The SUPA Information Model

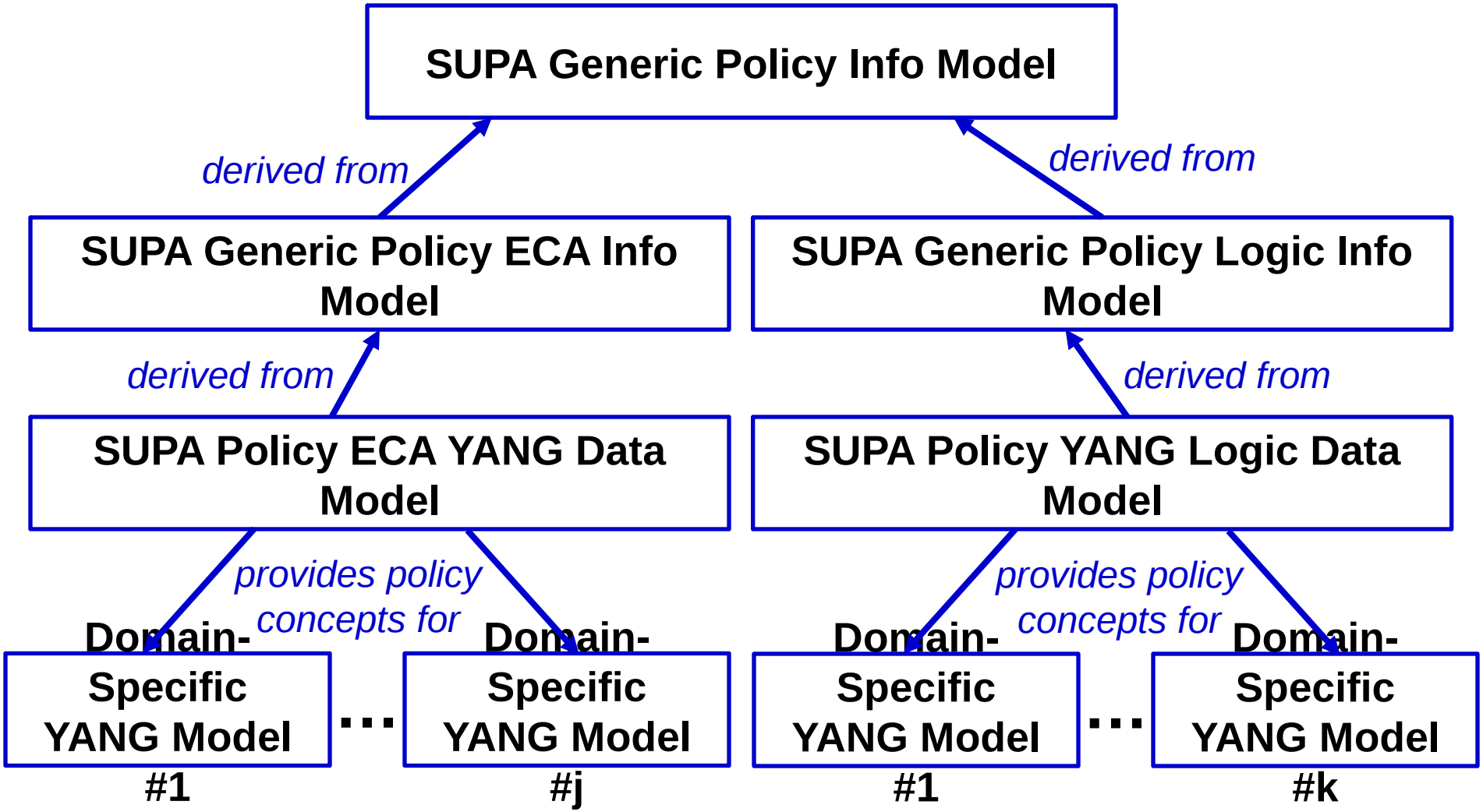
John Strassner, Huawei

On Behalf of the Info Model Design Team

Motivating Example

- **An operator wants to ensure that only certain traffic is allowed to a given application server**
- **This means that the operator has to potentially configure many different types of heterogeneous systems:**
  - Routers, switches, firewalls, load balancers, servers, …
- **This means that each of these systems have to be configured and monitored**
  - But each system has different languages and protocols for this
- ***A policy-based environment enables the operator to focus on what the environment should do as opposed to configuring each system and device individually***

SUPA Unifies Different Data Models



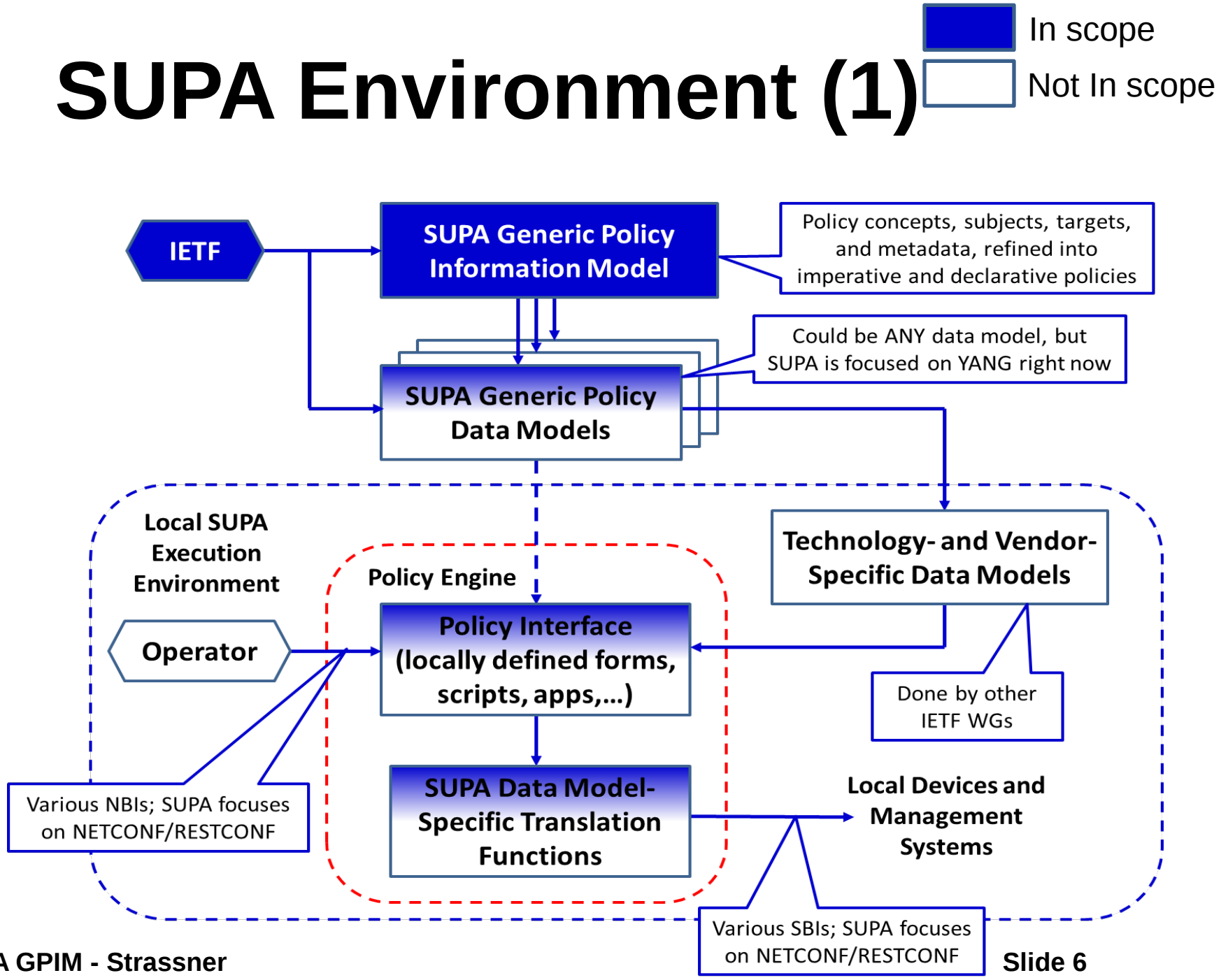
Why an Information Model?

- **Promotes Reusability and Interoperability**
  - Independent of repository, protocol, query and data definition languages, and implementation
  - Enables interoperation between management systems and managed elements
  - Builds a library of reusable objects (policies, policy components, and metadata)
- **Defines common vocabulary and concepts to govern heterogeneous devices and technologies**
  - The world is NOT “just YANG” (Puppet/Chef, storage and compute, etc.) …
  - … BUT we are focusing on just YANG and NETCONF/RESTCONF right now.
- **Enables different YANG data models for different applications to more easily interoperate with YANG and other data models**
  - Common reusable objects, policies, and metadata
- **Defines semantics that can guide implementations**

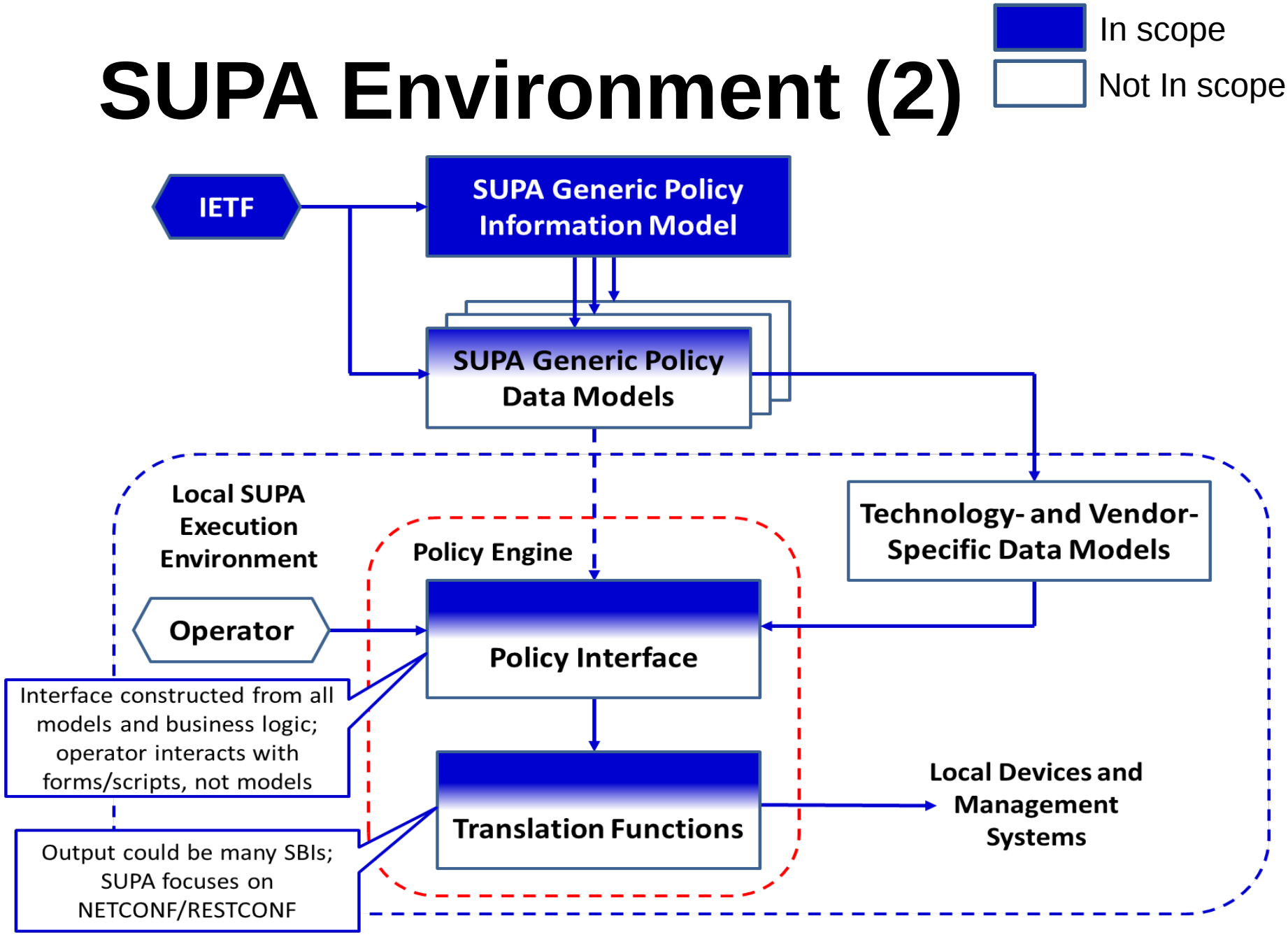
The Generic Policy Info Model

- **Three Sets of Information Models**
  - Generic model of policy concepts
    - Used to derive specific model of ECA rules, and
    - Used to derive specific model of logic statements
  - *Designed to fit into a larger info model, and/or generate data models that can be part of a larger system*
    - SUPA only models Policy; we will reuse resource and service models of other WGs
- **Languages are limited in their expressiveness**
  - Example: how to express “hard” vs. “soft” constraints
  - Better to start with a model first to guide the choice of language used
- **Supports multiple implementations**
  - Model can be translated to scripts, providing an extensible templating approach
  - The model’s software patterns can be used to dynamically change policies at runtime using machine-generated code without recompiling or redeploying

SUPA Environment (1)



SUPA Environment (2)



SUPA Usage Example

- **Service chaining is typically hard-wired**
  - If the condition is not matched, then no action is taken
  - Actions are typically hardwired
  - If different actions are required, then the logic emulates a switch statement and can require reprogramming
- **Policy makes this more robust**
  - The events triggering the service chain, along with its conditions and actions, are all abstracted, enabling scripts or code generation to be supported
  - This also makes each runtime substitutable, enabling each to be dynamically changed based on *context*
  - Declarative and ECA policies are both applicable, and can be used individually or in combination; also simplifies supplying parameters for scripts
- **Policy abstracts how configuration is sent to the device**
  - Input to Policy Engine (for network elements) can be logic, events, and network element attributes supplied using NETCONF or RESTCONF
  - Output from Policy Engine can be YANG, CLI, Puppet Manifests, AWS Cloud Formation Templates

Summary

- **Use the Info Model to define an extensible set of policy concepts and vocabulary**
  - Provide hooks for interoperating with heterogeneous data models, but focus solely on YANG with NETCONF/RESTCONF for now
  - Finish this work by the end of 2015
- **Use the Info Model to define generic YANG data models**
  - Currently, even simple condition-action statements lack standardization
  - Finish this work by Q1 2016
- **Use Cases are work from other WGs**
  - Ensure that work from other WGs is not impacted
  - Other WG work as use cases to prove policy can be applied generically
- **Rest of effort focused on interoperable implementations**
  - Work with open source consortia and other SDOs as necessary
  - Finish this work by the end of 2016

Questions?

“Create like a god. Command like a king. Work like a slave”

- Constantin Brancusi

Why Is Policy Needed?

- **Policies are used by multiple actors**
  - App developers, operators, security and compliance teams, administrators, end-users, … **each has different concepts and terms**
- **Policies focused on different technologies must be able to work collaboratively**
  - Requires a **common set of concepts and vocabulary** across domains
- **Policies exist at different levels of abstraction**
  - Per-port, -device, -network, -VM, -application, -service, …
- **Different Policies exist for different operations on the same device**
  - Monitoring vs. configuration vs. audit
  - Deployment vs. backup vs. provisioning vs. billing vs. retirement …
- ***Policies help heterogeneous systems interoperate***
- ***Plan for the future, but focus just on YANG and NETCONF/RESTCONF***

Terminology

- **Information Model**
  - A representation of managed objects and their relationships that is independent of data repository, language, and protocol
- **Data Model**
  - A representation of managed objects and their relationships that is dependent on data repository, language, and/or protocol (typically all three)
- **Policy Rule**
  - A mechanism to manage the changing and/or maintaining of the state of one or more managed objects
- **ECA Policy Rule**
  - A type of policy that is triggered by a set of events. This in turn causes its set of conditions to be evaluated; if true, then its set of actions may be executed
- **Declarative Policy**
  - A type of policy that defines a goal to be achieved, but not how to achieve the goal

How SUPA Uses Terminology

- **Information Model**
  - Enables heterogeneous environment to be managed in a unified manner (e.g., common semantics can be described by policy and policy components)
- **Data Model**
  - Focus is currently on YANG
  - Enables other data models (SNMP, CLI, AWS, Puppet/Chef/Ansible/Salt, …) to be integrated with network-based YANG models
- **Policy Rule**
  - Standardizes semantics for policy-based management
- **ECA Policy Rule**
  - Standardizes WHEN <event> IF <condition> THEN <action> semantics
- **Declarative Policy**
  - Standardizes declaratively asserting facts to achieve a set of goals

Four Different ECA Policy Examples

- **draft-ietf-netmod-acl-model-02 (uses ‘matches’ and ‘actions’ lists)**
  - Defines filtering on source & dest port range, DSCP, protocol, IP version, and MAC address
  - Defines permit and deny packet handling action
- **draft-hares-i2rs-bnp-eca-data-model-00 (uses ‘rule group’ and ‘rule’ leaf-lists, and ‘rule-match-act’ list containing ‘bnp-matches’ and ‘bnp-action’)**
  - Defines filtering on interface, L1-L4 header, packet size, or service header
  - Defines L1-L4 actions, service actions, or forwarding on interface, next hop, route attributes, or RIB route attributes
- **draft-dunbar-i2rs-discover-traffic-rules-00 (uses RBNF)**
  - Defines filtering on L2-L4 header, VLAN, VNID, service chain ID, size, event, …
  - Defines egress port specific actions including adding VLANID tags, removing service header fields, forwarding traffic out of a particular interface or tunnel, …
- **draft-shaikh-rtgwg-policy-model-00 (uses policy-definition’ leaf-lists with ‘conditions’ and ‘actions’ presence containers)**
  - Defines filtering on how a route was installed, neighbor set, BGP-specific parameters, …
  - Defines accept & reject route and IGP actions

Technical Overview

- SUPAPolicy can be an individual policy or a set of policies
- A SUPAPolicy can be a SUPAECAPolicyRule or a SUPALogicStatement
  - SUPAECAPolicyRule contains 3 clauses in 1 statement
  - SUPALogicStatement contains 1 or more statements
  - They may be combined
  - Both SUPAECAPolicyRule and SUPALogicStatement MUST have at least 1 SUPAPolicyStatement
- SUPAPolicyStatement can be made up of any combination of its subclasses
  - SUPAEncodedClause is an encoded SUPAPolicyStatement (or a portion of one)
  - SUPABooleanClause is an individual or a set of Boolean clauses in CNF or DNF
  - SUPALogicClause is a set of one or more logic clauses in propositional logic or first order logic
- SUPAPolicyStatement can be constructed from a SUPAPolicyTerm and/or a SUPAECAComponent
  - SUPAPolicyTerms are a {variable, operator, value} tuple
  - SUPAECAComponents can be used individually or as a group, and define events, conditions, and actions
    - Each SUPAECAComponent can be constructed from a SUPAPolicyTerm or from dedicated objects
- An optional set of GPIM SUPAPolicySubjects can be defined to represent the authoring of a SUPAPolicyRule
- An optional set of GPIM SUPAPolicyTargets can be defined to represent the set of managed entities that will be affected by this SUPAECAPolicyRule
- An optional set of SUPAPolicyMetadata can be defined for any of the objects that make up a