

SUPA
Internet-Draft
Intended status: Standards Track
Expires: June 11, 2016

T. Zhou
Y. Xia
Huawei Technologies Co., Ltd
B. Wijnen, Ed.
Consultant
December 9, 2015

A YANG Data Model for Declarative Policy
draft-bw-supa-declarative-policy-data-model-00

Abstract

This document describes a base YANG data model for declarative policies that can be used within the SUPA (Simplified Use of Policy Abstractions) framework. The base model can be augmented by additional YANG modules defining data models for intent related policies and functions to support various network scenarios and applications. The base declarative policy data model provides common building blocks for extensions.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 11, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Requirements Language and Terminology	3
3.	Design of The Declarative Policy Module	3
3.1.	Design of The Operation	5
3.2.	Design of The Result	7
4.	Declarative Policy Base YANG Data Model	9
4.1.	Declarative Policy Data Model Hierarchy	9
4.2.	Declarative Policy Base Data Model in YANG Module	12
5.	Declarative Policy Data Model Examples	31
5.1.	Link utilization optimization	31
5.2.	Policy Based Bandwidth on Demand	32
5.3.	Service function chaining	33
6.	Security Considerations	33
7.	IANA Considerations	33
8.	Acknowledgements	33
9.	Informative References	34
	Authors' Addresses	35

[1.](#) Introduction

Cloud computing and Software Defined Networking (SDN) are moving the IT world from a network-centric view to an application-centric view. Intent based North Bound Interface (NBI) provides applications the ability signal the "intent" (E.g. create a network between 3 applications) to the network layer rather than specify the details of the network. The network intent is declarative ("go to the store") rather than imperative ("follow this route to the store"), leaving the details to the network implementation.

In the SUPA proposition [[I-D.klyus-supa-proposition](#)], the declarative policy is introduced as a higher level abstraction for service management. The NEMO language specifications [[I-D.xia-sdnrg-nemo-language](#)] describe a set of declarative primitives to manipulate and manage virtual networks, where the declarative policy model is a very important part. The declarative policy base model is also the shared part that can be used in working

groups within IETF and also in other organizations. It provides a consistent abstraction for a higher level policy based network management system while concealing various implementation techniques and multi-vendor devices.

This document introduces YANG [[RFC6020](#)][RFC6021] data models for the declarative policy. This set of models facilitates the standardization for a more abstracted policy configuration interface for SUPA. The basic model can be augmented by additional YANG modules defining data models for intent related protocols and functions to support various different network scenarios and applications. The base declarative policy data model provides common building blocks for extensions, such as specific operation and result.

[2.](#) Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [[RFC2119](#)] key words.

[3.](#) Design of The Declarative Policy Module

Declarative policies that can apply to services and/or resources describe what to do, but not how to do it. This section gives an overview of the declarative policy module as shown in Figure 1

Similar to other kinds of policies [[I-D.chen-sup-a-eca-data-model](#)], there will be a policy header which contains general policy information such as:

- o policy-id: is the globally unique identifier for a policy instance.
- o policy-name: is an easy to recognize and refer sting for policy instance naming.

- o `policy-priority`: indicates the priority of the policy instance. Higher priority policy instance will get the prior order to execute.
- o `policy-validity-period`: controls the life cycle of the policy instance with the start/end time or period.
- o `policy-rule-deploy-status`: describes the deployment status of the policy rule.
- o `policy-rule-exec-status`: describes the execution status of the policy rule.

The declarative policy expression need to contain a target-object, such as a VPN, a flow, a link and so on, to which the policy instance will apply. The target-object also defines the governance scope of the policy application. The target-object is a data model definition to be provided by other WGs in a separate module and is not part of the declarative policy module itself. For example, in the policy, guarantee the the utilization of a link below a threshold, the dedicated link instance modeled by the policy user must be described in certain place. So that the policy model can refer the target object with an object-id.

The policy can also apply to a collection of objects, e.g., a group of links with the same policy requirements. In this case, the collection as a whole should be considered as an object with an unique object-id.

The intent of the policy is described with a list of policy rules which include declarative statements. In general, there are two ways to express the intent without touching the implementation details.

One type of declarative statement is to express key operations that the user wants to execute, with the semantic "I want to do some thing". We use the operation item to describe this kind of statement. The underlying policy resolver engine can extend the operation and generate a complete action list from the requests. The translation and the fulfillment is implementation related but user agnostic.

The other type of declarative statement is to express the final

result or desired state, with the semantic "I want to achieve a certain state". In this case, a user may only need to describe the result without defining how to realize it. For example, require secure communication between two sites. The policy engine will monitor the required state, and do the adjustment automatically with pre-defined logic.

```

module: supa-declarative-policy
| ...
+--rw declarative-policy
|   +--rw policy-id                policy-id
|   +--rw policy-name              policy-name
|   +--rw priority?                uint8
|   +--rw policy-validity-period
|   |   +--rw start?               yang:date-and-time
|   |   +--rw end?                 yang:date-and-time
|   |   +--rw duration?            uint32
|   |   +--rw periodicity?         enumeration
|   +--rw policy-rule-deploy-status? enumeration
|   +--rw policy-rule-exec-status? enumeration
|   +--rw target-object             object-id
|   +--rw declarative-policy-rule
|   |   +--rw operation
|   |   |   | ...
|   |   +--rw result
|   |   |   | ...

```

Figure 1: Figure 1 Top level design of the base declarative policy data model

[3.1.](#) Design of The Operation

Operation refers to some specific actions in order to achieve the goal. An operation must contain one or more actions. We can use the semantic "on condition, do action, with constraint". However, condition and constraint can be optionally defined in operations. It depends on specific scenario and users' requirement. Once a condition is involved in operation, actions will not be executed until condition is true. For example, "do load balancing when the utilization of a link exceeds 80%". In this example, "utilization of a link > 80%" is the trigger, and "do load balancing" is the action.

Condition must be modeled on top of events. In contrast to an event in Event-Condition-Action (ECA) policy model, a condition is state-based and therefore temporal constrained. It is triggered by some initializing event and ended by another event or after a certain time span.

In the declarative statement, an action will not be detailed configurations in devices, but high-level and abstracted functions which can be later translated into configurations by the policy engine. For example, the action "do load balancing" is device independent, and may configure a load balancer pool depending on specific devices.

Constraint restricts action itself or the scope of action.

The Condition-Action-Constraint (CAC) model used in operation expression is essentially different from the ECA policy model. Firstly, the event and condition in ECA model quite depend on the underlying implementation, while the declarative operation only care about the condition to execute the action. Secondly, the ECA model does not have a constraint description, while the declarative operation does not care about the implementation of the action but does care about the constraint to the action.

```
+--rw condition-parameter-definitions
|   +--rw condition-parameter-definition* [parameter-name]
|       +--rw parameter-name                parameter-name
|       +--rw parameter-value-type?          enumeration
```

```

|      +---rw parameter-match-patterns
|      +---rw parameter-match-pattern* enumeration
+---rw constraint-parameter-definitions
|  +---rw constraint-parameter-definition* [parameter-name]
|      +---rw parameter-name parameter-name
|      +---rw parameter-value-type? enumeration
|      +---rw parameter-match-patterns
|      +---rw parameter-match-pattern* enumeration
+---rw action-definitions
|  +---rw action-definition* [action-name]
|      +---rw action-name action-name
|      +---rw parameter-value-type? enumeration
+---rw declarative-policy
|  ...
+---rw declarative-policy-rule
|  ...
+---rw operation
+---rw condition
|  +---rw condition-segment* [condition-segment-id]
|      +---rw condition-segment-id condition-segment-id
|      +---rw condition-para-name? condition-parameter-name
|      +---rw condition-para-match-pattern? enumeration
|      +---rw condition-para-target-value
|          +---rw string-value? string
|          +---rw int-value? int64
|          +---rw range-value
|              +---rw min int64
|              +---rw max int64
|      +---rw precursor-relation-operator? enumeration
|      +---rw order? uint32
+---rw action* [action-name]
|  +---rw action-name action-name
|  +---rw parameter-values

```

```

|  |  +---rw string-value* [value order]
|  |  |  +---rw value string
|  |  |  +---rw order uint32
|  |  +---rw int-value* [value order]
|  |  |  +---rw value int64
|  |  |  +---rw order uint32
|  |  +---rw range-value
|  |      +---rw min int64

```

```

| |      +---rw max                int64
| +---rw order?                    uint32
+---rw constraint
    +---rw constraint-segment* [constraint-segment-id]
        +---rw constraint-segment-id        constraint-segment-id
        +---rw constraint-para-name?         constraint-parameter-name
        +---rw constraint-para-match-pattern? enumeration
        +---rw constraint-para-target-value
            | +---rw string-value?           string
            | +---rw int-value?              int64
            | +---rw range-value
            |     +---rw min                  int64
            |     +---rw max                  int64
        +---rw precursor-relation-operator? enumeration
        +---rw order?                      uint32

```

Figure 2 The snippet of operation

3.2. Design of The Result

The result statement describes the final state of objects without caring about how to achieve it. For example, a result could be that the company accesses any sites on the Internet safely. It just defines a result that ignores technology details, such as, firewall, ACL, and so on.

Result refers to desired state which is expected or ito be avoided. For example, a user may express an intent that the link utilization must be lower than 80%, which describes an expected state. The same intent can also be expressed as the link utilization should not exceed 80%, which describes an state to avoid. The result expression model should be flexible for users so they can describe iit in either way.

As shown in Figure 2, the result can be described as an expression with left, right operands and standard relational operators.

```

+--rw left-value
|   +--rw (value-type)?
|       +--:(string)
|           |   +--rw single-string-value?           string
|       +--:(string-list)
|           |   +--rw string-value* [value order]
|               |   +--rw value                       string
|               |   +--rw order                       uint32
|       +--:(int)
|           |   +--rw single-int-value?               int64
|       +--:(int-list)
|           |   +--rw int-value* [value order]
|               |   +--rw value                       int64
|               |   +--rw order                       uint32
|       +--:(variable)
|           |   +--rw variable-name?                 string
|       +--:(calculation-expression)
|           |   +--rw calculation-operator?           enumeration
|           |   +--rw calculation-leaf-value?         string
|           |   +--rw calculation-right-value?        string
+--rw relational-operator
|   +--rw relational-operator?                       enumeration
+--rw right-value
    +--rw (value-type)?
        +--:(string)
            |   +--rw single-string-value?           string
        +--:(string-list)
            |   +--rw string-value* [value order]
                |   +--rw value                       string
                |   +--rw order                       uint32
        +--:(int)
            |   +--rw single-int-value?               int64
        +--:(int-list)
            |   +--rw int-value* [value order]
                |   +--rw value                       int64
                |   +--rw order                       uint32
        +--:(variable)
            |   +--rw variable-name?                 string
        +--:(calculation-expression)
            |   +--rw calculation-operator?           enumeration
            |   +--rw calculation-leaf-value?         string
            |   +--rw calculation-right-value?        string

```

Figure 2: Figure 3 The snippet of result

The relational operator is an enumeration type including at least:

eq: Checks if the values of two operands are equal or not, if yes then the result of the expression becomes true.

ne: Checks if the values of two operands are equal or not, if values are not equal then the result of the expression becomes true.

gt: Checks if the value of left operand is greater than the value of right operand, if yes then the result of the expression becomes true.

ge: Checks if the value of left operand is greater than or equal to the value of right operand, if yes then the result of the expression becomes true.

lt: Checks if the value of left operand is less than the value of right operand, if yes then the result of the expression becomes true.

le: Checks if the value of left operand is less than or equal to the value of right operand, if yes then the result of the expression becomes true.

bl: Checks if the left set belongs to the right set, if yes then the result of the expression becomes true.

nb: Checks if the left set belongs to the right set, if no then the result of the expression becomes true.

The left and right operands can be a set or a single value or the expression with calculation operators. For example, a policy rule could be, the marketing network and the sales network do not belong to the internal network. Typical calculation operators include: add/minus/multiply/division, max/min/average, intersection/union/complement, and/or/not, any/all, etc.

[4.](#) Declarative Policy Base YANG Data Model

[4.1.](#) Declarative Policy Data Model Hierarchy

Figure 3 shows the structure of base data model for the declarative policy module.

```
module: supa-declarative-policy
  +--rw condition-parameter-definitions
  |   +--rw condition-parameter-definition* [parameter-name]
```

	+-rw parameter-name	parameter-name
	+-rw parameter-value-type?	enumeration

```

|      +-rw parameter-match-patterns
|      +-rw parameter-match-pattern*  enumeration
+-rw constraint-parameter-definitions
|  +-rw constraint-parameter-definition* [parameter-name]
|      +-rw parameter-name            parameter-name
|      +-rw parameter-value-type?     enumeration
|      +-rw parameter-match-patterns
|      +-rw parameter-match-pattern*  enumeration
+-rw action-definitions
|  +-rw action-definition* [action-name]
|      +-rw action-name                action-name
|      +-rw parameter-value-type?     enumeration
+-rw declarative-policy
|  +-rw policy-id                      policy-id
|  +-rw policy-name                    policy-name
|  +-rw priority?                      uint8
|  +-rw policy-validity-period
|  |  +-rw start?                      yang:date-and-time
|  |  +-rw end?                        yang:date-and-time
|  |  +-rw duration?                  uint32
|  |  +-rw periodicity?               enumeration
+-rw policy-rule-deploy-status?        enumeration
+-rw policy-rule-exec-status?          enumeration
+-rw target-object                     object-id
+-rw declarative-policy-rule
|  +-rw operation
|  |  +-rw condition
|  |  |  +-rw condition-segment* [condition-segment-id]
|  |  |  |  +-rw condition-segment-id  condition-segment-id
|  |  |  |  +-rw condition-para-name?  condition-parameter-name
|  |  |  |  +-rw condition-para-match-pattern? enumeration
|  |  |  |  +-rw condition-para-target-value
|  |  |  |  |  +-rw string-value?      string
|  |  |  |  |  +-rw int-value?        int64
|  |  |  |  |  +-rw range-value
|  |  |  |  |  |  +-rw min              int64
|  |  |  |  |  |  +-rw max              int64
|  |  |  |  |  +-rw precursor-relation-operator? enumeration
|  |  |  |  +-rw order?                uint32

```

```

|   +---rw action* [action-name]
|   |   +---rw action-name          action-name
|   |   +---rw parameter-values
|   |   |   +---rw string-value* [value order]
|   |   |   |   +---rw value          string
|   |   |   |   +---rw order          uint32
|   |   |   +---rw int-value* [value order]
|   |   |   |   +---rw value          int64
|   |   |   |   +---rw order          uint32

```

```

|   |   |   +---rw range-value
|   |   |   |   +---rw min          int64
|   |   |   |   +---rw max          int64
|   |   +---rw order?          uint32
|   +---rw constraint
|   |   +---rw constraint-segment* [constraint-segment-id]
|   |   |   +---rw constraint-segment-id constraint-segment-id
|   |   |   +---rw constraint-para-name? constraint-parameter-name
|   |   |   +---rw constraint-para-match-pattern? enumeration
|   |   |   +---rw constraint-para-target-value
|   |   |   |   +---rw string-value? string
|   |   |   |   +---rw int-value?   int64
|   |   |   |   +---rw range-value
|   |   |   |   |   +---rw min          int64
|   |   |   |   |   +---rw max          int64
|   |   |   +---rw precursor-relation-operator? enumeration
|   |   +---rw order?          uint32
+---rw result
|   +---rw left-value
|   |   +---rw (value-type)?
|   |   |   +---:(string)
|   |   |   |   +---rw single-string-value? string
|   |   |   +---:(string-list)
|   |   |   |   +---rw string-value* [value order]
|   |   |   |   |   +---rw value          string
|   |   |   |   |   +---rw order          uint32
|   |   |   +---:(int)
|   |   |   |   +---rw single-int-value? int64
|   |   |   +---:(int-list)
|   |   |   |   +---rw int-value* [value order]
|   |   |   |   |   +---rw value          int64
|   |   |   |   |   +---rw order          uint32

```

```

|      +--:(variable)
|      |  +--rw variable-name?                string
|      +--:(calculation-expression)
|      |  +--rw calculation-operator?          enumeration
|      |  +--rw calculation-leaf-value?        string
|      |  +--rw calculation-right-value?       string
+--rw relational-operator
|  +--rw relational-operator?                  enumeration
+--rw right-value
|  +--rw (value-type)?
|  |  +--:(string)
|  |  |  +--rw single-string-value?            string
|  |  +--:(string-list)
|  |  |  +--rw string-value* [value order]
|  |  |  |  +--rw value                        string
|  |  |  |  +--rw order                       uint32

```

```

+--:(int)
|  +--rw single-int-value?                int64
+--:(int-list)
|  +--rw int-value* [value order]
|  |  +--rw value                        int64
|  |  +--rw order                       uint32
+--:(variable)
|  +--rw variable-name?                  string
+--:(calculation-expression)
|  +--rw calculation-operator?            enumeration
|  +--rw calculation-leaf-value?         string
|  +--rw calculation-right-value?        string

```

Figure 3: Figure 4 structure of the declarative policy module

[4.2.](#) Declarative Policy Base Data Model in YANG Module

<CODE BEGINS> file "ietf-declarative-policy@2015-12-09.yang"

```

module ietf-declarative-policy {

  yang-version 1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-declarative-policy";

```

```

prefix "dpolicy";

import ietf-yang-types {
    prefix yang;
}

organization
    "Individual I-D as input to SUPA WG";

contact
    "Editor:    Bert Wijnen
               <mailto:bwietf@bwijnen.net>

    Author:    Tianran Zhou
               <mailto:zhoutianran@huawei.com>

    Author:    Yinben Xia
               <mailto:xiayinben@huawei.com>
    ";

description
    "This YANG module defines essential components for the
    management of declarative policies.

```

Copyright (c) 2015 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, is permitted pursuant to, and subject to the license terms contained in, the Simplified BSD License set forth in [Section 4.c](#) of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in the module text are to be interpreted as described in [RFC 2119](#) (<http://tools.ietf.org/html/rfc2119>).

This version of this YANG module is part of RFC XXXX (<http://tools.ietf.org/html/rfcXXXX>); see the RFC itself for full legal notices.";

```

revision "2015-12-09" {
    description
        "Initial revision.";
    reference
        "RFC XXXX: A YANG Data Model for Declarative Policy.";
}

typedef uuid {
    type string {
        pattern '[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-' +
            '[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}';
    }
    description
        "A Universally Unique Identifier in the string
        representation defined in RFC 4122. The canonical
        representation uses lower case characters.

        The following is an example of a UUID in string
        representation:
        f81d4fae-7dec-11d0-a765-00a0c91e6bf6";
    reference
        "RFC 4122: A Universally Unique Identifier (UUID) URN
        Namespace";
}

typedef unique-id {
    type uuid;
    description
        "A globally unique identifier.";
}

```

```

}

typedef name {
    type string {
        length "1..256";
        pattern '[a-zA-Z]([a-zA-Z0-9\-\_\.])*';
    }
    description
        "A generic string name type. Must start with a
        letter";
}

```

```

typedef parameter-name {
    type name;
    description
        "A name for a parameter.";
}

typedef action-name {
    type name;
    description
        "A name for an action.";
}

typedef condition-segment-id {
    type unique-id;
    description
        "A unique ID for a segment of the condition in an
        operation.";
}

typedef constraint-segment-id {
    type unique-id;
    description
        "A unique ID for a segment of the constraint in
        an operation.";
}

typedef condition-parameter-name {
    type name;
    description
        "A name for a parameter of condition.";
}

typedef constraint-parameter-name {
    type name;
    description
        "A name for a parameter of constraint.";
}

```

```

}

typedef policy-id {
    type unique-id;
}

```

```

        description
            "A unique ID for a policy.";
    }

typedef policy-name {
    type string;
    description
        "A name for a policy.";
}

typedef object-id {
    type unique-id;
    description
        "A unique ID for a parameterized object.";
}

typedef match-pattern-operator {
    type enumeration {
        enum less-than {
            description "less than";
        }
        enum not-less-than {
            description "not less than";
        }
        enum equal {
            description "equal";
        }
        enum not-equal {
            description "not equal";
        }
        enum greater-than {
            description "greater than";
        }
        enum not-greater-than {
            description "not greater than";
        }
        enum between {
            description "between";
        }
        enum periodical {
            description "periodical";
        }
    }
    description "pattern matching operators";
}

```

```
    }

    container top_declarative_policy {
        description "Top level declarative policy container";

        container condition-parameter-definitions {
            description "List of condition parameters";
            list condition-parameter-definition {
                key "parameter-name";
                description
                    "Defines the condition parameter with its
                     metadata.";
                leaf parameter-name {
                    type parameter-name;
                    mandatory true;
                    description
                        "A name for the condition parameter.";
                }

                leaf parameter-value-type {
                    type enumeration {
                        enum string {
                            description
                                "A string-valued parameter.";
                        }
                        enum int {
                            description
                                "An integer-valued parameter.";
                        }
                        enum range {
                            description
                                "An integer-range parameter.";
                        }
                    }
                    description
                        "A type of value for the condition
                         parameter.";
                }
            }
        }

        container parameter-match-patterns {
            description
                "Defines the match patterns of the condition
                 parameter.";

            leaf-list parameter-match-pattern {
                type match-pattern-operator;
                description
```

"pattern matching operator";

```
    }
  }
}

container constraint-parameter-definitions {
  description "Parameter definitions fro constraints";
  list constraint-parameter-definition {
    key "parameter-name";
    description
      "Defines the constraint parameter with its
      metadata.";

    leaf parameter-name {
      type parameter-name;
      mandatory true;
      description
        "A name for the constraint parameter.";
    }

    leaf parameter-value-type {
      type enumeration {
        enum string {
          description
            "A string-valued parameter.";
        }
        enum int {
          description
            "An integer-valued parameter.";
        }
        enum range {
          description
            "An integer-range parameter.";
        }
      }
      description
        "A type of value for the constraint
        parameter.";
    }
  }
}
```

```

    container parameter-match-patterns {
        description
            "Defines the match patterns of the constraint
            parameter.";

        leaf-list parameter-match-pattern {
            type match-pattern-operator;

```

```

        description
            "pattern matching operator";
    }
}
}

container action-definitions {
    description "Action definitions";
    list action-definition {
        key "action-name";
        description
            "Defines the actions which will be supported.";

        leaf action-name {
            type action-name;
            mandatory true;
            description
                "A name for the action definition.";
        }

        leaf parameter-value-type {
            type enumeration {
                enum string {
                    description
                        "A string-valued parameter.";
                }
                enum int {
                    description
                        "An integer-valued parameter.";
                }
                enum range {
                    description

```

```

        "An integer-range parameter.";
    }
}
default string;
description
    "The type of the action parameter.";
}
}

grouping condition-instance {
    description
        "Specific a instance of condition whose metadata has
        been filled in.";
}

```

```

list condition-segment {
    key "condition-segment-id";
    description
        "The segment entities will be composed into a
        whole condition entity. Each segment will be
        a sub expression of the condition.";

    leaf condition-segment-id {
        type condition-segment-id;
        mandatory true;
        description
            "A unique ID for a condition segment.";
    }

    leaf condition-parameter-name {
        type condition-parameter-name;
        description
            "A name for a condition parameter.";
    }

    leaf condition-parameter-match-pattern {
        type match-pattern-operator;
        description
            "The operator for the condition parameter
            and it's target value.";
    }
}

```

```

container condition-parameter-target-value {
    description
        "The target value of condition parameter.";

    leaf string-value {
        type string;
        description "The value is of type 'string'";
    }

    leaf int-value {
        type int64;
        description "The value is of type 'int64'";
    }

    container range-value {
        description
            "The min/max of a range.";
        leaf min {
            type int64;
            mandatory true;
            description "minimum value of the range";

```

```

    }

    leaf max {
        type int64;
        mandatory true;
        description "maximum value of the range";
    }
}

leaf precursor-relation-operator {
    type enumeration {
        enum none {
            description
                "no precursor-relation-operator";
        }
        enum and {
            description
                "'and' precursor-relation-operator";

```

```

    }
    enum or {
        description
            "'or' precursor-relation-operator";
    }
    enum not {
        description
            "'not' precursor-relation-operator";
    }
}
description "the pre-cursor relation operation.";
}

leaf order {
    type uint32;
    description
        "The order of the segment in the whole
        condition entity.";
}
}
}

```

```

grouping constraint-instance {
    description
        "Specific a instance of constraint whose metadata
        has been filled in.";

    list constraint-segment {

```

```

key "constraint-segment-id";
description
    "The segment entities will be composed into a
    whole constraint entity. Each segment will be
    a sub expression of the constraint.";

leaf constraint-segment-id {
    type constraint-segment-id;
    mandatory true;
    description
        "A unique ID for a constraint segment.";
}

```

```

leaf constraint-parameter-name {
    type constraint-parameter-name;
    description
        "A name for a constraint parameter.";
}

leaf constraint-parameter-match-pattern {
    type match-pattern-operator;
    description
        "The operator for the constraint parameter
        and it's target value.";
}

container constraint-parameter-target-value {
    description
        "The target value of constraint parameter.";

    leaf string-value {
        type string;
        description "the string value.";
    }

    leaf int-value {
        type int64;
        description "the int64 value.";
    }

    container range-value {
        description "a range of values.";
        leaf min {
            type int64;
            mandatory true;
            description "the min value in the range.";
        }
    }
}

```

```

leaf max {
    type int64;
    mandatory true;
    description "the max value in the range.";
}

```

```

    }
}

leaf precursor-relation-operator {
    type enumeration {
        enum none {
            description "no relational operation.";
        }
        enum and {
            description "'and' relational operation.";
        }
        enum or {
            description "'or' relational operation.";
        }
        enum not {
            description "'not' relational operation.";
        }
    }
    description
        "The logical operator between current segment
        and the next segment.";
}

leaf order {
    type uint32;
    description
        "The order of the segment in the whole
        constraint entity.";
}
}

grouping action-instance {
    description
        "Specific a action instance whose metadata has
        been filled in.";

    leaf action-name {
        type action-name;
        mandatory true;
        description

```

```

        "A name for a action instance.";
    }

    container parameter-values {
        description
            "The parameter value list of an action.";

        list string-value {
            key "value order";
            description "list of string values.";
            leaf value {
                type string;
                description "The string value.";
            }

            leaf order {
                type uint32;
                description "The order of the value in the
                    list.";
            }
        }

        list int-value {
            key "value order";
            description "list of integer values.";
            leaf value {
                type int64;
                description "The int64 value.";
            }

            leaf order {
                type uint32;
                description "The order of the value in the
                    list.";
            }
        }
    }

    container range-value {
        description "a value range.";
        leaf min {
            type int64;
            mandatory true;
            description "The min value in the range.";
        }

        leaf max {
            type int64;
            mandatory true;

```

Internet-Draft

Data Model for Declarative Policy

December 2015

```
        description "The max value in the range.";
    }
}
}

grouping operation {
    description "Defines an operation.";
    container condition {
        uses condition-instance;
        description "Defines a condition";
    }

    list action {
        key "action-name";
        uses action-instance;
        min-elements 1;
        description
            "The action list for the operation instance.";
        leaf order {
            type uint32;
            description
                "The order of an action instance in
                execution sequence.";
        }
    }

    container constraint{
        uses constraint-instance;
        description "The constraint definition.";
    }
}

grouping relational-expression-value {
    description "The relational expression.";
    choice value-type {

        description
            "The value of the relation expression can be
            different type.";
        case string{
            leaf single-string-value {
```

```

        type string;
        description "The string value of the
                    variable.";
    }
}
case string-list {

```

```

    list string-value {
        key "value order";
        description "List of string values.";
        leaf value {
            type string;
            description "The string value of the
                        variable.";
        }
        leaf order {
            type uint32;
            description "the order in the list.";
        }
    }
}
case int {
    leaf single-int-value {
        type int64;
        description "The value of the int64
                    variable.";
    }
}
case int-list {
    list int-value {
        key "value order";
        description "List of integer values.";
        leaf value {
            type int64;
            description "the value of the int64
                        variable.";
        }
    }

    leaf order {
        type uint32;
        description "the order in the list.";
    }
}

```

```

    }
}
case variable {
    leaf variable-name {
        type string;
        description "the name of the variable.";
    }
}

case calculation-expression {

    leaf calculation-operator{
        type enumeration {

```

```

    enum add {
        description "'add' operator";
    }
    enum minus {
        description "'subtract' operator";
    }
    enum multiply {
        description "'multiply' operator";
    }
    enum divide {
        description "'divide' operator";
    }
    enum any {
        description "'any' operator";
    }
    enum all {
        description "'all' operator";
    }
    enum max {
        description "'max' operator";
    }
    enum min {
        description "'min' operator";
    }
    enum average {
        description "'avg' operator";
    }
    enum sum {

```

```

        description "'sum' operator";
    }
    enum count {
        description "'count' operator";
    }
    enum and {
        description "'and' operator";
    }
    enum or {
        description "'or' operator";
    }
    enum not {
        description "'not' operator";
    }
    enum intersection {
        description "'intersetion'
                    operator";
    }
    enum union {
        description "'union' operator";
    }

```

```

    }
    enum complement {
        description "'complement' operator";
    }
}
description
    "The calculation to be performed";
}

leaf calculation-leaf-value {
    type string;
    description
        "The content of the leaf value for a
         calculation expression is an instance
         of relational-expression-value.";
}

leaf calculation-right-value {
    type string;
    description
        "The content of the right value for a

```

```

        calculation expression is an instance
        of relational-expression-value.";
    }
}
}

```

```

grouping result {
  description "The result.";
  container left-value {
    description "The 'left' value for the relational
                operation.";
    uses relational-expression-value;
  }

  container relational-operator {
    description
      "The relational operation to be performed";
    leaf relational-operator {
      type enumeration {
        enum eq {
          description "'equal' operator.";
        }
        enum ne {
          description "'not equal' operator.";
        }
        enum gt {

```

```

        description "'greater than' operator.";
    }
    enum ge {
      description "greater than or equal'
                  operator.";
    }
    enum lt {
      description "'less than' operator.";
    }
    enum le {
      description "'less than or equal'
                  operator.";
    }
    enum bl {

```

```

        description "'belongs to' or
                    'is an element of'
                    operator.";
    }
    enum nb {
        description "'does not belong to' or
                    'is not an element of'
                    operator.";
    }
}
description "the relational operation to be
            performed.";
}
}
container right-value {
    description "The 'right' value for the relational
                operation.";
    uses relational-expression-value;
}
}

container declarative-policy {
    description
        "Aarative policy definition";

    leaf policy-id {
        type policy-id;
        mandatory true;
        description
            "A unique ID for a policy.";
    }

    leaf policy-name {
        type policy-name;

```

```

        mandatory true;
        description
            "A user-visible and unique name for a policy.";
    }

    leaf priority {
        type uint8;

```

```

        default 0;
        description
            "Defines the priority of a operation instance.";
    }

    container policy-validity-period {
        description
            "The valid time of the policy. E.g., the policy
            will be valid 9am-9am daily";
        leaf start {
            type yang:date-and-time;
            description "date and time to start the policy";
        }
        leaf end {
            type yang:date-and-time;
            description "date and time to end the policy";
        }
        leaf duration {
            type uint32;
            description "duration of the policy";
        }
        leaf periodicity {
            type enumeration {
                enum daily {
                    value 0;
                    description
                        "The policy is repeated daily";
                }
                enum monthly {
                    value 1;
                    description
                        "The policy is repeated monthly";
                }
            }
            description "How the policy is repeated";
        }
    }

    leaf policy-rule-deploy-status {
        type enumeration {
            enum undefined {

```

```

        description "undefined";
    }
    enum deployed_and_enabled {
        description "deployed and enabled";
    }
    enum deployed_and_testing {
        description "deployed and in test";
    }
    enum deployed_and_disabled {
        description "deployed but not enabled";
    }
    enum ready_to_be_deployed {
        description "ready to be deployed";
    }
    enum not_deployed {
        description "not deployed";
    }
}
description
    "The deploy status of the policy.";
}

leaf policy-rule-exec-status {
    type enumeration {
        enum undefined {
            description "undefined";
        }
        enum executed_and_succeeded{
            description
                "executed and SUCCEEDED (operational mode)";
        }
        enum executed_and_failed{
            description
                "executed and FAILED (operational mode)";
        }
        enum executing {
            description
                "currently executing (operational mode)";
        }
        enum tested_and_succeeded {
            description
                "executed and SUCCEEDED (test mode)";
        }
    }
    enum tested_and_failed {
        description
            "executed and FAILED (test mode)";
    }
    enum testing {

```

Internet-Draft

Data Model for Declarative Policy

December 2015

```
        description
            "currently executing (test mode)";
    }
}
description
    "The executing status of the policy.";
}

leaf target-object {
    type object-id;
    mandatory true;
    description
        "The target object which the operation instance
        will apply to.";
}

container declarative-policy-rule {
    description "declarative policy rules.";
    container operation {
        uses operation;
        description
            "An operation based declarative policy rule";
    }
    container result {
        uses result;
        description
            "A result based declarative policy rule";
    }
}
}
}
<CODE ENDS>
```

[5. Declarative Policy Data Model Examples](#)

[5.1. Link utilization optimization](#)

For the link utilization optimization, there will be many kinds of policy requirements and expressions. For example:

Policy 1: In one set of links, keep all link utilization below 70%.

Policy 2: In one set of links, minimize the average link utilization.

Both of the policies are target to a set of links, which could include the following attributes:

uti: link bandwidth utility.

bandwidth: bandwidth of the link.

For the policy 1, we can use the result to express the declarative rule statement.

result:

left: uti

operator: lt

right: 70

ult:

For policy 2, we can in alternative use the operation to express the declarative rule statement.

operation:

action: minimize the average link utilization.

[5.2.](#) Policy Based Bandwidth on Demand

There are branch and headquarter sites connected via a WAN connection. The bandwidth of the connection can be adjusted to 1000M during daytime and 100M at night.

This automatic bandwidth adjustment requirement can be expressed by policies target to the connection object. And we can use the operation to express the declarative rule statement.

one operation is:

condition: time = day

action: set the bandwidth to 1000M

the other operation is:

condition: time = night

action: set the bandwidth to 100M

[5.3.](#) Service function chaining

We can also use declarative policies to describe the SFC requirement to simplify the expression.

For example, the gold service user traffic go through the firewall and load balancer services. The policy user do not need to care about the exact function instance to use, but focus on the service intent expression.

This policy is targeted to a flow object, which could include the following attributes:

srcip,destip: the source and destination IP address of the flow.

userlevel: the user's service level of the flow, it can be gold or normal.

The operation rule is:

action: go through, and with parameters: firewall, load balancer.

[6.](#) Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [[RFC6241](#)]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [[RFC6242](#)]. The NETCONF access control model [[RFC6536](#)] provides the means to restrict access for particular

NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

There are a number of data nodes defined in the YANG module which are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., <edit-config>) to these data nodes without proper protection can have a negative effect on network operations.

[7.](#) IANA Considerations

This memo includes no request to IANA.

[8.](#) Acknowledgements

The authors would like to thanks the valuable comments made by:.

This document was produced using the xml2rfc tool [[RFC2629](#)].

Zhou, et al.

Expires June 11, 2016

[Page 33]

Internet-Draft

Data Model for Declarative Policy

December 2015

[9.](#) Informative References

[I-D.chen-sup-a-eca-data-model]

Chen, M., Contreras, L., Hayashi, M., and T. Tsou, "ECA Policy YANG Data Model", [draft-chen-sup-a-eca-data-model-05](#) (work in progress), October 2015.

[I-D.klyus-sup-a-proposition]

Klyus, M. and J. Strassner, "SUPA Value Proposition", [draft-klyus-sup-a-proposition-02](#) (work in progress), July 2015.

[I-D.xia-sdnrg-nemo-language]

Xia, Y., Jiang, S., Zhou, T., and S. Hares, "NEMO (NEtwork MOdeling) Language", [draft-xia-sdnrg-nemo-language-03](#) (work in progress), October 2015.

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", [RFC 2629](#), DOI 10.17487/RFC2629, June 1999, <<http://www.rfc-editor.org/info/rfc2629>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", [RFC 6020](#), DOI 10.17487/RFC6020, October 2010, <<http://www.rfc-editor.org/info/rfc6020>>.
- [RFC6021] Schoenwaelder, J., Ed., "Common YANG Data Types", [RFC 6021](#), DOI 10.17487/RFC6021, October 2010, <<http://www.rfc-editor.org/info/rfc6021>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", [RFC 6241](#), DOI 10.17487/RFC6241, June 2011, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", [RFC 6242](#), DOI 10.17487/RFC6242, June 2011, <<http://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", [RFC 6536](#), DOI 10.17487/RFC6536, March 2012, <<http://www.rfc-editor.org/info/rfc6536>>.

Authors' Addresses

Tianran Zhou
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: zhoutianran@huawei.com

Yinben Xia
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095

P.R. China

Email: xiayinben@huawei.com

Bert Wijnen (editor)

Consultant

Schagen 33

3461 GL Linschoten

The Netherlands

Email: bertietf@bwijnen.net