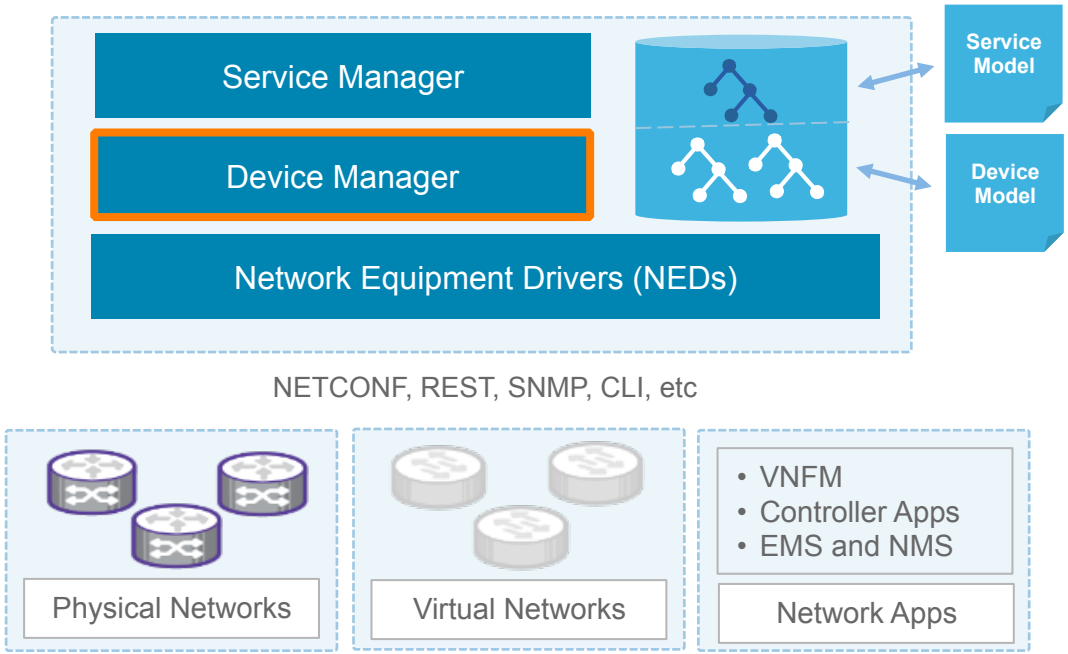
- Logically centralized network services
- Data models for data structures
- Structured representations of:
  - Service instances
  - Network configuration and state
- Mapping service operations to network configuration changes
- Transactional integrity
- Multiprotocol and multivendor support

NSO Main Features
#1 Model-based Architecture



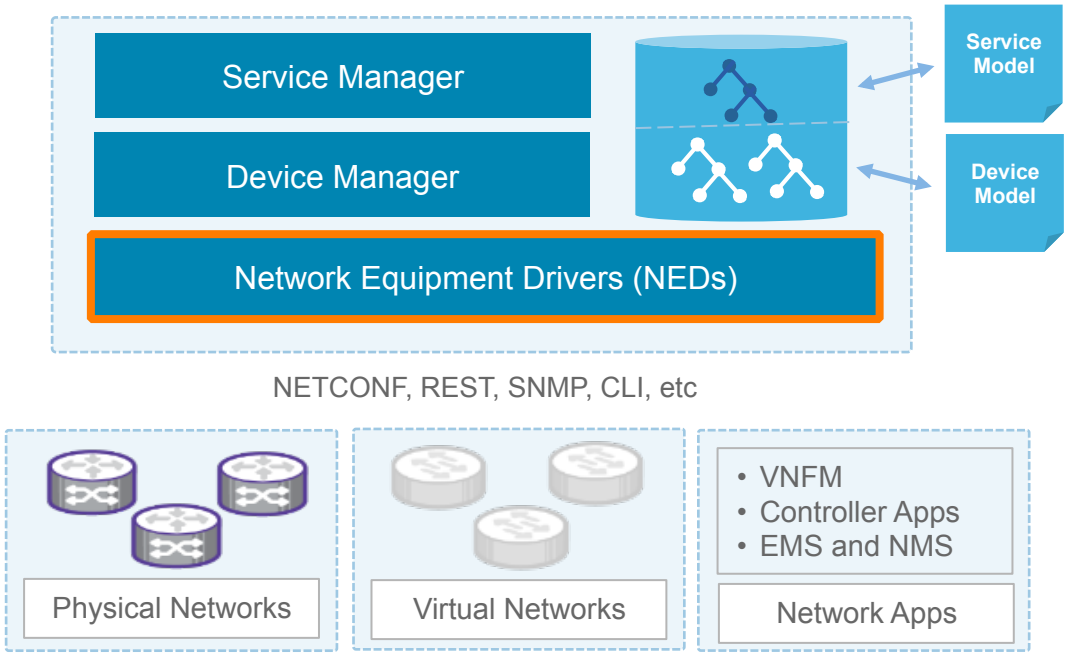
- No hard-coded assumptions about:
  - Network services
  - Network architecture
  - Network devices
- Instead:
  - Data models written in YANG (RFC 6020)

NSO Main Features
#2 Device Manager



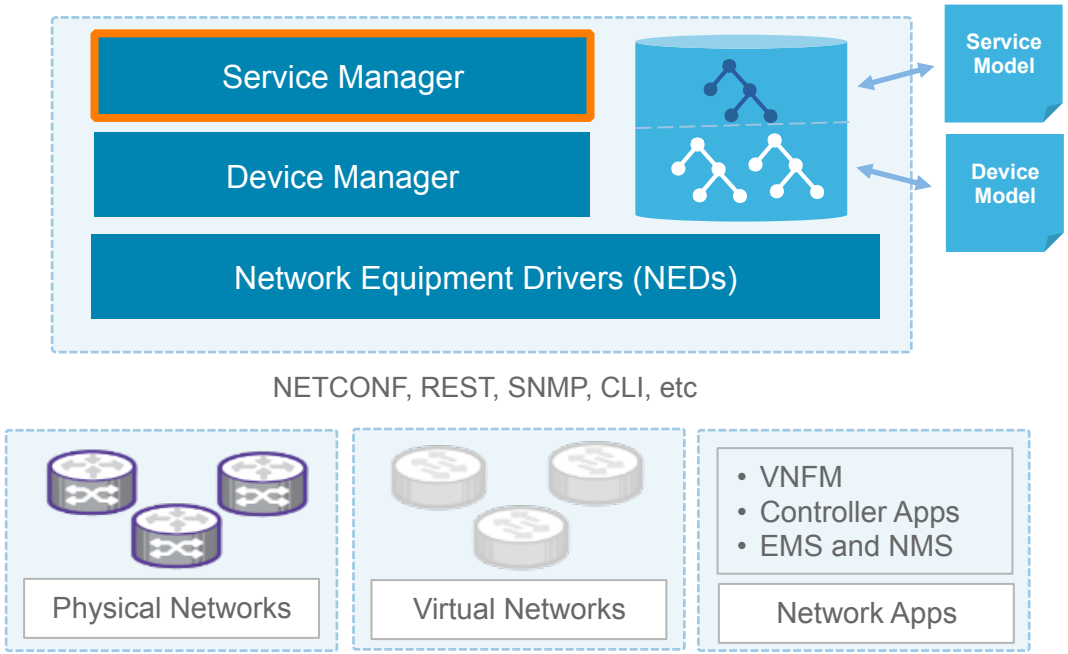
- Device model management
- Version and capability management
- Transactions and rollbacks
- Network configuration audit
- Configuration validation
- Policies and templates

NSO Main Features
#3 Network Equipment Drivers (NEDs)



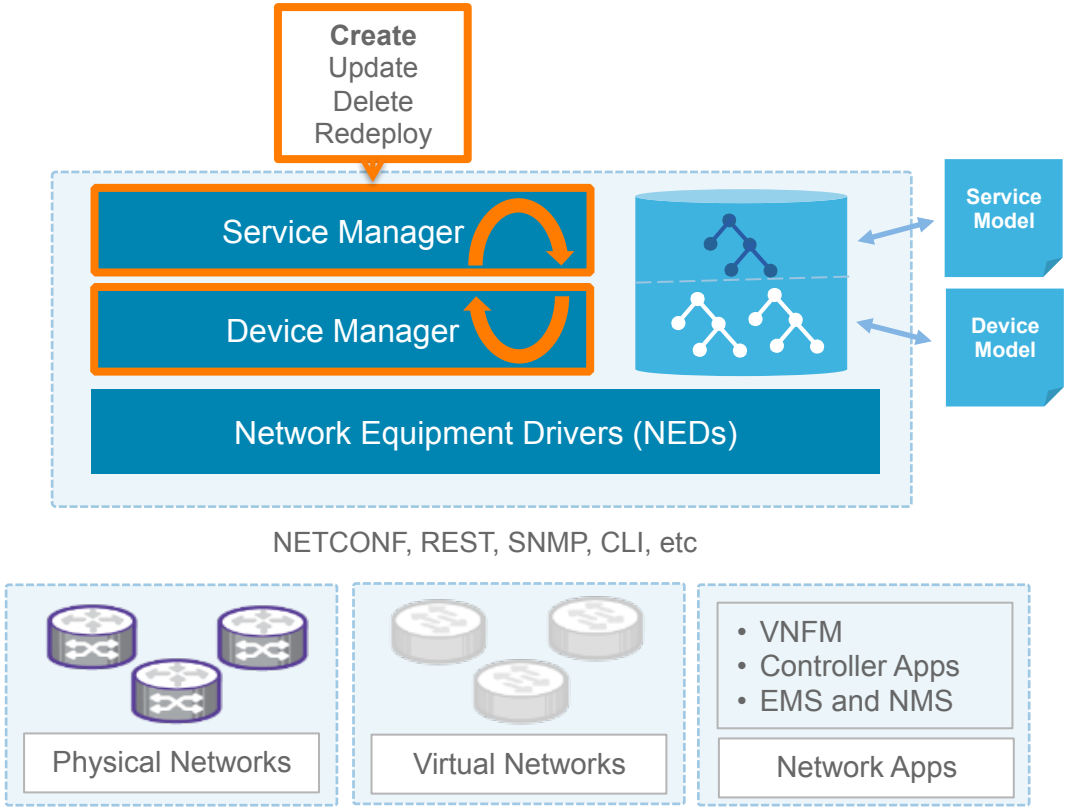
- Device model management
- Version and capability management
- Transactions and rollbacks
- Network configuration audit
- Configuration validation
- Policies and templates

NSO Main Features
#4 Service Manager



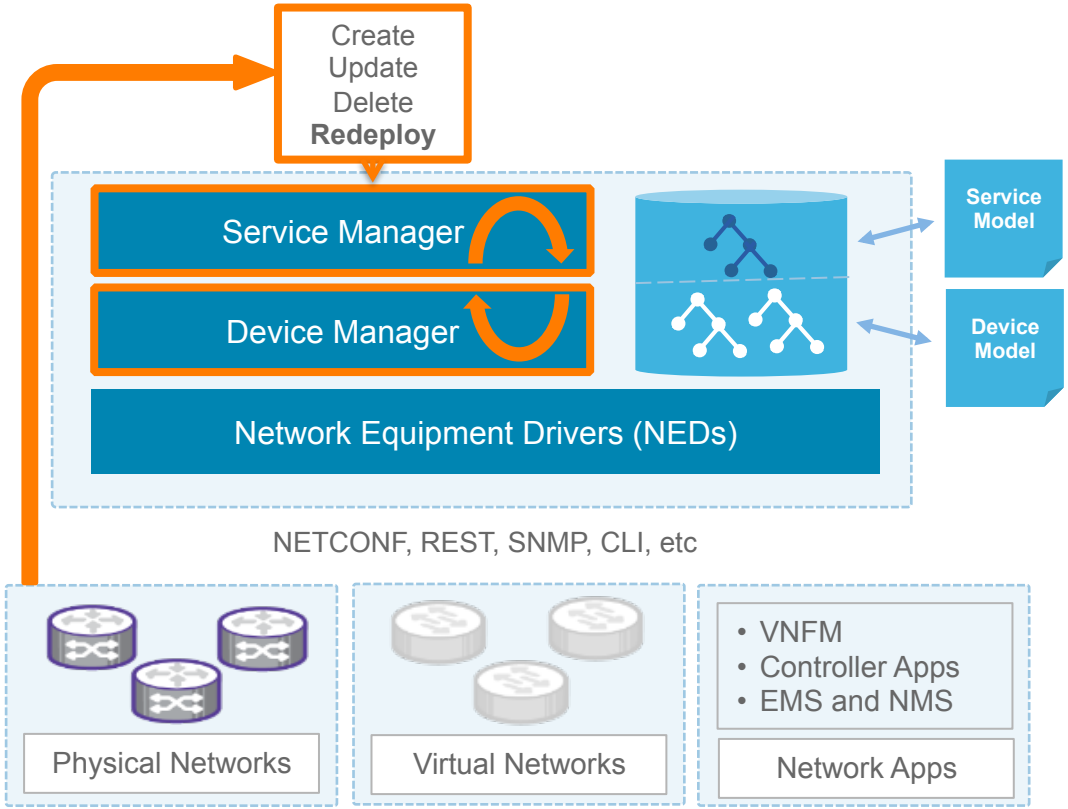
- Service model management
- Mapping to device model
- Device effects
- Service check-sync
- Service restoration
- Service testing
- Aggregated operational data

NSO Main Features
#5 Fastmap



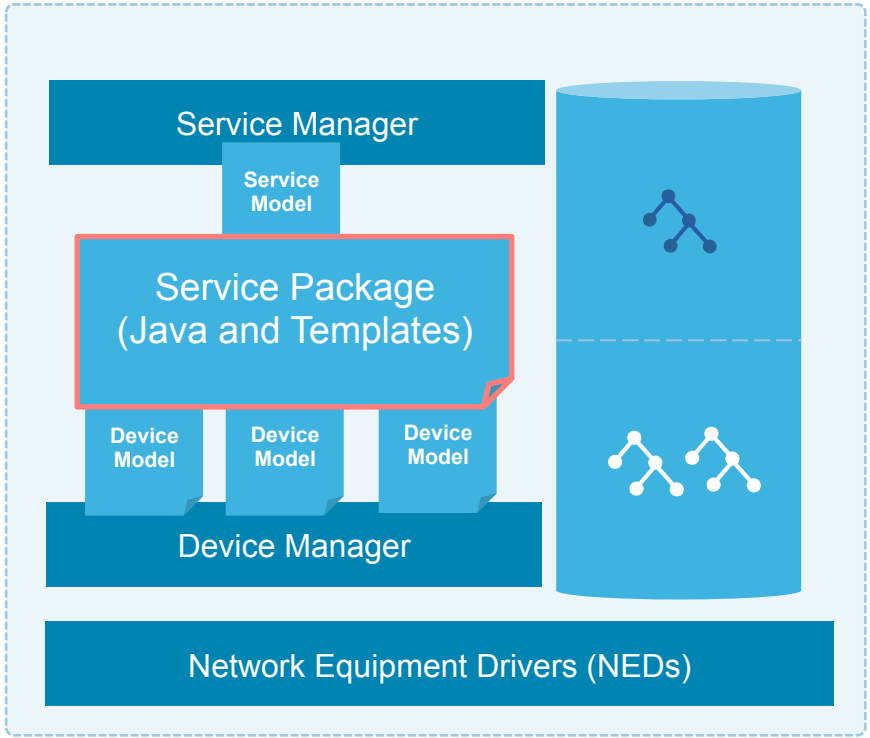
- FastMap:
  - Only the CREATE operation needs to be specified
  - UPDATE, DELETE and REDEPLOY automatic
- Benefits:
  - Reduces service implementation code by two orders of magnitude
  - Supports modifications of services at runtime

NSO Main Features (for Bonus Points)
#5 Reactive Fastmap (RFM)



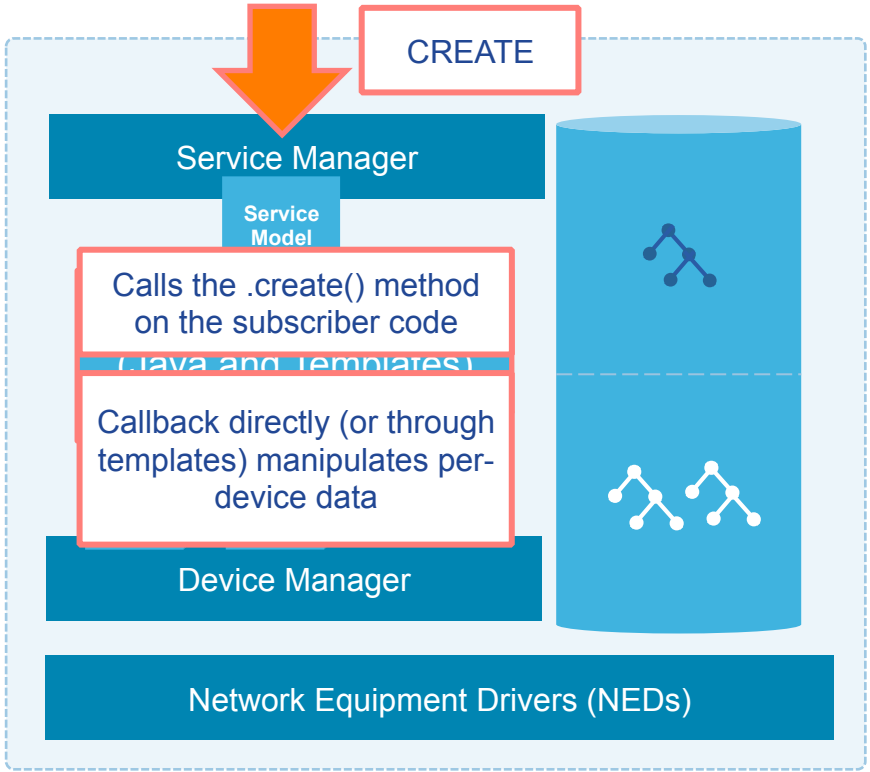
- Development pattern to:
  - Redeploy service configuration on operational changes
  - Idempotent
- One algorithm supporting:
  - Provisioning
  - Orchestration
  - Elasticity
  - Virtual machine and VNF mobility
  - Self-healing network

Homing in on the Demo!



- Service Packages
  - Subscribe to operations on service models
  - Populate the device manager with associated changes
  - Using Java and/or templates
- Service Decomposition
  - Focus only on normalized service-to-device mapping
  - All-declarative can be done with templates
  - Realistic use-cases use Java (Python on roadmap)

Homing in on the Demo!



- Service Packages
  - Subscribe to operations on service models
  - Populate the device manager with associated changes
  - Using Java and/or templates
- Service Decomposition
  - Focus only on normalized service-to-device mapping
  - All-declarative can be done with templates
  - Realistic use-cases use Java (Python on roadmap)

Demo Time!