Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 1, 2018

I. Bryskin
Huawei Technologies
X. Liu
Jabil
A. Clemm
Huawei
H. Birkholz
Fraunhofer SIT
T. Zhou
Huawei

February 28, 2018

Generalized Network Control Automation YANG Model
draft-bryskin-netconf-automation-yang-01

Abstract

This document describes a YANG data model for the Generalized Network Control Automation (GNCA), aimed to define an abstract and uniform semantics for NETCONF/YANG scripts in the form of Event-Condition-Action (ECA) containers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

NETCONF/YANG has proved to be very successful in facilitating interactive network control paradigm, in which a network control application (controller) requests the network server to perform certain (re-)configurations, then, retrieves network states of interest, then asks for more (re-)configurations and so forth. This said, there are multiple use cases that require stringing network configuration requests together with conditioning their order of execution based on certain events detected in the network, as well as current, historical or predicted network states, and pushing such request batches/scripts to the network server in order to control the network close loop automation processes. The Generalized Network Control Automation (GNCA) YANG model introduced by this document defines an abstract and uniform semantics of such NETCONF/YANG scripts in the form of Event-Condition-Action (ECA) containers.

2. Purpose

The purpose of the Generalized Network Control Automation (GNCA) YANG model is to enable an environment allowing for manipulation of close loop network automation via configuration of abstract Event-Condition-Action (ECA) scripts. The model defines the semantics of said scripts; however, how the scripts are actually implemented by the server is not governed by the model. The server may, for example, based on pushed ECA configuration, generate and execute server specific scripts. How this is done is outside of the scope of this document.

Although not all event response behaviors could be delegated to the network server, there are many circumstances where it is highly desirable. For example:

a. Reaction to many network events is well understood and does not require additional information (such as broader or higher level network view or analytics input);

b. It is often imperative for the network to start acting as quickly as the event is detected. In other words, there might simply be no time for the network-controller communication;

c. A paradigm in which a controller micro-manages every network behavior does not scale. Simply put, there are always things that the network has to do autonomously (i.e. when the controller is not looking), which does not mean that the controller should have no say as to how said things need to be done;

d. Numerous important use cases and applications can benefit from ability to define in an abstract and uniformly way a programmable logic in one place (e.g. on the controller) and execute it someplace else (e.g. on the server).

The main objective of the GNCA is to generalize the ECA network control style, so that it could be applied to arbitrary network events/conditions and manipulate network auto-control of large variety of network resources. Such a generalization would allow, for example, conveying to the network a coherent/ordered/prioritized behavior for recovering from a network failure or failures affecting simultaneously multiple services, instruct the network how to fight rapidly developing congestions, what to do when power outage is detected and so forth. In fact, it could be argued that without some sort of close loop network control automation hierarchical network control realistically could not be achieved.

The GNCA YANG model could be also thought of as an attempt to generalize the smart filter subscription machinery by stating that sending a notification is one of possibly multiple actions that network could perform reacting to the specified smart trigger. According to "Smart filters for Push Updates - Problem Statement" [I-D.clemm-netconf-push-smart-filters-ps]:

"They [smart filters] are also useful for network automation, in which automated actions are automatically triggered based on when certain events in the network occur while certain conditions hold. A YANG-Push subscription with a smart filter can in effect act as a source for such events. Combined with an optional check for a condition when an event is observed, this can serve as the basis of action."

In a nutshell GNCA facilitates an environment in which the network automation logic could be defined in a form of ECA scripts by a client, pushed to the network before the events of interest happen, and executed by the network after the events are detected.

Finally, according to the smart filters problem statement, the following smart filters will be considered:

"o Filters that involve freely programmable logic"

ECA scripts could serve as a vehicle to associate with a smart filter a complex freely programmable logic to detect the event of interest in the first place.

3. GNCA concepts and constructs

3.1. Policy Variable (PV)

Policy Variable (PV) is a structure to keep interim results/meta data during the execution of an ECA script. For example, a PV could be used as an output parameter of an RPC invoked by ECA1 to be used in a condition expression for ECA2.

With respect to ECAs the scope of a PV is either global or (ECA) local. Global PVs are permanent. They are kept in the top level PV container and are shared between all ECAs of the script. Local PVs are kept within the internal PV container of an ECA. Local PVs could be either dynamic - appear/disappear with start/stop of the ECA execution, or static - exist as long as the ECA is configured.

Each PV has the following attributes:

o Globally or ECA scope unique name;

- type - either pre-defined (e.g. Boolea, integer, uint64,etc.) or specified via XPath pointing to a data store node or a sub-tree of the required structure. For example, PV named "Link" could be associated with the "/TE_Topologies/TE_Links/TE_Link" XPath in accordance with the TE Topology model [I-D.ietf-teas-yang-te-topo] and hence be capable of accommodating the entire TE Link container;

- value - data stored in the PV structured according to the PV type.

The following operations are allowed with/on a PV:

- initialize (with a constant/enum/identity);

- assign (with contents of another same type PV);

- read (retrieve data store contents pointed by the specified same type XPath/sub-tree);

- write (modify same type CONFIG=TRUE data store state with the PV's content/value);

- insert (PV's content into a same type list);

- iterate (copy into PV one by one same type list elements) (See examples of definition of and operations with PVs in Section 5).

- function calls in a form of F(dst, src), where F is an identity of a function from extendable function library, dst and src are destination and source PVs respectively, the function's input parameters, with the result returned in dst.

Arbitrary expressions with PVs are for future study.

PVs could be used as input/output of an ECA invoked RPC. PVs could also be a source of information sent to the client in notification messages.

PVs could be used in condition expressions (see Section 5).

The model structure for the Policy Variable is shown below:

```
   +--rw policy-variables
   |  +--rw policy-variable* [name]
   |     +--rw name     string
   |     +--rw (type-choice)?
   |     |  +--:(common)
   |     |  |  +--rw type?    identityref
   |     |  +--:(xpath)
   |     |     +--rw xpath?   string
   |     +--rw value?   <anydata>
```

3.2. ECA Event

ECA Event is any subscribable event notification either explicitly defined in a YANG module supported by the server or a smart filter conveyed to the server via smart filter subscription. Additionally, an event could be associated with a one-time or periodic timer.

Event notification contents become in the ECA context implicit local dynamic PVs with automatically generated names. Let's assume Network_Failure_Is_Detected event is defined that carries to a subscriber the detected failure type (failureType), ID (failureID) and the affected by the failure TE link state (teLInk). In the context of the associated with the Networ_Failure_Is_Detected event ECA failureType, failureID and teLink are implicit local dynamic PVs that could be used in the embodied Condition and Action containers along with explicitly defined global and ECA local (static and/or dynamic) PVs.

One way to think of NETWONF/YANG Event Notification is as Remote Procedure Call-back, i.e. server->client directed RPC When a subscribable NETWONF/YANG Event and associated with it ECA is pushed to the server within an ECA script, the server is expected to interpret this as follows: take the contents of the event notification and execute the logic defined by the associated Condition-Action chain (as I (client) would do on receipt of the notification) in order to decide how to react to the event. Recall that the whole purpose of ECA scripts is to reduce as much as possible the client-server communication.

All events (specified in at least one ECA pushed to the server) are required to be constantly monitored by the server. One way to think of this is that the server subscribes to its own publications with respect to all events that are associated with at least one ECA.

3.3. ECA Condition

ECA Condition is evaluated to TRUE or FALSE logical expression. There are two ways how an ECA Condition could be specified:

- in a form of XPath expression;

- as a hierarchy of comparisons and logical combinations of thereof.

The former option allows for specifying a condition of arbitrary complexity as a single string with an XPath expression, in which pertinent PVs and data store states are referred to by their respective positions in the YANG tree.

The latter option allows for configuring logical hierarchies. The bottom of said hierarchies are primitive comparisons (micro-conditions) specified in a form of:

<arg1><relation><arg2>

where arg1 and arg2 represent either constant/enum/identity, PV or pointed by XPath data store node or sub-tree,

relation is one of the comparison operations from the set: ==, !=, >, <, >=, <=

Primitive comparisons could be combined hierarchically into macro-conditions via && and || logical operations.

Regardless of the choice of their specification, ECA Conditions are associated with ECA Events and evaluated only within event threads triggered by the event detection.

When an ECA Condition is evaluated to TRUE, the associated with it ECA Action is executed.

The model structure for the ECA Condition is shown below:

```
   +--rw conditions
   |  +--rw condition* [name]
   |     +--rw name                      string
   |     +--rw (expression-choice)?
   |        +--:(logical-operation)
   |        |  +--rw logical-operation-type?   identityref
   |        |  +--rw comparison-operation* [name]
   |        |  |  +--rw name                string
   |        |  |  +--rw comparision-type?   identityref
   |        |  |  +--rw arg1
   |        |  |  |  +--rw policy-argument
   |        |  |  |     +--rw type?                    identityref
   |        |  |  |     +--rw (argument-choice)?
   |        |  |  |        +--:(policy-constant)
   |        |  |  |        |  +--rw constant?                string
   |        |  |  |        +--:(policy-variable)
   |        |  |  |        |  +--rw policy-variable?         leafref
   |        |  |  |        +--:(local-policy-variable)
   |        |  |  |        |  +--rw local-policy-variable?   leafref
   |        |  |  |        +--:(xpath)
   |        |  |  |           +--rw xpath?                   string
   |        |  |  +--rw arg2
   |        |  |     +--rw policy-argument
   |        |  |        +--rw type?                    identityref
   |        |  |        +--rw (argument-choice)?
   |        |  |           +--:(policy-constant)
   |        |  |           |  +--rw constant?                string
   |        |  |           +--:(policy-variable)
   |        |  |           |  +--rw policy-variable?         leafref
   |        |  |           +--:(local-policy-variable)
   |        |  |           |  +--rw local-policy-variable?   leafref
   |        |  |           +--:(xpath)
   |        |  |              +--rw xpath?                   string
   |        |  +--rw sub-condition* [name]
   |        |     +--rw name    -> /gnca/conditions/condition/name
   |        +--:(xpath)
   |           +--rw condition-xpath?          string
```

The policy arguments arg1 and arg2 have the following structure:

```
   +--rw policy-argument
      +--rw type?                     identityref
      +--rw (argument-choice)?
         +--:(policy-constant)
         |  +--rw constant?                 string
         +--:(policy-variable)
         |  +--rw policy-variable?          leafref
         +--:(local-policy-variable)
         |  +--rw local-policy-variable?   leafref
         +--:(xpath)
            +--rw xpath?                    string
```

3.4. ECA Action

ECA Action is one of the following operations to be carried out by a server:

o (-re)configuration - modifying a CONFIG=TRUE data store state

o (re-)configuration scheduling - scheduling one time or periodic (re-)configuration in the future

o sending one time notification;

o adding/removing event notify subscription (essentially, the same action as performed when a client explicitly adds/removes a subscription)

o executing an RPC defined by a YANG module supported by the server (the same action as performed when a client interactively calls the RPC);

o performing operations and function calls on PVs (such as assign, read, insert, iterate, etc);

o starting/stopping timers;

o stopping current ECA;

o invoking another ECA;

o NO-ACTION action - meaningful only within ECA's Cleanup Condition-Action list to indicate that the ECA's Normal Condition-Action thread must be terminated as soon as one of the required Actions is rejected by the server (see more Section 3.4).

Two points are worth noting:

1. When a NETWONF/YANG RPC appears in an ECA Action body, the server is expected to interpret this as follows: execute the same logic, as when the client explicitly calls said RPC via NETCONF. For example, when TE_Tunnel_Path_Computation RPC is found in the currently executed Action, the server is expected to call its TE path computation engine and pass to it the specified parameters in the Action input.

2. When a "Send notification" action is configured as an ECA Action, the notification message to be sent to the client may contain not only elements of the data store (as, for example, YANG PUSH or smart filter notifications do), but also the contents of global and local PVs, which store results of arbitrary operations performed on the data store contents (possibly over arbitrary period of time) to determine, for example, history/evolution of data store changes, median values, ranges and rates of the changes, results of configured function calls and expressions, etc. - in short, any data the client may find interesting about the associated event with all the logic to compute said data delegated to the server.

Multiple ECA Condition/Action pairs could be combined into a macro-action.

Multiple ECA (macro-)Actions could be triggered by a single ECA event.

Any given ECA Condition or Action may appear in more than one ECAs.

The model structure for the ECA Action is shown below:

```
   +--rw actions
   |  +--rw action* [name]
   |     +--rw name               string
   |     +--rw action-element* [name]
   |     |  +--rw name                string
   |     |  +--rw action-type?        identityref
   |     |  +--rw (action-operation)?
   |     |     +--:(action)
   |     |     |  +--rw action-name?
   |     |     |          -> /gnca/actions/action/name
   |     |     +--:(content-moving)
   |     |     |  +--rw content-moving
   |     |     |     +--rw content-moving-type?   identityref
   |     |     |     +--rw src
   |     |     |     |  +--rw policy-argument
   |     |     |     +--rw dst
   |     |     |        +--rw policy-argument
   |     |     +--:(function-call)
   |     |     |  +--rw function-call
   |     |     |     +--rw function-type?   identityref
   |     |     |     +--rw src
   |     |     |     |  +--rw policy-argument
   |     |     |     +--rw dst
   |     |     |        +--rw policy-argument
   |     |     +--:(rpc-operation)
   |     |     |  +--rw rpc-operation
   |     |     |     +--rw name?              string
   |     |     |     +--rw nc-action-xpath?   string
   |     |     |     +--rw policy-variable* [name]
   |     |     |        +--rw name              string
   |     |     |        +--rw policy-argument
   |     |     +--:(notify-operation)
   |     |        +--rw notify-operation
   |     |           +--rw name?              string
   |     |           +--rw policy-variable* [name]
   |     |              +--rw name              string
   |     |              +--rw policy-argument
   |     +--rw time-schedule
   |        +--rw start?             yang:date-and-time
   |        +--rw repeat-interval?   string
```

3.5. ECA

An ECA container includes:

o Event name

- List of local PVs. As mentioned, local PVs could be configured as dynamic (their instances appear/disappear with start/stop of the ECA execution) or static (their instances exisr as long as the ECA is configured). The ECA input (the contents of the associated notification message, such as YANG PUSH or smart filter notification message) are the ECA's implict local dynamic PVs with automatically generated names

- Normal CONDITION-ACTION list: configured conditions each with associated actions to be executed if the condition is evaluated to TRUE

- Cleanup CONDITION-ACTION list: configured conditions/actions to be evaluated/executed in case any Action from the Normal CONDITION-ACTION list was attempted and rejected by the server. In other words, this list specifies the ECA cleanup/unroll logic after rejection of an Action from the Normal CONDITION-ACTION list. An empty Cleanup CONDITION-ACTION list signifies that the ECA's normal Actions should be executed regardless of whether the previously attempted ECA Actions were rejected or not by the server. Cleanup CONDITION-ACTION list containing a single NO-ACTION Action signifies that the ECA thread is to be immediately terminated on rejection of any attempted Action (without executing any cleanup logic)

4. Where ECA scripts are executed?

It could be said that the main idea of the GNCA is decoupling the place where the network control logic is defined from the place where it is executed. In previous sections of this document it is assumed that the network control logic is defined by a client and pushed to and executed by the network (server). This is accomplished via ECA scripts, which are essentially bunches of regular NETCONF style operations (such as get, set, call rpc) and event notifications glued together via Policy Variables, PVs. It is worth noting that said ECA scripts could be easily moved around and executed by any entity supporting the GNCA YANG model (i.e. capable of interpreting the ECA scripts). One interesting implication of this is that the ECA scripts could be executed by neither client nor server, but a 3d party entity, for instance, with a special focus on the control of a particular network domain or/and special availability of/proximity to information/ resources that could contribute to the network control decision process. For example, the ECA scripts could be pushed to a Path Computation Element (PCE) adopted to support the GNCA YANG model. Specialized ECA scripts could be fanned out to multiple specialized controllers to take care of different aspects of a network domain control.

Another interesting idea is to combine the GNCA with hierarchical T-SDN architecture. In particular, the ECA scripts conveyed by a client to a network orchestrator could be pushed (modified or unmodified) hierarchically down to lower level controllers. After all, the goal of the hierarchical T-SDN is to create a paradigm in which the higher level of a controller in the hierarchy, the broader (topologically and/or functionally) its control on the network and the lesser its involvement in the network's micro-management in real time. On the other hand, it is desirable for a higher level controller to have a say as to how the subordinate controllers and, by extension, the network under control should deal with events and situations that are handled autonomously (i.e. without bothering the higher level controller in real time). The ECA scripts pushed down the T-SDN hierarchy may help to achieve this objective.

5. ECA script example

Consider a situation on a TE network when a network failure simultaneously affecting multiple TE tunnels. Normally, the TE network relies in this case on TE tunnels pre-configured protection/ restoration capabilities and lets the TE tunnels to auto-recover themselves independently from each other. However, this default behavior may not be desired in some configurations/use cases because:

a. Recovery procedures of a "greedy" TE tunnel may block the recovery of other TE tunnels;

b. Shared mesh protection/restoration schemes are in place

In such cases the network has to perform the recovery of failure affected TE tunnels as a coordinated process. Furthermore, it is quite common that network resources available for the dynamic recovery procedures are limited, in which case it is desirable to convey to the network the policy/order in which the TE tunnels should be recovered. Different policies may be considered, to name a few:

1. Recover as many TE tunnels as possible;

2. Recover TE tunnels in accordance with their importance/priority;

3. Recover all unprotected TE tunnels before recovering broken connections/LSPs of protected TE tunnels (because the latter can tolerate the failure hopefully until it is repaired).

Let's describe an ECA script that could be pushed by the controller application instructing the network to perform multiple TE tunnel failure recovery according to policy (3) above.

Assumptions: it is assumed that in one or several YANG modules supported by the server the following is defined:

- Subscribable "Network_Failure_Is_Detected" event carrying in the notification message the detected failure type (failureType), ID (failureID) and the affected by the failure TE link ID (linkID);

- RPC "PathDependsOnLink" taking as an input a TE_Path and TE_Link_ID and returning in its output Boolean indicating whether or not the specified path goes through the link with the specified ID;

- RPC "ReplaceTunnelsAwayFromLink" taking as an input a list of TE tunnel key leafrefs and ID of to be avoided link, performing the tunnel replacement away from the link and returning no output.

Explicit (global) PVs:

- name: Yes type: Boolean

- name: lsp xpath: /TE_Tunnels/lsps/lsp

- name tunnel xpath: /TE_Tunnels/tunnels/te_tunnel

- name: unprotected_tunnels xpath: /TE_Tunnels/tunnels/te_tunnel/dependent_tunnels

- name protected_tunnels xpath: /TE_Tunnels/tunnels/te_tunnel/dependent_tunnels

Actions:

name: PopulateTunnelLists

body:

```
lsp iterate xpath:/TE_Tunnels/lsps
{
  rpc: PathDependsOnLink(<lsp>/rro, Yes);
  if(Yes == TRUE )
  {
    tunnel = <lsp>/parrent_tunnel;
    if(<tunnel>/protectionType == UNPROTECTED)
      <tunne>/tunnelName insert unprotected_tunnels
    if(<tunnel>/protectionType != UNPROTECTED)
      <tunne>/tunnelName insert protected_tunnels
  }
}
```

name: RepairTunnels

Body:

```
rpc: ReplaceTunnelsAwayFromLink(unprotected_tunnels, linkID);
rpc: ReplaceTunnelsAwayFromLink(protected_tunnels, linkID);
```

ECA:

eventName: Network_Failure_Is_Detected;

eventParams: failureType, failureID, linkID

Condition: TRUE (always, every time)

Actions:

unprotected_tunnels = 0; protected_tunnels =0;

namedAction:PopulateTunnelLists;

namedAction:RepairTunnels

Note: RPC "PathDependsOnLink" is used in the example for simplicity. The RPC could be easily replaced by a scripted named action similar to PopulateTunnelListes .

6. Complete Model Tree Structure

The complete tree structure of the YANG model defined in this document is as follows:

```
   module: ietf-gnca
       +--rw gnca
          +--rw policy-variables
          |  +--rw policy-variable* [name]
          |     +--rw name     string
          |     +--rw (type-choice)?
          |     |  +--:(common)
          |     |  |  +--rw type?    identityref
          |     |  +--:(xpath)
          |     |     +--rw xpath?   string
          |     +--rw value?   <anydata>
          +--rw conditions
          |  +--rw condition* [name]
          |     +--rw name                       string
          |     +--rw (expression-choice)?
          |        +--:(logical-operation)
          |        |  +--rw logical-operation-type?   identityref
          |        |  +--rw comparison-operation* [name]
          |        |  |  +--rw name                 string
          |        |  |  +--rw comparision-type?   identityref
          |        |  |  +--rw arg1
          |        |  |  |  +--rw policy-argument
          |        |  |  |     +--rw type?
    identityref
          |        |  |  |     +--rw (argument-choice)?
          |        |  |  |        +--:(policy-constant)
          |        |  |  |        |  +--rw constant?
    string
          |        |  |  |        +--:(policy-variable)
          |        |  |  |        |  +--rw policy-variable?
    leafref
          |        |  |  |        +--:(local-policy-variable)
          |        |  |  |        |  +--rw local-policy-variable?
    leafref
          |        |  |  |        +--:(xpath)
          |        |  |  |           +--rw xpath?
    string
          |        |  |  +--rw arg2
          |        |  |     +--rw policy-argument
          |        |  |        +--rw type?
    identityref
          |        |  |        +--rw (argument-choice)?
          |        |  |           +--:(policy-constant)
          |        |  |           |  +--rw constant?
    string
          |        |  |           +--:(policy-variable)
          |        |  |           |  +--rw policy-variable?
    leafref
```

```
          |        |  |           +--:(local-policy-variable)
          |        |  |           |  +--rw local-policy-variable?
    leafref
          |        |  |           +--:(xpath)
          |        |  |              +--rw xpath?
    string
          |        |  +--rw sub-condition* [name]
          |        |     +--rw name    -> /gnca/conditions/condition
   /name
          |        +--:(xpath)
          |           +--rw condition-xpath?          string
          +--rw actions
          |  +--rw action* [name]
          |     +--rw name              string
          |     +--rw action-element* [name]
          |     |  +--rw name                string
          |     |  +--rw action-type?        identityref
          |     |  +--rw (action-operation)?
          |     |     +--:(action)
          |     |     |  +--rw action-name?
          |     |     |           -> /gnca/actions/action/name
          |     |     +--:(content-moving)
          |     |     |  +--rw content-moving
          |     |     |     +--rw content-moving-type?   identityref
          |     |     |     +--rw src
          |     |     |     |  +--rw policy-argument
          |     |     |     |     +--rw type?
          |     |     |     |     |       identityref
          |     |     |     |     +--rw (argument-choice)?
          |     |     |     |        +--:(policy-constant)
          |     |     |     |        |  +--rw constant?
          |     |     |     |        |          string
          |     |     |     |        +--:(policy-variable)
          |     |     |     |        |  +--rw policy-variable?
    leafref
          |     |     |     |        +--:(local-policy-variable)
          |     |     |     |        |  +--rw local-policy-variable?
    leafref
          |     |     |     |        +--:(xpath)
          |     |     |     |           +--rw xpath?
          |     |     |     |                   string
          |     |     |     +--rw dst
          |     |     |        +--rw policy-argument
          |     |     |           +--rw type?
          |     |     |           |       identityref
          |     |     |           +--rw (argument-choice)?
          |     |     |              +--:(policy-constant)
          |     |     |              |  +--rw constant?
```

```
         |     |     |            |          string
         |     |     |            +--:(policy-variable)
         |     |     |            |  +--rw policy-variable?
    leafref
         |     |     |            +--:(local-policy-variable)
         |     |     |            |  +--rw local-policy-variable?
    leafref
         |     |     |            +--:(xpath)
         |     |     |               +--rw xpath?
         |     |     |                        string
         |     |     +--:(function-call)
         |     |     |  +--rw function-call
         |     |     |     +--rw function-type?   identityref
         |     |     |     +--rw src
         |     |     |     |  +--rw policy-argument
         |     |     |     |     +--rw type?
         |     |     |     |     |       identityref
         |     |     |     |     +--rw (argument-choice)?
         |     |     |     |        +--:(policy-constant)
         |     |     |     |        |  +--rw constant?
         |     |     |     |        |          string
         |     |     |     |        +--:(policy-variable)
         |     |     |     |        |  +--rw policy-variable?
    leafref
         |     |     |     |        +--:(local-policy-variable)
         |     |     |     |        |  +--rw local-policy-variable?
    leafref
         |     |     |     |        +--:(xpath)
         |     |     |     |           +--rw xpath?
         |     |     |     |                    string
         |     |     |     +--rw dst
         |     |     |        +--rw policy-argument
         |     |     |           +--rw type?
         |     |     |           |       identityref
         |     |     |           +--rw (argument-choice)?
         |     |     |              +--:(policy-constant)
         |     |     |              |  +--rw constant?
         |     |     |              |          string
         |     |     |              +--:(policy-variable)
         |     |     |              |  +--rw policy-variable?
    leafref
         |     |     |              +--:(local-policy-variable)
         |     |     |              |  +--rw local-policy-variable?
    leafref
         |     |     |              +--:(xpath)
         |     |     |                 +--rw xpath?
         |     |     |                          string
         |     |     +--:(rpc-operation)
```

```
          |     |     |  +--rw rpc-operation
          |     |     |     +--rw name?              string
          |     |     |     +--rw nc-action-xpath?   string
          |     |     |     +--rw policy-variable* [name]
          |     |     |        +--rw name               string
          |     |     |        +--rw policy-argument
          |     |     |           +--rw type?
          |     |     |           |       identityref
          |     |     |           +--rw (argument-choice)?
          |     |     |              +--:(policy-constant)
          |     |     |              |  +--rw constant?
          |     |     |              |          string
          |     |     |              +--:(policy-variable)
          |     |     |              |  +--rw policy-variable?
    leafref
          |     |     |              +--:(local-policy-variable)
          |     |     |              |  +--rw local-policy-variable?
    leafref
          |     |     |              +--:(xpath)
          |     |     |                 +--rw xpath?
          |     |     |                         string
          |     |     +--:(notify-operation)
          |     |        +--rw notify-operation
          |     |           +--rw name?              string
          |     |           +--rw policy-variable* [name]
          |     |              +--rw name               string
          |     |              +--rw policy-argument
          |     |                 +--rw type?
          |     |                 |       identityref
          |     |                 +--rw (argument-choice)?
          |     |                    +--:(policy-constant)
          |     |                    |  +--rw constant?
          |     |                    |          string
          |     |                    +--:(policy-variable)
          |     |                    |  +--rw policy-variable?
    leafref
          |     |                    +--:(local-policy-variable)
          |     |                    |  +--rw local-policy-variable?
    leafref
          |     |                    +--:(xpath)
          |     |                       +--rw xpath?
          |     |                               string
          |     +--rw time-schedule
          |        +--rw start?             yang:date-and-time
          |        +--rw repeat-interval?   string
          +--rw ecas
          |  +--rw eca* [name]
          |     +--rw name                        string
```

```
          |     +--rw event-name                  string
          |     +--rw policy-variable* [name]
          |     |  +--rw name         string
          |     |  +--rw (type-choice)?
          |     |  |  +--:(common)
          |     |  |  |  +--rw type?        identityref
          |     |  |  +--:(xpath)
          |     |  |     +--rw xpath?       string
          |     |  +--rw value?       <anydata>
          |     |  +--rw is-static?   boolean
          |     +--rw condition-action* [name]
          |     |  +--rw name         string
          |     |  +--rw condition?   -> /gnca/conditions/condition/name
          |     |  +--rw action?      -> /gnca/actions/action/name
          |     +--rw cleanup-condition-action* [name]
          |     |  +--rw name         string
          |     |  +--rw condition?   -> /gnca/conditions/condition/name
          |     |  +--rw action?      -> /gnca/actions/action/name
          |     +---x start
          |     +---x stop
          |     +---x pause
          |     +---x resume
          |     +---x next-action
          |     +--ro execution* [id]
          |        +--ro id                                uint32
          |        +--ro oper-status?                      enumeration
          |        +--ro start-time?
          |        |       yang:date-and-time
          |        +--ro stop-time?
          |        |       yang:date-and-time
          |        +--ro next-scheduled-time?
          |        |       yang:date-and-time
          |        +--ro last-condition-action?
          |        |       -> ../../condition-action/name
          |        +--ro last-condition?
          |        |       -> ../../condition-action/condition
          |        +--ro last-action?
          |        |       -> ../../condition-action/action
          |        +--ro last-cleanup-condition-action?
          |                -> ../../cleanup-condition-action/name
          +--rw eca-scripts
          |  +--rw eca-script* [script-name]
          |     +--rw script-name    string
          |     +--rw eca* [eca-name]
          |        +--rw eca-name    -> /gnca/ecas/eca/name
          +--rw running-script?
                  -> /gnca/eca-scripts/eca-script/script-name
```

7. YANG Module

```
   <CODE BEGINS> file "ietf-gnca@2018-02-28.yang"
   module ietf-gnca {
     yang-version 1.1;
     namespace "urn:ietf:params:xml:ns:yang:ietf-gnca";

     prefix "gnca";

     import ietf-yang-types {
       prefix "yang";
     }

     organization
       "IETF Network Configuration (NETCONF) Working Group";

     contact
       "WG Web:   <http://tools.ietf.org/wg/netconf/>
        WG List:  <mailto:netconf@ietf.org>

        Editor:   Igor Bryskin
                  <mailto:Igor.Bryskin@huawei.com>

        Editor:   Xufeng Liu
                  <mailto:Xufeng_Liu@jabil.com>

        Editor:   Alexander Clemm
                  <mailto:ludwig@clemm.org>";

     description
       "Event Condition Action (ECA) model.";

     revision 2018-02-28 {
       description "Initial revision";
       reference "RFC XXXX";
     }

     /*
      * Typedefs
      */
     identity argument-type {
       description
         "Possible values are:
          constant, variable, or datastore state.";
     }

     identity comparison-type {
```

```
         description
           "Possible values are:
            equal, not-equal, greater, greater-equal, less, less-equal.";
       }

       identity logical-operation-type {
         description
           "Possible values are:
            not, or, and.";
       }

       identity function-type {
         description
           "Possible values are:
            plus, minus, mult, divide, remain.";
       }

       identity content-moving-operation-type {
         description
           "Possible values are:
            copy, iterate, insert.";
       }

       identity action-type {
         description
           "Possible values are:
            action, content-move, function-call, rpc, notify.";
       }

       identity policy-variable-type {
         description
           "Possible values are:
            boolean, int32, int64, uint32, uint64, string, etc.";
       }

       /*
        * Groupings
        */
       grouping policy-variable-attributes {
         description
           "Defining the policy variable attributes, including name, type
            and value. These attributes are used as part of the Policy
            Variable (PV) definition.";
         leaf name {
           type string;
           description
             "A string to uniquely identify a Policy Variable (PV), either
              globally for a global PV, or within the soope of ECA for a
```

```
             local PV.";
        }
        choice type-choice {
          description
            "The type of a policy variable may be either a common
             primative type like boolean or a type from existing
             schema node referenced by an XPath string.";
          case common {
            leaf type {
              type identityref {
                base policy-variable-type;
              }
              description
                "A common policy variable type, defined as an
                 identity.";
            }
          }
          case xpath {
            leaf xpath {
              type string;
              description
                "A XPath string, referencing a schema node, whose
                 type is used as the type of the policy variable.";
            }
          }
        }
        anydata value {
          description
            "The value of the policy variable, in a format that is
             determined by the policy type.";
        }
      } // policy-variable-attributes

      grouping policy-argument {
        description
          "Defining a policy argument, which can be used in a comparison
           or an action.";
        container policy-argument {
          description
            "Containing the attributes of a policy argument.";
          leaf type {
            type identityref {
              base argument-type;
            }
            description
              "Identifies the argument type.";
          }
          choice argument-choice {
```

```
           description
             "Argument formation options, depending on the policy
              type.";
           case policy-constant {
             leaf constant {
               type string;
               description
                 "The constant value of the policy argument.";
             }
           }
           case policy-variable {
             leaf policy-variable {
               type leafref {
                 path "/gnca/policy-variables/"
                   + "policy-variable/name";
               }
               description
                 "A reference to a global policy variable, which
                  is shared by all ECA scripts.";
             }
           }
           case local-policy-variable {
             leaf local-policy-variable {
               type leafref {
                 path "/gnca/ecas/eca/policy-variable/name";
               }
               description
                 "A reference to a local policy variable, which
                  is kept within an ECA instance, and appears/
                  disappears with start/stop of the ECA execution.";
             }
           }
           case xpath {
             leaf xpath {
               type string;
               description
                 "An XPath string, referencing the data in the
                  datastore.";
             }
           }
         }
       }
     } // policy-argument

     grouping action-element-attributes {
       description
         "Grouping of action element attributes.";
       leaf action-type {
```

```
          type identityref {
            base action-type;
          }
          description
            "Identifies the action type.";
        }
        choice action-operation {
          description
            "The operation choices that an ECA Action can take.";
          case action {
            leaf action-name {
              type leafref {
                path "/gnca/actions/action/name";
              }
              description
                "The operation is to execute a configured ECA Action.";
            }
          } // action
          case content-moving {
            container content-moving {
              description
                "The operation is to move contents between two policy
                 arguments.";
              leaf content-moving-type {
                type identityref {
                  base content-moving-operation-type;
                }
                description
                  "The type of moving operation, which can be copy,
                   iterate (copy a list of elements one by one), or
                   insert.";
              }
              container src {
                description
                  "The source policy argment.";
                uses policy-argument;
              }
              container dst {
                description
                  "The destination policy argument.";
                uses policy-argument;
              }
            }
          } // content-moving
          case function-call {
            container function-call {
              description
                "The operation is to call a function, which is of one of
```

```
                a few basic predefined types, such as plus, minus,
                multiply, devide, or remainder.";
             leaf function-type {
               type identityref {
                 base function-type;
               }
               description
                 "One of the predefined basic function types, such as
                  plus, minus, multiply, devide, or remainder.";
                      }
             container src {
               description
                 "The source policy argument.";
               uses policy-argument;
             }
             container dst {
               description
                 "The distination policy argument.";
               uses policy-argument;
             }
           }
         } // function-call
         case rpc-operation {
           container rpc-operation {
             description
               "The operation is to call an RPC, which is defined by
                a YANG module supported by the server.";
             leaf name {
               type string;
               description
                 "The name of the YANG RPC or YANG action to be
                  called.";
             }
             leaf nc-action-xpath {
               type string;
               description
                 "The location where the YANG action is defined.
                  This is used if and only if a YANG action is called.
                  This leaf is not set when a YANG RPC is called.";
             }
             list policy-variable {
               key name;
               description
                 "A list of policy arguments used as the input or output
                  parameters passed to the RPC.";
               leaf name {
                 type string;
                 description
```

```
                "A string name used as the list key to form a list
                 of policy arguments.";
            }
            uses policy-argument;
          }
        }
      } // rpc-operation
      case notify-operation {
        container notify-operation {
          description
            "The operation is to send a YANG notification.";
          leaf name {
            type string;
            description
              "Name of the subscribed YANG notification.";
          }
          list policy-variable {
            key name;
            description
              "A list of policy arguments carried in the notification
               message.";
            leaf name {
              type string;
              description
                "A string name used as the list key to form a list
                 of policy arguments.";
            }
            uses policy-argument;
          }
        }
      } // notify-operation
    }
  } // action-element-attributes

  /*
   * Data nodes
   */
  container gnca {
    description
      "Top level container for Generalized Network Control Automation
       (GNCA).";

    // policy-variables
    container policy-variables {
      description
        "Container of global Policy Variables (PVs).";
      list policy-variable {
        key name;
```

```
          description
            "A list of global Policy Variables (PVs), with a string
             name as the entry key.";
          uses policy-variable-attributes;
        }
      } // policy-variables

      container conditions {
        description
          "Container of ECA Conditions.";
        list condition {
          key name;
          description
            "A list of ECA Conditions.";
          leaf name {
            type string;
            description
              "A string name to uniquely identify an ECA Condition
               globally.";
          }
          choice expression-choice {
            description
              "The choices of expression format to specify a condition,
               which can be either a XPath string or a YANG logical
               operation structure.";
            case logical-operation {
              leaf logical-operation-type {
                type identityref {
                  base logical-operation-type;
                }
                description
                  "The logical operation type used to combine the
                   results from the list comparison-operation and the
                   list sub-condition, defined below.";
              }
              list comparison-operation {
                key name;
                description
                  "A list of comparison oprations, each of them defines
                   a comparison in the form of <arg1><relation><arg2>,
                   where <arg1> and <arg2> are policy arguments, while
                   <relation> is the comparison-type, which can be
                   ==, !=, >, <, >=, <=";
                leaf name {
                  type string;
                  description
                    "A string name to uniquely identify a comparison
                     operation.";
```

```
                }
                leaf comparision-type {
                  type identityref {
                    base comparison-type;
                  }
                  description
                    "The comparison operation applied to the two
                     arguments arg1 and arg2 defined blow.";
                }
                container arg1 {
                  description
                    "The policy argument used as the first parameter of
                     the comparison opration.
                     A policy argument represents either a constant, PV
                     or data store value pointed by XPath.";
                  uses policy-argument;
                }
                container arg2 {
                  description
                    "The policy argument used as the secone parameter
                     of the comparison opration.
                     A policy argument represents either a constant, PV
                     or data store value pointed by XPath.";
                  uses policy-argument;
                }
              }
              list sub-condition {
                key name;
                description
                  "A list of sub conditions applied by the
                   logical-operation-type. This list of sub conditions
                   provides the capability of hierarchically combining
                   conditions.";
                leaf name {
                  type leafref {
                    path "/gnca/conditions/condition/name";
                  }
                  description
                    "A reference to a defined condition, which is used
                     as a sub-condition for the logical operation at
                     this hierarchy level.";
                }
              } // sub-condition
            } // logical-operation
            case xpath {
              leaf condition-xpath {
                type string;
                description
```

```
                   "A XPath string, representing a logical expression,
                    which can contain comparisons of datastore values
                    and logical operations in the XPath format.";
              }
            } // xpath
          } // expression-choice
        } // condition
      } // conditions

      container actions {
        description
          "Container of ECA Actions.";
        list action {
          key name;
          description
            "A list of ECA Actions.";
          leaf name {
            type string;
            description
              "A string name to uniquely identify an ECA Action
               globally.";
          }

          list action-element {
            key name;
            description
              "A list of elements contained in an ECA Action. ";
            leaf name {
              type string;
              description
                "A string name to uniquely identify the action element
                 within the scope of an ECA action.";
            }
            uses action-element-attributes;
          }

          container time-schedule {
            description
              "Specifying the time schedule to execute this ECA
               Action.
               If not specified, the ECA Action is executed immediately
               when it is called.";
            leaf start {
              type yang:date-and-time;
              description
                "The start time of the ECA Action.
                 If not specified, the ECA Action is executed
                 immediately when it is called.";
```

```
             }
             leaf repeat-interval {
               type string {
                 pattern
                   '(R\d*/)?P(\d+Y)?(\d+M)?(\d+W)?(\d+D)?T(\d+H)?'
                   + '(\d+M)?(\d+S)?';
               }
               description
                 "The repeat interval to execute this ECA Action.
                  The repeat interval is a string in ISO 8601 format,
                  representing a delay duration or a repeated delay
                  duration.
                  If not specified, the ECA Action is executed without
                  delay and without repetition.";
             }
           } // time-schedule
         }
       } // actions

       container ecas {
         description
           "Container of ECAs.";
         list eca {
           key name;
           description
             "A lis of ECAs";
           leaf name {
             type string;
             description
               "A string name to uniquely identify an ECA globally.";
           }
           leaf event-name {
             type string;
             mandatory true;
             description
               "The name of an event that triggers the execution of
                this ECA.";
           }

           list policy-variable {
             key name;
             description
               "A list of ECA local Policy Variables (PVs), with a
                string name as the entry key.";
             uses policy-variable-attributes;
             leaf is-static {
               type boolean;
               description
```

```
                 "'true' if the PV is static; 'false' if the PV is
                  dynamic.
                  A dynamic PV appears/disappears with the start/stop
                  of the ECA execution; a static PV exists as long as
                  the ECA is configured.";
             }
           }

           list condition-action {
             key name;
             description
               "A list of Condition-Actions, which are configured
                conditions each with associated actions to be executed
                if the condition is evaluated to TRUE";
             leaf name {
               type string;
               description
                 "A string name uniquely identify a Condition-Action
                  within this ECA.";
             }
             leaf condition {
               type leafref {
                 path "/gnca/conditions/condition/name";
               }
               description
                 "The reference to a configured condition.";
             }
             leaf action {
               type leafref {
                 path "/gnca/actions/action/name";
               }
               description
                 "The reference to a configured action.";
             }
           } // condition-action

           list cleanup-condition-action {
             key name;
             description
               "A list of Condition-Actions, which are configured
                conditions each with associated actions to be executed
                if the condition is evaluated to TRUE.
                This is the exception handler of this ECA, and is
                evaluated and executed in case any Action from the
                normal Condition-Action list was attempted and rejected
                by the server.";
             leaf name {
               type string;
```

```
              description
                "A string name uniquely identify a Condition-Action
                 within this ECA.";
            }
            leaf condition {
              type leafref {
                path "/gnca/conditions/condition/name";
              }
              description
                "The reference to a configured condition.";
            }
            leaf action {
              type leafref {
                path "/gnca/actions/action/name";
              }
              description
                "The reference to a configured action.";
            }
          } // cleanup-condition-action

          action start {
            description
              "Start to execute this ECA.";
          }
          action stop {
            description
              "Stop the execution of this ECA.";
          }
          action pause {
            description
              "Pause the execution of this ECA.";
          }
          action resume {
            description
              "Resume the execution of this ECA.";
          }
          action next-action {
            description
              "Resume the execution of this ECA to complete the next
               action.";
          }
          list execution {
            key id;
            config false;
            description
              "A list of executions that this ECA has completed,
               are currently running, and will start in the scheduled
               future.";
```

```
             leaf id {
               type uint32;
               description
                 "The ID to uniquely identify an execution of the ECA.";
             }
             leaf oper-status {
               type enumeration {
                 enum completed {
                   description "Completed with no error.";
                 }
                 enum running {
                   description "Currently with no error.";
                 }
                 enum sleeping {
                   description "Sleeping because of time schedule.";
                 }
                 enum paused {
                   description "Paused by the operator.";
                 }
                 enum stoped {
                   description "Stopped by the operator.";
                 }
                 enum failed {
                   description "Failed with errors.";
                 }
                 enum error-handling {
                   description
                     "Asking the operator to handle an error.";
                 }
               }
               description
                 "The running status of the execution.";
             }
             leaf start-time {
               type yang:date-and-time;
               description
                 "The time when the ECA started.";
             }
             leaf stop-time {
               type yang:date-and-time;
               description
                 "The time when the ECA completed or stopped.";
             }
             leaf next-scheduled-time {
               type yang:date-and-time;
               description
                 "The next time when the ECA is scheduled to resume.";
             }
```

```
            leaf last-condition-action {
              type leafref {
                path "../../condition-action/name";
              }
              description
                "The reference to a condition-action last executed
                 or being executed.";
            }
            leaf last-condition {
              type leafref {
                path "../../condition-action/condition";
              }
              description
                "The reference to a condition last executed or being
                 executed.";
            }
            leaf last-action {
              type leafref {
                path "../../condition-action/action";
              }
              description
                "The reference to aa action last executed or being
                 executed.";
            }
            leaf last-cleanup-condition-action {
              type leafref {
                path "../../cleanup-condition-action/name";
              }
              description
                "The reference to a cleanup-condition-action last
                 executed or being executed.";
            }
          }
        }
      } // ecas

      container eca-scripts {
        description
          "Container of ECA Scripts.";
        list eca-script {
          key script-name;
          description
            "A list of ECA Script.";
          leaf script-name {
            type string;
            description
              "A string name to uniquely identify an ECA Script.";
          }
```

```
          list eca {
            key eca-name;
            description
              "A list of ECAs contained in this ECA Script.";
            leaf eca-name {
              type leafref {
                path "/gnca/ecas/eca/name";
              }
              description
                "The reference to a configured ECA.";
            }
          }
        }
      } // eca-scripts

      leaf running-script {
        type leafref {
          path "/gnca/eca-scripts/eca-script/script-name";
        }
        description
          "The reference to the ECA script that is currently running.";
      }
    }
  }
  <CODE ENDS>
```

8. IANA Considerations

TBD.

9. Security Considerations

TBD.

10. Acknowledgements

The authors would like to thank Joel Halpern and Robert Wilton for their helpful comments and valuable contributions.

11. References

11.1. Normative References

[I-D.clemm-netconf-push-smart-filters-ps]
Clemm, A., Voit, E., Liu, X., Bryskin, I., Zhou, T., Zheng, G., and H. Birkholz, "Smart filters for Push Updates - Problem Statement", draft-clemm-netconf-push-smart-filters-ps-00 (work in progress), October 2017.

[I-D.ietf-teas-yang-te-topo]
Liu, X., Bryskin, I., Beeram, V., Saad, T., Shah, H., and O. Dios, "YANG Data Model for Traffic Engineering (TE) Topologies", draft-ietf-teas-yang-te-topo-14 (work in progress), February 2018.

11.2. Informative References

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[I-D.ietf-netconf-subscribed-notifications]
Voit, E., Clemm, A., Prieto, A., Nilsen-Nygaard, E., and A. Tripathy, "Custom Subscription to Event Streams", draft-ietf-netconf-subscribed-notifications-09 (work in progress), January 2018.

[I-D.ietf-netconf-yang-push]
Clemm, A., Voit, E., Prieto, A., Tripathy, A., Nilsen-Nygaard, E., Bierman, A., and B. Lengyel, "YANG Datastore Subscription", draft-ietf-netconf-yang-push-14 (work in progress), February 2018.

Authors' Addresses

Igor Bryskin
Huawei Technologies

EMail: Igor.Bryskin@huawei.com

Xufeng Liu
Jabil

EMail: Xufeng_Liu@jabil.com

Alexander Clemm
Huawei

EMail: ludwig@clemm.org

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de

Tianran Zhou
Huawei

EMail: zhoutianran@huawei.com