Abstract

This document describes a YANG data model for SUPA (Simplified Use of Policy Abstractions) ECA (Event-Condition-Action) policies using policy abstractions defined in [I-D. strassner-supa-generic-policy-info-model]. The EPDM (ECA policy data model) is refined from SGPIM (SUPA Generic Policy Information Model) and EPRIM (ECA Policy Rule Information Model) to be applied to deliver various management policies for controlling managed entities throughout the service development and deployment lifecycle. The generic ECA policy data model could be augmented by additional YANG data modules modeling and configuring policy-related protocols and functions. Reusability as the major advantage of this approach can be realized. The policy data model described in this document provides common building blocks for such 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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 26, 2016.

Copyright Notice

Copyright (c) 2014 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 ..................................................3
2. Conventions used in this document .............................4
   2.1. Tree Diagrams ............................................4
3. SUPA Policy Modules Top Level Design ..........................5
   3.1. supa-policy-target Design for ECA policy data model ......6
   3.2. ECA Policy Data Model Design .............................6
      3.2.1. supa-ECA-component sub class ........................7
   3.3. supa-policy-statement Design for ECA policy data model ...8
      3.3.1. supa-entity sub class ...............................8
      3.3.2. supa-script sub class ...............................9
   3.4. event-list sub class .....................................9
   3.5. condition-list sub class ................................10
   3.6. action-list sub class ...................................10
4. Generic ECA Policy Data Model ................................11
   4.1. Abstract Generic ECA Policy Data Model Hierarchy ......11
   4.2. SUPA Generic ECA Policy Data Model in YANG Module ......13
5. ECA Policy Data Model Example .................................20
   5.1. Redefine the supa-policy-target ..........................21
   5.2. Define the supa-ECA-component ............................21
   5.3. Define Event, Condition and Action clause ...............22
6. Specific ECA Policy Data Model for service flow policy ......24
1. Introduction

As defined in [I-D. strassner-supa-generic-policy-info-model], policies either be used in a stand-alone policy rule or aggregated into policy composite to perform more elaborate functions. The SUPA policy is tree-structured and can be embedded into hierarchal model.

In SUPA framework, the EPRIM is a set of subclasses that specialize the concepts defined in the SGPIM for representing the components of a Policy that uses ECA semantics. Note that, the information model is independent of data repository, data definition language, query language, implementation language, and protocol. While the ECA policy has to be defined with data repository, data definition language, query language, implementation language, and protocol.

In this way, an ECA policy data model defines:

- An event or a set of events that trigger the evaluation of policy: This is the trigger for the service management application to evaluate if a policy needs to be applied. For example a user action to provision a new VPN service can be an event.

- A set of conditions that need to be satisfied for the policy to be applicable: This enables service management to select the right policy by validating the conditions against the current network state.

- A set of actions that should be triggered as part of the policy execution: This enables the service management to provision the service.

This document introduces YANG [RFC6020] [RFC6021] data models for SUPA configuration. Such models can facilitate the standardization for the interface of SUPA, as they are compatible to a variety of protocols such as NETCONF [RFC6241] and [RESTCONF]. Please note that in the context of SUPA, the term "application" refers to an
operational and management applications employed, and possibly implemented, by an operator.

With respect to the scope, defining an information model for the policy exchange between the policy manager and policy agent and a corresponding data model based on yang to support specific DDC service use case is initial goal. The protocol specific aspects are deferred to respective implementations. Also certain foundational concepts of the model are intentionally left open to enable future extension.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

2.1. Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- Each node is printed as:

  \(<status> <flags> <name> <opts> <type>\)

  \(<status>\) is one of:
  + for current
  x for deprecated
  o for obsolete

  \(<flags>\) is one of:
  rw for Read/Write
  ro for ReadOnly
  -x for rpcs (remote procedure calls)
  -n for notifications

  \(<name>\) is the name of the node

  - If the node is augmented into the tree from another module, its name is printed as \(<prefix>:\<name>\).
3. SUPA Policy Modules Top Level Design

In this section, a generic ECA policy data model is defined with SGPIM to specify the top level sub-class. The SUPA policy is constructed hierarchically with possible extension at each leaf node. According to SGPIM framework, a supa-policy MUST have at least one supa-policy-statement that is used to define the content of the policy.

As shown in figure 2, the top level design policy data model is:

- supa-policy: The root of the SUPA generic ECA policy data model
- supa-policy-target: The managed object that a supa-policy monitors and/or controls the state of. The target will be the where the policy will be worked on, including domain, subnet and so on. Also the managed object will be specified such as a VPN, flow, link and so on. This class is specified by the user as the scope and the object needs to be told.
- supa-policy-atomic: A Policy that can be used in a stand-alone manner, and hierarchic policy and composite policy has not been taken into account in this document. Here the atomic means there is only one ECA policy rule in the policy data model. The major advantage of this design fashion is the separation of policy data and how to manage these policy data into policy rules. That means, only change the supa-policy-atomic here can generate new policy without redefine all the policy data.
- supa-policy-statement: It is used to define the content of a supa-policy, all the event, condition and action clauses are defined here. This part will not be affected by the changing of policy structures or designs.
3.1. supa-policy-target Design for ECA policy data model

The supa-policy-target target defines the working object of the policy. More specifically, the scope of the policy will be worked on and the instance that the policy will be worked on. This part should be the input from policy user and is not part of ECA policy itself. E.g., if the bandwidth of a flow of voice stream reaches threshold, more bandwidth will be assigned to guarantee the voice service. Here, which flow is for voice service and will be adjusted is specified by the user as the input of the ECA policy. So some kind of template is needed here to allow users to provide information of the working object of the policy.

As shown in figure 3, four attributes of supa-policy-target are defined by the user to specify the working scope of the policy. Then the instance defines the specific work object of the policy, such as a VPN, a flow, a link and so on. Obviously, VPN, flow or link needs different elements to be indicated, so it is can be defined with a "augment" statement to indicate the working object. E.g., if user wants to work on a flow, corresponding elements will be defined within "flow" case.

```
+--rw supa-policy-target
    +--rw profileType? string
    +--rw asDomainName? string
    +--rw adminSubnetwork? string
    +--rw businessTypeName? string
    +--rw instanceName
        +--rw instanceElement? empty
```

Figure 3: The snippet of supa-policy-target

3.2. ECA Policy Data Model Design

A supa-ECA-policy-rule, is a subclasses of the supa-policy-atomic class. Therefore, it can be used as part of a hierarchy of
Policies or in a stand-alone manner. The EPRIM specializes the supa-policy-atomic class to create a supa-ECA-policy-rule; it also specializes the supa-policy class to create a supa-ECA-component, and the supa-policy-statement to create corresponding clauses. The supa-ECA-policy-rule uses the rest of the SGPIM infrastructure to define a complete Policy model according to ECA semantics.

```
+--rw supa-policy-atomic
 |   +--rw supa-ECA-policy-rule
 |   |   |   |   +--rw supa-ECA-component
```

Figure 4: The snippet of supa-policy-atomic with ECA policy rule

-A supa-ECA-policy-rule is defined as a subclass of the SGPIM supa-policy-atomic class. All the related information of the ECA policy are defined here with some basic attributes of the policy and the supa-ECA-component sub class.

-A supa-ECA-component is one of core parts of the policy data model; it defines how the event, condition and action clauses are integrated into one working policy. Note that supa-ECA-component does not define the content of the policy itself but the structure as well the association of each policy statement clauses.

3.2.1. supa-ECA-component sub class

The principal subclasses of supa-policy-component that are defined in this version of this document are supa-policy-events, supa-policy-conditions, and supa-policy-actions. Each of the sub classes take care the event, condition and action part of the ECA policy respectively. The snippet of supa-ECA-component sub class is shown in figure 5.

```
+--rw supa-ECA-component
    +--rw supa-policy-events
    |   +--rw has-policy-events?   boolean
    +--rw supa-policy-conditions
    |   +--rw has-policy-conditions?   boolean
    |   |   +--rw conjunctive-type?   enumeration
    +--rw supa-policy-actions
    |   +--rw action-execution?    enumeration
```

Figure 5: The snippet of supa-ECA-component objects
The supa-policy-events sub class has one leaf to specify whether the policy has an event statement. If TRUE, the policy will take the event clause defined in the supa-policy-statement class.

The supa-policy-conditions sub class defines two things: one, does the policy have conditions, similar to the events part; two, if more than one conditions, how all the conditions are integrated into one single statement. Note that the ECA policy only makes the evaluation of condition statement once. So all the condition clauses needs to be integrated into one statement connected via AND or OR operator. Conjunctive-type defines use AND or OR operator to connect condition clauses.

The supa-policy-actions sub class defines how the action clause defined in supa-policy-statement will be executed.

3.3. supa-policy-statement Design for ECA policy data model

This is a mandatory abstract class that separates the representation of a supa-policy from its implementation. This abstraction is missing in [RFC3060], [RFC3460]. Basically, all the policy statements are defined here as clauses. SUPA use three types of mechanisms to define policies, entity, script template and Boolean clause. The statement has three part, event-list, condition-list and action-list, each has one or more clauses.

-supa-entity, which is a mechanism to directly use existing defined object as the input of event; this is described in more detail in Section 3.3.1.

-supa-script, which is a mechanism to directly encode the content of the supa-policy-statement into a script template and needs further execution which is out of SUPA; this is described in more detail in Section 3.3.2.

3.3.1. supa-entity sub class

This is a mandatory class that specializes a supa-policy-statement. It is defined that then event object can use the predefined object in existing module.

-entity can refer to an existing leaf node defined by other module. An example will be given in the following section.
The supa-entity has been defined as a "grouping" to improve reusability.

3.3.2. supa-script sub class

This is a mandatory concrete class that specializes (i.e., is a subclass of) a supa-policy-statement. It defines a generalized extension scripting mechanism for representing supa-policy-statement that has not been modeled with other supa policy objects. Rather, the Policy Clause is directly encoded into script template and then been executed in the network management function/controller.

-supap-script-content defines the content of this script template. It works with another attribute of the supa-script class, called supa-script-type, which defines how to interpret this script. These two attributes form a tuple, and together enable a machine to understand the script and know how to execute the script. Note that, the scripting approach is to improve the logic expression without defining new logic terminologies. Anything supported by script being used can be accommodated by SUPA.

-supap-script-type defines the type of this script being used. It works with another attribute of the supa-script class, called supa-script-content, which defines the content (i.e., the value) of the script template.

The supa-script has been defined as a "grouping" to improve reusability as the event and condition statement can both use the script template.

3.4. event-list sub class

All the event clauses are defined here with either encoded clause or Boolean clause. As shown in figure 8, each event can only be and must one type of the two clauses. Each event clause is defined
by calling the predefined two types of clauses in a "choice" statement.

```
+--rw event-list
    +--rw event-name
    +--rw (eventType)?
        +--:(entity)
            |  +--rw entity?  empty
        +--:(script)
            |  +--rw supa-script-type?  scriptType
            |  +--rw supa-script-content
```

Figure 8: The snippet of event-list sub class

3.5. condition-list sub class

All the condition clauses are defined using a "augment" statement. If there is more than one condition clause, just simply add more "container" to define more condition clause.

```
+--rw condition-list
    +--rw condition-name  empty
```

Figure 9: The snippet of condition-list sub class

3.6. action-list sub class

The action-list sub class defines all the action clauses those will be executed while the condition statement is being evaluating as TRUE. Since the action can only be defined by users as each action may have different attributes and elements to configure, the predefined structures and statements will not help. Not only the value of the leaf but also the number of leafs will depend on the type of actions. As shown in figure 10, here a "augment" statement is designed to keep the structure of the action statement stable while allows extensibility. The user can define new action by adding more case statement with self-defined element and statement structure without affecting existing one.

```
+--rw action-list
    +--rw actionName
        +--rw actionElement?  empty
```

Figure 10: The snippet of action-list sub class
4. Generic ECA Policy Data Model

4.1. Abstract Generic ECA Policy Data Model Hierarchy

Figure 11 shows the structure of abstract SUPA Generic ECA policy data model.
module: ietf-supapolicy
  +--rw supapolicy
     +--rw supapolicy-name?     string
     +--rw supapolicy-priority?  uint8
     +--rw supapolicy-validity-period
       +--rw start?           yang:date-and-time
       +--rw end?            yang:date-and-time
       +--rw duration?       uint32
       +--rw periodicity?     enumeration
     +--rw supapolicy-target
       +--rw profileType?     string
       +--rw asDomainName?     string
       +--rw adminSubnetwork?  string
       +--rw businessTypeName? string
     +--rw instance
     +--rw supapolicy-atomic
       +--rw supapolicy-atomic-rule
         +--rw policy-rule-deploy-status?   enumeration
         +--rw policy-rule-exec-status?     enumeration
         +--rw supapolicy-component
           +--rw supapolicy-events
             +--rw has-policy-events?   boolean
           +--rw supapolicy-conditions
             +--rw has-policy-conditions?   boolean
             +--rw conjunctive-type?     enumeration
           +--rw supapolicy-actions
             +--rw action-execution?   enumeration
     +--rw supapolicy-statement
     +--rw event-list
       +--rw event-name
         +--rw (eventType)?
           +--:(entity)
             +--rw entity?     empty
           +--:(script)
             +--rw supapolicy-script-type? scriptType
             +--rw supapolicy-script-content
     +--rw condition-list
     +--rw action-list

Figure 11: The structure of abstract SUPA Generic ECA policy data model
4.2. SUPA Generic ECA Policy Data Model in YANG Module

<CODE BEGINS> file "ietf-eca-policy@2015-10-10.yang"

module ietf-eca-policy {
    // replace with IANA namespace when assigned
    prefix policy;

    import ietf-yang-types {
        prefix yang;
    }

    organization "IETF";
    contact
        "Editor: Maoke Chen";

    description
        "This YANG module defines a component that describing
        the generic ECA policy data model refining from SGPIM and
        EPRIM.

    Terms and Acronyms
    ";

    revision 2015-08-25 {
        reference "";
    }

    container supa-policy{
        description
            "This defines a Generic ECA policy data model ";
        leaf supa-policy-name {
            type string;
            description
                "The name of the policy";
        }
        leaf supa-policy-priority {
            type uint8;
            description
                "The priority of the defined policy";
        }
        container supa-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";
}
}
container supa-policy-target {
    description "SUPAPolicyTarget is an abstract class that defines a
    set of managed objects that may be affected by the
    actions of a SUPAPolicyStatement."
    leaf profileType {
        type string;
        description "Which profile the policy will be worked on";
    }
    leaf asDomainName {
        type string;
        description "Which domain the policy will be worked on";
    }
    leaf adminSubnetwork {
        type string;
        description "Which subnet the policy will be worked on";
    }
    leaf businessTypeName {
        description "Which business type the policy will be worked on";
    }
}
type string;
description
  "Which business the policy will be worked on";
}

container instance {
  description
  "Which instance the policy will be worked on? E.g., 
a VPN, a flow or a link";
}

container supa-policy-atomic {
  description
  "Define a atomic ECA policy rule";
  container supa-ECA-policy-rule {
    description
    "SUPA policy atomic defines a standalone policy
    rule.";
    leaf policy-rule-deploy-status {
      type enumeration {
        enum 0{
          description "undefined";
        }
        enum 1{
          description "deployed and enabled";
        }
        enum 2{
          description "deployed and in test";
        }
        enum 3{
          description "deployed but not enabled";
        }
        enum 4{
          description "ready to be deployed";
        }
        enum 5{
          description "not deployed";
        }
      }
      description
      "The deploy status of the policy.";
    }
    leaf policy-rule-exec-status {
      type enumeration {
        enum 0{
          description "undefined";
        }
      }
    }
  }
}

enum 1{
    description
    "executed and SUCEEEDED (operational mode)";
}

enum 2{
    description
    "executed and FAILED (operational mode)";
}

enum 3{
    description
    "currently executing (operational mode)";
}

enum 4{
    description
    "executed and SUCEEEDED (test mode)";
}

enum 5{
    description
    "executed and FAILED (test mode)";
}

enum 6{
    description
    "currently executing (test mode)";
}

description
"The executing status of the policy.";
}

container supa-ECA-component{
    description
    "The component defines how the event, condition
    and action clauses are constructed into policy";
}

container supa-policy-events {
    description
    "An event or a set of events that trigger the
    evaluation of policy: This is the trigger for
    the service management application to
    evaluate if a policy needs to be applied. For
    example a user action to provision a new VPN
    service can be an event.";
}

leaf has-policy-events {
    type boolean;
    description
    "Whether the policy has an event";
}
}

container supa-policy-conditions {

description
"A set of conditions that need to be satisfied for the policy to be applicable: This enables service management to select the right policy by validating the conditions against the current network state."

leaf has-policy-conditions {
  type boolean;
  description
  "Whether the policy has a condition?";
}

leaf conjunctive-type {
  type enumeration {
    enum 0 {
      description "AND: all the conditions must be matched";
    }
    enum 1 {
      description "OR: one or more of the conditions are matched";
    }
  }
  description
  "Define how the condition clauses will be conjuncted, AND or OR";
}

container supa-policy-actions {
  description
  "A set of actions that should be triggered as part of the policy execution: This enables the service management to provision the service."
  leaf action-execution {
    type enumeration{
      enum 0 {
        description "Single: execute one action";
      }
      enum 1 {
        description "Sequenced: execute actions one by one in sequence";
      }
    }
    description
    "How the actions will be executed";
  }
}
container supa-policy-statement {
  description
  "The individual policy statement clauses.";
  /*typedef scriptTemplate {
   type string;
   description
   "The script defined in the YANG model can be sent to the policy engine to execute. Here is the content of the script template";
  }*/
  typedef scriptType {
    type enumeration{
      enum 0{
        description
        "Python";
      }
      enum 1{
        description
        "Perl";
      }
      enum 2{
        description
        "Javascript";
      }
    }
    description
    "Here is the type of the script to be executed. E.g., Python, Perl";
  }
  grouping supa-entity{
    leaf entity{
      type empty;
      description
      "The path of the reference node needs to be specified when using this type to define event or condition";
    }
    description
    "Use predefined object to specify the event and condition";
  }
}
description
"The script can be used to specify the event and condition clauses. The script can be executed in the policy engine.";
leaf supa-script-type {
    type scriptType;
    description
    "Use which type of script, such as Python, Perl and so on.";
}
anyxml supa-script-content {
    description
    "The script template that will be sent to the policy engine to specify the event or condition clause";
}

container event-list{
    description
    "The event clauses. Each one can be a predefined network entity, a script or boolean clause,";
    container event-name {
        description
        "The event clause of the policy.";
        choice eventType{
            description
            "User define to use which type event.";
            case entity{
                uses supa-entity;
            }
            case script {
                uses supa-script;
            }
        }
    }
}

container condition-list{
    description
    "The condition clauses. Each one can be a predefined network entity, a script or boolean clause, and conjuncted by AND or OR";
}

container action-list{
    description

"Defines actions clause here. Each action has unique attributes so a choice statement here to allow self-defined action without changing."

5. ECA Policy Data Model Example

This section will provide one example to show how to use this generic ECA policy data model to generate specific policy data model a service flow policy that can be mapped into configurations.

The generic ECA policy data model contains no configuration information and lack of action elements, it cannot be mapped into configuration such as XML instance by just filling the value of the leaves. In order to make a working ECA policy, the user needs to define some part of the generic policy data model and fill in some of the leaves but do not need change the structure. Basically, instantiate a generic ECA policy data model into a specific ECA policy data model only needs adding some leaves and specify some values.

More specifically, for a service flow policy "If the bandwidth of a voice stream flow exceeds 8Mbps, change the CIR to 20Mbps to guarantee the voice service", how to use generic ECA data model to generate a working data model to deploy this policy? Instructions with data model will be given in next few sections. The major steps are:

1. Fill in the basic attributes of the policy, such as name, priority, valid period and so on.

2. Redefine the supa-policy-target

3. Specify the leaf value in supa-ECA-component to define how the policy clauses in supa-policy-statement will be integrated into a policy rule.

4. Define the event clause, condition clause and action clause using augment statement.
5.1. Redefine the supa-policy-target

The first step is to redefine the target of the policy. As shown in figure 12, the user fills in all the attributes to define the working scope of the policy such as the profile, domain and subnet. Then, in order to tell the system which flow to be worked on, the user will define a flow filter with possible elements to get the right flow. Here, dscp value, source IP, destination IP, source port and destination port are attributes needed to indicate a flow. For this working policy, user also fills in the value of each leaf, dscp = 5, src-ip-addr = 10.111.10.1, dst-ip-addr = 10.112.10.1, src-port = 8080 and dst-port = 8090. After all this, the desired voice stream flow (dscp=5) will be selected as policy target.

augment /supa:supa-policy/supa:supa-policy-target/supa:instance:
  +--rw flowFilter
    +--rw dscp?          uint32
    +--rw src-ip-addr?   string
    +--rw dst-ip-addr?   string
    +--rw src-port?      uint32
    +--rw dst-port?      uint32

Figure 12: The snippet of specific policy target

Note that, the supa-policy-target is reusable and extensible as the user can add more instance into case statement without affecting existing one. E.g., user could also add VPN with elements such as VPNName, source IP and destination IP.

5.2. Define the supa-ECA-component

In order to define how the policy clauses are organized and associated into one policy, the user needs to fill in all the leaf value in supa-ECA-component. As shown in figure 13, the corresponding value will be filled in. The policy has event and condition, but only one condition. And the policy will execute a single action if the condition being evaluated as TRUE.
5.3. Define Event, Condition and Action clause

The core part of ECA policy is the policy statement as individual clauses. For the example policy, event is the flow entry, condition is that its bandwidth >= 8M, and action is to do CAR with CIR = 20M. As shown in figure 14, the user first needs to choose the type of event and condition clause. In this case, event clause should be entity and use predefined flow object. Condition clause should be script template to send the bandwidth >= 8M script to the controller. Then design the event and condition clause by filling in the leaf. Note that the choice of clause type does not include in YANG data model but will be accomplished in NETCONF via <edit-config> operation.

More specifically, the path of leafref of entity is 
"/supa:flows/supa:flow/supa:flowId", which pointed to anther module. We suppose that module has already defined the object "flow".

For the condition part, the condition elements are defined using augment statement. "bandwidth" and "threshold" leaf are added into "condition-bandwidth" class.
augment /supa:supa-policy/supa:supa-policy-statement/supa:condition-list:
    +++rw condition-bandwidth
    +++rw bandwidth?  uint32
    +++rw threshold?  uint32

augment /supa:supa-policy/supa:supa-policy-statement/supa:action-list:
    +++rw action-redirect
    +++rw cir?  uint32
    +++rw pir?  uint32
    +++rw Cbs?  uint32
    +++rw Pbs?  uint32

Figure 14: The snippet of specific policy statement

The design of action clause is more complicated as different action has different number and type of attributes to be specified. And the action is only valid when the condition is evaluated as TRUE. So here a "when" statement is used to do the augment. The "when" expression is the Xpath expression to evaluate if the predefined "bandwidth" exceeds the "threshold".

Finally, with the refined ECA policy data model as shown in section 6, with working policy "If the bandwidth of a voice stream flow exceeds 8Mbps, change the CIR to 20Mbps to guarantee the voice service" can be mapped into XML instance.
6. Specific ECA Policy Data Model for service flow policy

6.1. SUPA specific ECA Policy Data Model in YANG Module

<CODE BEGINS> file "ietf-supap-service-flow-policy@2015-10-10.yang"
module ietf-supap-service-flow-policy {
    +"policy";
    // replace with IANA namespace when assigned
    prefix flow;

    /*import ietf-yang-types {
        prefix yang;
    }*/
    import ietf-inet-types {
        prefix inet;
    }
    import ietf-eca-policy {
        prefix supa;
    }
    organization "IETF";
    contact
        "Editor: Maoke Chen";

    description
        "This YANG module defines a component that describing
        the specific ECA policy data model for service flow
        refining from SGPIM and EPRIM.
        Terms and Acronyms"
    ;

    revision 2015-08-25 {
        reference "";
    }

    //flow filter parameters for a flow
    augment "/supa:supa-policy/supa:supa-policy-
    target/supa:instance"
    {
        description
            "$Use the base ECA policy model to define service flow
            policy.";
        container flowFilter{
            description
                "$Use the base ECA policy model to define service flow
                policy.";
        }
    }

    /*import ietf-yang-types {
        prefix yang;
    }*/
    import ietf-inet-types {
        prefix inet;
    }
    import ietf-eca-policy {
        prefix supa;
    }
    organization "IETF";
    contact
        "Editor: Maoke Chen";

    description
        "This YANG module defines a component that describing
        the specific ECA policy data model for service flow
        refining from SGPIM and EPRIM.
        Terms and Acronyms"
    ;

    revision 2015-08-25 {
        reference "";
    }

    //flow filter parameters for a flow
    augment "/supa:supa-policy/supa:supa-policy-
    target/supa:instance"
    {
        description
            "$Use the base ECA policy model to define service flow
            policy.";
        container flowFilter{
            description
                "$Use the base ECA policy model to define service flow
                policy.";
        }
    }

"Self defined flow filter to specify the policy target."
leaf dscp {
    type uint32;
    description "dscp value of the indicated flow";
}
leaf src-ip-addr{
    type inet:ipv4-address;
    description "source ip addresses of the flow";
}
leaf dst-ip-addr{
    type inet:ipv4-address;
    description "destination ip addresses of the flow";
}
leaf src-port{
    type uint32;
    description "source port number of the flow";
}
leaf dst-port{
    type uint32;
    description "destination port number of the flow";
}
// Add condition clauses into the condition list
augment "/supa:supa-policy/supa:supa-policy-statement/supa:"
+"condition-list" {
    description "Define the condition clause with parameters.";
    container condition-bandwidth{
        description "Define the bandwidth with threshold value.";
        leaf bandwidth{
            type uint32;
            description "The flow bandwidth, unit is Mbps";
        }
        leaf threshold{
            type uint32;
            description "The threshold to trigger the action.";
        }
    }
}
//action node is depending on the condition
augment "/supa:supa-policy/supa:supa-policy-
statement/supa:action-
"list" {
  container action-redirect{
    when "/ietf-supu-policy/supa-policy/supa-policy-
statement//
  +"condition-list/condition-bandwidth/bandwidth>/ietf-
supa-" +"policy/supa-policy/supa-policy-statement/condition-
list/"
  +"condition-bandwidth/threshold"{
      description
        "If the condition has been evaluated as TRUE, then
the action is added to the policy.";
    }
  }
  leaf cir {
    type uint32;
    description
      "Committed information rate";
  }
  leaf pir {
    type uint32;
    description
      "Peak information rate";
  }
  leaf Cbs {
    type uint32;
    description
      "Committed burst size";
  }
  leaf Pbs {
    type uint32;
    description
      "Peak burst size";
  }
  description
    "Define the action clause in the policy statement.";
}
description
  "The augment of the action node is only valid when the
condition is evaluated as TRUE";
7. Security Considerations
    TBD

8. IANA Considerations
    This document has no actions for IANA.

9. Contributor List
    Andy Bierman
    YumaWorks, Inc.
    Email: andy@yumaworks.com

10. Acknowledgments
    This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Liya Zhang, John Strassner, Juergen Schoenwaelder.

11. References

11.1. Normative References


11.2. Informative References

Abstractions (SUPA)*, draft-strassner-supageneric-policy-infomodel-01 (work in progress), May 2015.


Authors’ Addresses

Maoke Chen
BBIX, Inc.
Tokyo Shiodome Building, Higashi-Shimbashi 1-9-1
Minato-ku, Tokyo, 105-7310, Japan
Email: maoke@bbix.net

Luis M. Contreras
Telefonica I+D
Ronda de la Comunicacion, Sur-3 building, 3rd floor
Madrid 28050, Spain
Email: luismiguel.contrerasmurillo@telefonica.com
URI: http://people.tid.es/LuisM.Contreras/

Michiaki Hayashi
KDDI R&D Labs. Inc.
2-1-15 Ohara, Fujimino, Saitama, Japan. 356-8502
Email: mc-hayashi@kddilabs.jp

Tina Tsou
Huawei Technologies
Email: Tina.Tsou.Zouting@huawei.com
Abstract

This document contains the Basic Network Policy and Filters (BNP IM) Data Model which provides a policy model that support an ordered list of match-condition-action (aka event-condition-action (ECA)) for multiple layers (interface, L1-L4, application) and other factors (size of packet, time of day). The actions allow for setting actions (QOS and other), decapsulation, encapsulation, plus forwarding actions. The policy model can be used with the I2RS filter-based RIB.

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 April 21, 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
This generic network policy provides a model to support an ordered list of routing policy or an ordered list of filter rule. Two examples of the ordered-based filters are the I2RS Filter-based RIBs, and another is flow-specification filters. The first section of this draft contains an overview of the policy structure. The second provides a high-level Yang module. The third contains the Yang module.

1.1. Definitions and Acronyms

INSTANCE: Routing Code often has the ability to spin up multiple copies of itself into virtual machines. Each Routing code instance or each protocol instance is denoted as Foo_INSTANCE in the text below.

NETCONF: The Network Configuration Protocol

PCIM - Policy Core Information Model

RESTconf - http programmatic protocol to access Yang modules
1.2. Antecedents this Policy in IETF

Antecedents to this generic policy are the generic policy work done in PCIM WG. The PCIM work contains a Policy Core Information Model (PCIM) [RFC3060], Policy Core Informational Model Extensions [RFC3460] and the Quality of Service (QoS) Policy Information Model (QPIM) ([RFC3644]) From PCIM comes the concept that policy rules which are combined into policy groups. PCIM also refined a concept of policy sets that allowed the nesting and aggregation of policy groups. This generic model did not utilize the concept of sets of groups, but could be expanded to include sets of groups in the future.

2. Generic Route Filters/Policy Overview

This generic policy model represents filter or routing policies as rules and groups of rules.

The basic concept are:

Rule Group

A rule group is is an ordered set of rules.

Rule

A Rule is represented by the semantics "If Condition then Action". A Rule may have a priority assigned to it.
Rule groups have the following elements:

- name that identifies the grouping of policy rules
- role - that is a combination of target resource (E.g. IPv4 FB-FIB filters) and a scope (read, read-write, write-only).
- list of rules

The rule has the following elements: name, order, status, priority, reference cnt, and match-action as shown in figure 2. The order indicates the order of the rule within the list. The status of the rule is (active, inactive). The priority is the priority within a specific order of policy/filter rules. A reference count (reftcnt) indicates the number of entities (E.g. network modules) using this policy. The generic rule match-action conditions have match operator, a match variable and a match value. The rule actions have an action operator, action variable, and an action value.
The generic rules can be included with other types of rules as figure 2 shows.

```
Figure 2 - Rule Group
+-------------------------------------+ (optional)
|             Rule Group              |....
+--------------------------------------+   :
*      *                   *        ^    :
|      |                   |        :....:
|      |                   |
|      |                   |
+------+   +-------------------+
| Name |   |     Rule_list     |
|      |   |                   |
+------+   +------|------------+
+----------------|-----------------+
|           rule                   |
+---|--+ +-|----+ +---|-------+ +-|----+
| Name | |rule  | | ECA       | |rule  |
+------+ |order | | match     | |status|
|number| |qos-actions | +------+
|       |fwd-actionx|
```

The generic match conditions are specific to a particular layer are refined by matches to a specific layer (as figure 3 shows), and figure 5’s high-level yang defines. The general actions may be generic actions that are specific to a particular layer (L1, L2, L3, service layer) or time of day or packet size. The qos actions can be setting fields in the packet at any layer (L1-L4, service) or encapsulating or decapsulating the packet at a layer. The fwd-actions are forwarding functions that forward on an interface or to a next-hop. The rule status is the operational status per rule.
Figure 3

```
+-------------+
| Match       |
| Condition   |
+-------------+
```

```
V       V       V
: L1    : L2    : L3    : Service : ...
: match : match : match : match : ...
```

4. BNP Generic Info Model in High Level Yang

Below is the high level inclusion

```
module:bnp-eca-policy
import ietf-inet
import ietf-interface
import ietf-i2rs-rib
import service-function-type prefix-sft
import service-function prefix-sf
import service-fucntion-chain prefix-sfc-sfc
```

Below is the high level yang diagram
module:i2rs-eca-policy
  +--i2rs-eca-policy
    +--rw rule-group* [group-name]
    +--rw group-name
    +--rw rule* [rule-name]
    +--rw rule-name string
    +--rw order unit16
    +--rw installer
    +--rw rule-match-act
      +--rw bnp-matches
        +--case: interface-match
        +--case: L1-header-match
        +--case: L2-header-match
        +--case: L3-header-match
        +--case: L4-header-match
        +--case: Service-header-match
        +--case: packet-size
          +--case: time-of-day
      +--rw bnp-action
        +--rw number-actions
          +--case interface-actions
            +--case L1-action
            +--case L2-action
            +--case L3-action
            +--case L4-action
            +--case service-action
      +--rw bnp-forward
        +--rw interface interface-ref
        +--rw next-hop rib-nexthop-ref
        +--rw route-attributes
          +--rw rib-route-attributes-ref
        +--rw fb-std-drop
      +--rw rule_status
        +--ro rules-status
        +--ro rule-inactive-reason
        +--ro rule-installer
        +--ro refcnt unit16

5. i2rs-eca-policy Yang module

    //<CODE BEGINS> file "i2rs eca-policy@2015-10-18.yang"

    module i2rs-eca-policy
    {
        namespace "urn:ietf:params:xml:ns:yang:i2rs-eca-policy";
        // replace with iana namespace when assigned
        prefix "i2rs-eca";

    }
// import some basic inet types

import ietf-inet-types { prefix "inet"; } // RFC6991
import ietf-interfaces { prefix "if"; }
import i2rs-rib { prefix "i2rs-rib"; }

// meta
organization
"IETF I2RS WG";

contact
"email: shares@ndzh.com
   email: russ.white@riw.com
   email: jeff.tantsura@ericsson.com
   email: linda.dunbar@huawei.com
   email: bill.wu@huawei.com";

description
"This module describes a basic network policy
model with filter per layer."

revision "2015-10-18" {
   description "initial revision";
   reference "draft-hares-i2rs-bnp-eca-policy-dm-01";
}

// interfaces - no identity matches

// L1 header match identities
identity l1-header-match-type {
   description
   "L1 header type for match ";
}

identity l1-hdr-sonet-type {
   description
   "L1 header sonet match ";
   base l1-header-match-type;
}

identity l1-hdr-OTN-type {
   description
   "L1 header OTN match ";
   base l1-header-match-type;
}

identity l1-hdr-dwdm-type {
description "L1 header DWDM match ";
base l1-header-match-type;

// L2 header match identities
identity l2-header-match-type {
    description "L2 header type for match ";
}

identity l2-802-1Q {
    description "L2 header type for 802.1Q match ";
    base l2-header-match-type;
}

identity l2-802-11 {
    description "L2 header type for 802.11 match ";
    base l2-header-match-type;
}

identity l2-802-15 {
    description "L2 header type for 802.15 match ";
    base l2-header-match-type;
}

identity l2-NVGRE {
    description "L2 header type for NVGRE match ";
    base l2-header-match-type;
}

identity l2-mpls {
    description "L2 header type for MPLS match ";
    base l2-header-match-type;
}

identity l2-VXLAN {
    description "L2 header type for VXLAN match ";
    base l2-header-match-type;
}

// L3 header match identities
identity l3-header-match-type {
  description "l3 header type for match";
}

identity l3-ipv4-hdr {
  description "l3 header type for IPv4 match";
  base l3-header-match-type;
}

identity l3-ipv6-hdr {
  description "l3 header type for IPv6 match";
  base l3-header-match-type;
}

identity l3-gre-tunnel {
  description "l3 header type for GRE tunnel match";
  base l3-header-match-type;
}

// L4 header match identities

identity l4-header-match-type {
  description "L4 header match types. (TCP, UDP, SCTP, etc.)";
}

identity l4-tcp-header {
  description "L4 header for TCP";
  base l4-header-match-type;
}

identity l4-udp-header {
  description "L4 header match for UDP";
  base l4-header-match-type;
}

identity l4-sctp-header {
  description "L4 header match for SCTP";
  base l4-header-match-type;
}

// Service header identities
identity service-header-match-type {
    description "service header
    match types: service function path
    (sf-path)), SF-chain, sf-discovery,
    and others (added here)";
}

identity sf-chain-meta-match {
    description "service header match for
    meta-match header";
    base service-header-match-type;
}

identity sf-path-meta-match {
    description "service header match for
    path-match header";
    base service-header-match-type;
}

identity rule-status-type {
    description "status
    values for rule: invalid (0),
    valid (1), valid and installed (2)";
}

identity rule-status-invalid {
    base rule-status-type;
}

identity rule-status-valid {
    base rule-status-type;
}

identity rule-status-valid-installed {
    base rule-status-type;
}

identity rule-status-valid-inactive {
    base rule-status-type;
}

grouping interface-match {
    description "interface
    has name, description, type, enabled
    as potential matches";

    leaf match-if-name {
        description "match on interface name";
        type if:interface-ref;
grouping interface-action {
  description
  "interface action up/down and enable/disable";
  leaf interface-up {
    description
    "action to put interface up";
    type boolean;
  }
  leaf interface-down {
    description
    "action to put interface down";
    type boolean;
  }
  leaf interface-enable {
    description
    "action to enable interface";
    type boolean;
  }
  leaf interface-disable {
    description
    "action to disable interface";
    type boolean;
  }
}

grouping L1-header-match {
  description
  "The Layer 1 header match includes any reference to L1 technology";
  // matches for OTN, SDH, DWDM
  choice l1-header-match-type {
    case l1-hdr-sonet-type {
      // sonet matches
    }
    case L1-hdr-OTN-type {
      // OTN matches
    }
    case L1-hdr-dwdm-type {
      // DWDM matches
    }
  }
}
grouping L1-header-actions {
    choice l1-header-match-type {
        case l1-hdr-sonet-type {
            // sonet actions
        }
        case L1-hdr-OTN-type {
            // OTN actions
        }
        case L1-hdr-dwdm-type {
            // DWDM actions
        }
    }
}

grouping L2-802-1Q-header {
    description "This is short-term 802.1 header match which will be replaced by reference to IEEE yang when it arrives. Qtag 1 is 802.1Q Qtag2 is 802.1AD";
    leaf vlan-present {
        description "Include VLAN in header";
        type boolean;
    }
    leaf qtag1-present {
        description "This flag value indicates inclusion of one 802.1Q tag in header";
        type boolean;
    }
    leaf qtag2-present{
        description "This flag indicates the inclusion of second 802.1Q tag in header";
        type boolean;
    }
    leaf dest-mac {
        description "IEEE destination MAC value from the header";
        type uint64;
        //change to uint48
    }
    leaf src-mac {
        description "IEEE source MAC from the header";
        type uint64;
        //change to uint48
    }
}
leaf vlan-tag {
  description "IEEE VLAN Tag from the header";
  type uint16;
}
leaf qtag1 {
  description "Qtag1 value from the header";
  type uint32;
}
leaf qtag2 {
  description "Qtag1 value from the header";
  type uint32;
}
leaf L2-ethertype {
  description "Ether type from the header";
  type uint16;
}

grouping L2-VXLAN-header {
  description "This VXLAN header may be replaced by actual VXLAN yang module reference";
  container vxlan-header {
    //vix outer mac header
    uses i2rs-rib:ipv4-header;
    leaf vxlan-network-id {
      description "VLAN network id";
      type uint32;
    }
  }
  //fix inner header here
}

grouping L2-NVGRE-header {
  description "This NVGRE header may be replaced by actual NVGRE yang module reference";
  container nvgre-header {
    uses L2-802-1Q-header;
    uses i2rs-rib:ipv4-header;
  }
}
leaf gre-version {
    description "L2-NVGRE GRE version";
    type uint8;
}

leaf gre,proto {
    description "L2-NVGRE protocol value";
    type uint16;
}

leaf virtual-subnet-id {
    description "L2-NVGRE subnet id value";
    type uint32;
}

leaf flow-id {
    description "L2-NVGRE Flow id value";
    type uint16;
    // uses L2-802-1Q-header;
}

grouping L2-header-match {
    description "The layer 2 header match includes any reference to L2 technology";
    choice l2-header-match-type {
        case l2-802-1Q {
            uses L2-802-1Q-header;
        }
        case l2-802-11 {
            // matches for 802.11 headers
        }
        case l2-802-15 {
            // matches for 802.1 Ethernet
        }
        case l2-NVGRE {
            // matches for NVGRE
            uses L2-NVGRE-header;
        }
        case l2-VXLAN-header {
            uses L2-VXLAN-header;
        }
        case l2-mpls-header {
            uses i2rs-rib:mpls-header;
        }
    }
}

grouping L2-header-actions {
description
"The layer 2 header match includes any reference to L2 technology";

choice l2-header-match-type {
  case l2-802-1Q {
    // actions for L2-802-1Q
  }
  case l2-802-11 {
    // actions for L2-802-11
  }
  case l2-802-15 {
    // actions 802.1 Ethernet
  }
  case l2-NVGRE {
    // actions for NVGRE
    leaf set-vsid {
      description
      "Boolean flag to set VSID in packet";
      type boolean;
    }
    leaf set-flowid {
      description
      "Boolean flag to set VSID in packet";
      type boolean;
    }
    leaf vsi {
      description
      "VSID value to set in packet";
      type uint32;
    }
    leaf flow-id {
      description
      "flow-id value to set in packet";
      type uint16;
    }
  }
  case l2-VXLAN-header {
    leaf set-network-id {
      description
      "flag to set network id in packet";
      type boolean;
    }
    leaf network-id {
      description
      "network id value to set in packet";
      type uint32;
    }
  }
}
case l2-mpls-header {
    leaf pop {
        description "Boolean flag to pop mpls header";
        type boolean;
    }
    leaf push {
        description "Boolean flag to push value into mpls header";
        type boolean;
    }
    leaf mpls-label {
        description "mpls label to push in header";
        type uint32;
    }
}

grouping L3-header-match {
    description "match for L3 headers";
    choice L3-header-match-type {
        case l3-ipv4-hdr {
            uses i2rs-rib:ipv4-header;
        }
        case l3-ipv6-hdr {
            uses i2rs-rib:ipv6-header;
        }
        case L3-gre-tunnel {
            uses i2rs-rib:gre-header;
        }
    }
}

grouping ipv4-encapsulate-gre {
    description "encapsulation actions for IPv4 headers";
    leaf encapsulate {
        description "flag to encapsulate headers";
        type boolean;
    }
    leaf ipv4-dest-address {
        description "Destination Address for GRE header";
        type inet:ipv4-address;
    }
    leaf ipv4-source-address {
        description "Source Address for GRE header";
        type inet:ipv4-address;
    }
}
grouping l3-header-actions {
    description "actions that can be performed on header";

    choice l3-header-act-type {
        case l3-ipv4-hdr {
            leaf set-ttl {
                description "flag to set TTL";
                type boolean;
            }
            leaf set-dscp {
                description "flag to set DSCP";
                type boolean;
            }
            leaf ttl-value {
                description "TTL value to set";
                type uint8;
            }
            leaf dscp-val {
                description "dscp value to set";
                type uint8;
            }
        }
        case l3-ipv6-hdr {
            leaf set-next-header {
                description "flag to set next routing header in IPv6 header";
                type boolean;
            }
            leaf set-traffic-class {
                description "flag to set traffic class in IPv6 header";
                type boolean;
            }
            leaf set-flow-label {
                description "flag to set flow label in IPv6 header";
                type boolean;
            }
            leaf set-hop-limit {
                type boolean;
            }
            leaf next-header {
                description "value to set in next header";
                type uint8;
            }
            leaf traffic-class {
                description "value for traffic class";
            }
        }
    }
}
升序类型 uint8;
}
leaf flow-label {
    description "value for flow label";
    type uint16;
}
leaf hop-limit {
    description "value for hop count";
    type uint8;
}

case L3-gre-tunnel {
    leaf decapsulate {
        description "flag to decapsulate packet";
        type boolean;
    }
}

grouping tcp-header-match {
    leaf source-port {
        description "source port match value";
        type uint16;
    }
    leaf dest-port {
        description "dest port match value";
        type uint16;
    }
    leaf sequence-number {
        description "sequence number match value";
        type uint32;
    }
    leaf ack-number {
        description "ack number match value";
        type uint32;
    }
}

grouping tcp-header-action {
    leaf set-source-port {
        description "flag to set source port value";
        type boolean;
    }
    leaf set-dest-port {
        description "flag to set source port value";
        type boolean;
    }
}
uses tcp-header-match;
}

grouping udp-header-match {
  leaf source-port {
    description "UDP source port match value";
    type uint16;
  }
  leaf dest-port {
    description "UDP destination port match value";
    type uint16;
  }
}

grouping udp-header-action {
  leaf set-source-port {
    description "flag to set UDP source port match value";
    type boolean;
  }
  leaf set-dest-port {
    description "flag to set UDP destination port match value";
    type boolean;
  }
  uses udp-header-match;
}

grouping sctp-chunk {
  leaf chunk-type {
    description "sctp chunk type value";
    type uint8;
  }
  leaf chunk-flag {
    description "sctp chunk type flag value";
    type uint8;
  }
  leaf chunk-length {
    description "sctp chunk length";
    type uint16;
  }
  leaf chunk-data-0 {
    description "byte zero of stcp chunk data";
    type uint32;
  }
}

grouping sctp-header-match {
  leaf source-port {

description "sctp header match source port value";
  type uint16;
}

leaf dest-port {
  description "sctp header match destination port value";
  type uint16;
}

leaf verification-tag {
  description "sctp header match verification tag value";
  type uint32;
  uses sctp-chunk;
}

grouping sctp-header-action {
  leaf set-source-port {
    description "set source port in sctp header";
    type boolean;
  }
  leaf set-dest-port {
    description "set destination port in sctp header";
    type boolean;
  }
  leaf set-chunk1 {
    description "set chunk value in sctp header";
    type boolean;
  }
  uses sctp-header-match;
}

grouping L4-header-match {
  choice l3-header-match-type {
    case l4-tcp-header {
      uses tcp-header-match;
    }
    case l4-udp-header {
      uses udp-header-match;
    }
    case l4-sctp {
      uses sctp-header-match;
    }
  }
}

grouping l4-header-action {
  choice L3-header-match-type {
    case l4-tcp-header {

uses tcp-header-action;
}
case l4-udp-header {
    uses udp-header-action;
}
case l4-sctp {
    uses sctp-header-action;
}
}

grouping service-header-match {
    choice service-header-match-type {
        case sf-chain-meta-match {
            // uses sfc-sfc:service-function-chain-grouping:service-function-chain;
        }
        case sf-path-meta-match {
            // uses sfc-spf:service-function-paths:service-function-path;
        }
    }
}

grouping service-header-actions {
    choice service-header-match-type {
        case sf-chain-meta-match {
            leaf set-chain {
                description "flag to set chain in sfc";
                type boolean;
            }
            //uses sfc-sfc:service-function-chain-grouping:service-function-chain;
        }
        case sf-path-meta-match {
            leaf set-path {
                description "flag to set path in sfc header";
                type boolean;
            }
            // uses sfc-spf:service-function-paths:service-function-path;
        }
    }
}

grouping rule_status {
    description "rule operational status";
    leaf rule-status {
        type string;
    }
    leaf rule-status-inactive {
description "description of why rule is inactive";
    type string;
}
leaf rule-status-installer {
    description "response on rule installed";
    type string;
}
leaf refcnt {
    description "reference count on rule.";
    type uint64;
}

grouping packet-size-match {
    description "packet size by layer
    only non-zero values are matched";
    leaf l1-size-match {
        description "L1 packet match size.";
        type uint32;
    }
    leaf l2-size-match {
        description "L2 packet match size.";
        type uint32;
    }
    leaf l3-size-match {
        description "L3 packet match size.";
        type uint32;
    }
    leaf l4-size-match {
        description "L4 packet match size.";
        type uint32;
    }
    leaf service-meta-size {
        description "service meta info match size.";
        type uint32;
    }
    leaf service-meta-payload {
        description "service meta-play match size";
        type uint32;
    }
}

grouping time-day-match {
    //matches for time of day;
}

grouping eca-matches {

description "ECA matches";
uses interface-match;
uses L1-header-match;
uses L2-header-match;
uses L3-header-match;
uses L4-header-match;
uses service-header-match;
uses packet-size-match;
uses time-day-match;
}

grouping eca-qos-actions {
    description "ECA set or change packet Actions";
    leaf cnt-actions {
        description "count of ECA actions";
        type uint32;
    }
    // actions may be added for interface, L1, L2, L3, and L4 and service forwarding.
}

grouping ip-next-fwd {
    leaf rib-name {
        description "name of RIB";
        type string;
    }
    leaf next-hop-name {
        description "name of next hop";
        type string;
    }
}

grouping eca-fwd-actions {
    description "ECA forwarding actions";
    leaf interface-fwd {
        description "name of interface to forward on";
        type if:interface-ref;
    }
    uses i2rs-rib:nexthop;
    uses ip-next-fwd;
    leaf drop-packet {
        description "drop packet flag";
        type boolean;
    }
}

container bnp-ecap-policy-set {
description "main bnp ecap policy";

container policy-groups {
  list rule-group {
    key "group-name";
    description "groups of ECA rules";

    leaf group-name {
      description "name of group of rules";
      type string;
    }

    list rule {
      key "rule-name";
      description "ECA rules";
      leaf rule-name {
        description "name of rule";
        type string;
      }

      leaf order-id {
        description "Number of order in ordered list (ascending)";
        type uint16;
      }

      leaf installer {
        description "Id of I2RS client that installs this rule.";
        type string;
      }

      uses eca-matches;
      uses eca-qos-actions;
      uses eca-fwd-actions;
    }  // end of rule
  }  // end of group list
}  // end of policy-groups
}  // end of policy set

//<CODE ENDS>
6. IANA Considerations

This draft includes no request to IANA.

7. Security Considerations

These generic filters are used in the I2RS FB-RIBs to filter packets in a traffic stream, act to modify packets, and forward data packets. These I2RS filters operate dynamically at same level as currently deployed configured filter-based RIBs to filter, change, and forward traffic. The dynamic nature of this protocol requires that I2RS Filters track the installer of group information and rules.

This section will be augmented after a discussion with security experts.

8. Informative References

[I-D.hares-i2rs-usecase-reqs-summary]

[I-D.ietf-i2rs-architecture]

[I-D.ietf-i2rs-rib-info-model]

[I-D.ietf-netconf-restconf]

[I-D.ietf-netmod-acl-model]


Authors’ Addresses

Susan Hares
Huawei
7453 Hickory Hill
Saline, MI  48176
USA

Email: shares@ndzh.com

Qin Wu
Huawei
101 Software Avenue, Yuhua District
Nanjing, Jiangsu  210012
China

Email: bill.wu@huawei.com
Abstract

The rapid growth in the variety and importance of traffic flowing over increasingly complex enterprise and service provider network architectures makes the task of network operations and management applications and deploying new services much more difficult. Simplified Use of Policy Abstractions (SUPA) defines an interface to a network management function that takes high-level, possibly network-wide policies as input and creates element configuration snippets as output. SUPA expresses policies using a generic policy information model, and outputs generic YANG data models.

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

Copyright Notice

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

The rapid growth in the variety and importance of traffic flowing over increasingly complex enterprise and service provider network architectures makes the task of network operations and management applications and deploying new services much more difficult. In addition, network operators want to deploy new services quickly and efficiently. Two possible mechanisms for dealing with this growing difficulty are the use of software abstractions to simplify the design and configuration of monitoring and control operations and the use of programmatic control over the configuration and operation of such networks. Policy-based management can be used to combine these two mechanisms into an extensible framework.

Policy statements can be used to express high-level network operator requirements directly, or from a set of management applications, to a network management or element system. The network management or element system can then interpret those requirements to control the configuration of network elements.

The key benefit of policy management is that it enables different network elements and services to be instructed to behave the same way, even if they are programmed differently.

Simplified Use of Policy Abstractions (SUPA) will define a generic policy information model (GPIM) for use in network operations and management applications. The GPIM represents different types of policies for controlling the configuration of network elements throughout the service development and deployment lifecycle. The GPIM will be translated into corresponding YANG data models to define interoperable implementations that can exchange and modify generic policies using protocols such as NETCONF/RESTCONF.

Management applications will benefit from using policy rules that enable scalable and consistent programmatic control over the configuration of network elements.
1.1. Problem Statement

Network operators are faced with networks of increasing size and complexity while trying to improve their quality and availability, as more and more business services depend on them.

Currently, different technologies and network elements require different forms of the same policy that governs the production of network configuration snippets. The power of policy management is its applicability to many different types of systems. This provides significant improvements in configuration agility, error detection, and uptime for operators.

Many different types of actors can be identified that can use a policy management system, including applications, end-users, developers, network administrators, and operators. Each of these actors typically has different skills and uses different concepts and terminologies. For example, an operator may want to express that only Platinum and Gold users can use streaming and interactive multimedia applications. As a second example, an operator may want to define a more concrete policy rule that looks at the number of dropped packets. If, for example, this number exceeds a certain threshold value, then the applied queuing, dropping and scheduling algorithms could be changed in order to reduce the number of dropped packets.

1.2. Proposed Solution

SUPA enables network operators to express policies to control network configuration data models. SUPA provides a generic infrastructure that defines policies to control the configuration of network elements. The configuration process is independent of domain or type of application, and results in configuration according to YANG data models.

Both of the above examples can be referred to as "policy rules", but they take very different forms, since they are at different levels of abstraction and likely authored by different actors. The first example described a very abstract policy rule, and did not contain any technology-specific terms, while the second example included a more concrete policy rule and likely used technical terms of a general (e.g., IP address range and port numbers) as well as vendor-specific nature (e.g., specific algorithms implemented in a particular device). Furthermore, these two policy rules could affect each other. For example, Gold and Platinum users might need different device configurations to give the proper QoS markings to their streaming multimedia traffic. This is very difficult to do if a common policy framework does not exist.
Note that SUPA is not limited to any one type of technology. While the above two policies could be considered "QoS" policies, other examples include:

- network elements must not accept passwords for logins
- all SNMP agents in this network must drop all SNMP traffic unless it is originating from, or targeting, the management network
- Periodically perform workload consolidation if average CPU utilization falls below X%

The above three examples are not QoS related, and will be explained more in Sections 4.1 and 4.2. This emphasizes the utility of the SUPA approach in being able to provide policies to control different types of network element configuration snippets.

There are many types of policies. SUPA differentiates between "management policies" and "embedded policies". Management policies are used to control the configuration of network elements. Management policies can be interpreted externally to network elements, and the interpretation typically results in configuration changes of collections of network elements. In contrast, "embedded policies" are policies that are embedded in the configuration of network elements, and are usually interpreted on network elements in isolation. Since embedded policies are interpreted in the network device, they are typically composed in a very specific fashion to run at near-realtime timescales.

1.3. Value of the SUPA Approach

SUPA will achieve an optimization and reduction in the amount of work required to define and implement policy-based data models in the IETF. Part of this is due to the generic and extensible framework of SUPA, which models concepts common to any type of policy as well as provides two information models (ECA and declarative), along with the associated YANG data models.

SUPA defines policy independent of where it is located. Other WGs are working on embedding policy in the configuration of a network element; SUPA is working on defining policies that can be interpreted external to network elements. Hence, SUPA policies can be used to define the behavior of and interaction between embedded policies.
SUPA can also be used to derive a (more abstract) information model from a (more specific) data model. This extracts data that is part of a particular technology and/or application and makes it reusable, so that these data can be applied to multiple technologies and/or domains.

The SUPA policy framework defines a set of consistent, flexible, and scalable mechanisms for monitoring and controlling resources and services. It may be used to create a management and operations interface that can enable existing IETF data models, such as those from I2RS and L3SM, to be managed in a unified way that is independent of application domain, technology and vendor. Resource and service management become more effective, because policy defines the context that different operations, such as configuration, are applied to.

2. Framework for Generic Policy-based Management

This section briefly describes the design and operation of the SUPA policy-based management framework.

2.1. Overview

Figure 1 shows a simplified functional architecture of how SUPA is used to define policies for creating network element configuration snippets. SUPA uses the Generic Policy Information Model (GPIM) to define a consensual vocabulary that different actors can use to interact with network elements. The GPIM defines a generic structure for imperative and declarative policies. This is converted to generic YANG data models. The IETF produces the models, and IANA is used to register the model and changes.

In the preferred approach, SUPA generic policy data models are then used to create vendor- and technology-specific data models. These define the specific elements that will be controlled by policies. The Policy Interface uses this information to create appropriate input mechanisms for the operator to define policies (e.g., a web form or a script) for creating and managing the network configuration. The operator interacts with the interface, which is then translated to configuration snippets. Note that the policy interface is NOT being designed in SUPA.

In one of possibly several alternate approaches (shown with asterisks in Figure 1), the SUPA generic policy YANG data models contain enough information for the Policy Interface to create appropriate input mechanisms for the operator to define policies. This transfers the work of building vendor- and technology-specific data models to the SUPA Data Model-Specific Translation Function.
Figure 1 is meant to be exemplary. The Operator actor shown in Figure 1 can interact with SUPA in other ways not shown in the Figure. In addition, other actors that can interact with SUPA were not shown for simplicity. For example, an application developer could build an application that uses the SUPA information and data models to directly output configuration snippets. In addition, other actors can use the SUPA framework.
SUPA defines an Event-Condition-Action (ECA) policy as an example of imperative policies; it also defines two forms of declarative policies using simple Propositional Logic and First Order Logic. An ECA policy rule is activated when its event clause is true; the condition clause is then evaluated and, if true, signals the execution of one or more actions in the action clause.

In contrast, a declarative policy defines what actions to take, but not how to execute them. Declarative policies in SUPA take the form of a set of statements that present facts, and a conclusion of those facts.

2.2. Operation

SUPA can be used to define various types of policies, including policies that affect services and/or the configuration of individual or groups of network elements. SUPA can be used by a centralized and/or distributed set of entities that for creating, managing, interacting with, and retiring policy rules. The Policy Interface and SUPA Translation Function are two entities that make up the Policy Management (PM) function.

The duties of the PM function depend on the type and nature of policies being used. For example, imperative (e.g., ECA) policies require conflict detection and resolution, while declarative policies do not. A short exemplary list of functions that are common to both types of policies include:

- policy creation, update, delete, and view functions (typically in conjunction with policy repositories)
- policy storage, search, and retrieval (typically uses distributed repositories that the PM communicates with)
- policy distribution (typically uses a message bus; note that this involves requesting and responding to requests for policy decisions as well as distributing policies and informing interested entities of policy results)
- making policy decisions (this SHOULD include more than the simple Policy Decision Point functions defined in [RFC3198])
- executing policy decisions (this SHOULD include more than the simple Policy Enforcement Point functions defined in [RFC3198])
- validating that the execution of the policy produced what was expected (this is NOT defined in [RFC3198]).

An exemplary architecture that illustrates these concepts is shown in [TR235].
The SUPA scope is limited to policy information and data models. SUPA will not define network resource data models, which is out of scope. Similarly, SUPA will not define network service data models, which is also out of scope. Instead, SUPA will make use of network resource data models defined by other WGs or SDOs.

2.3. Generic Policy Information Model

The GPIM provides a common vocabulary for representing concepts that are common to expressing different types of policy, but which are independent of language, protocol, repository, and level of abstraction.

This enables different policies at different levels of abstraction to form a continuum, where more abstract policies can be translated into more concrete policies, and vice-versa. For example, the information model can be extended by generalizing concepts from an existing data model into the GPIM; the GPIM extensions can then be used by other data models.

SUPA will develop an information model for expressing policy at different levels of abstraction. Specifically, three information model fragments are envisioned: (i) a generic policy information model (GPIM) that defines concepts needed by policy management independent of the form and content of the policy, (ii) a more specific information model that refines the GPIM to specify how to build policy rules of the event-condition-action paradigm, and (iii) a more specific information model that refines the GPIM to specify how to build policy rules that declaratively specify what goals to achieve (but not how to achieve those goals); this is often called "intent-based" policy. These are all contained in the Generic Policy Information Model block in Figure 1.

2.4. Refinement of the GPIM

An information model is abstract. As such, it cannot be directly instantiated (i.e., objects cannot be created directly from it). Therefore, SUPA translates its information model to two different data models (which can be instantiated).

SUPA will translate the GPIM into concrete YANG data models that define how to manage and communicate policies between systems. Any number of imperative and/or declarative policy YANG data models may be instantiated from the GPIM, and may be used separately or in combination. This is enabled by the SUPA GPIM.
The two data models differ in how they represent policies. However, they share common characteristics and behavior. Therefore, it is easier to define a set of three information models to represent the common, ECA, and declarative parts of a policy. These three information models are then translated into either a YANG ECA data model or a YANG declarative data model. Note that because they share a common information model, they can be used separately or together (e.g., a declarative policy could call an ECA policy). This provides two different types of abstractions that serve different use cases. It also helps prove the genericity of the GPIM.

2.4.1. Event-Condition-Action Policy Information Model

The SUPA ECA Policy Rule Information Model (EPRIM) represents a policy rule as a statement that consists of an event clause, a condition clause, and an action clause. An ECA policy rule is activated when its event clause is true; the condition clause is then evaluated and, if true, signals the execution of one or more actions in the action clause. This type of Policy Rule explicitly defines the current and desired states of the system being managed.

2.4.2. Declarative Policy Information Model

The SUPA Logic Statement Information Model (LSIM) is a set of (logic-based) propositions that form a (single) conclusion. A proposition is a type of statement that is either TRUE or FALSE. A proposition can be created from simpler propositions. This version of the LSIM defines two forms of SUPA Logic Statements: one using propositional logic, and one using first order logic.

3. Application of Generic Policy-based Management

This section provides examples of how SUPA can be used to define different types of policies. Examples applied to various domains, including system management, operations management, access control, routing, and service function chaining, are also included.

3.1. Declarative Examples

Declarative policies are policies that describe what to do, but not how to do it. Declarative policies can apply to services and/or resources. Here are some simple examples:
System and Operations Management Examples

All routers and switches must have password login disabled.

The above policy first resolves ‘routers and switches’ to a set of network elements, and then pushes the appropriate configuration to those network elements.

All SNMP agents must enable SNMPv3 and must disable all other versions of SNMP.

The above policy can be mapped to the leafs v1, v2c, and v3 in the ietf-snmp YANG data model (RFC 7407).

All SNMP traffic is dropped unless it originates from, or is directed to, an interface of a management system.

The above policy first resolves a management system interface to a list of IP addresses, and then creates a set of suitable ACL rules that are configured on all network elements.

Access to source code servers is limited to authorized Intranet users.

The above policy assumes that the user is authenticated and authorized to access the code server. It places an additional constraint of requiring Intranet access before granting access to the resource. Note that this rule is not limited to any one specific user or type of application.

Periodically perform workload consolidation if average CPU utilization falls below X%.

This policy moves workloads on a set of source VMs to a common target VM if the average CPU utilization for the CPUs on the source VM is less than a predefined threshold. Note that the policy did not specify which particular VM to move the workload on the source VM to; that is part of the search and optimization algorithms that are implied, but not specified, by this policy.

Service Management Examples

Proactively monitor Gold Service users to ensure their SLAs are not violated.
Gold Service is an aggregation of different traffic types, each with different constraints. The policy will dynamically create a service function chain based on the current context to ensure that the customer’s SLA is not violated.

Gold and Platinum Service Users must have WAN optimization applied to multimedia applications.

The above policy applies only to multimedia applications for users whose SLA types are either Gold or Platinum. It installs a service chain that performs WAN optimization (and likely content caching and other services) to ensure that the SLAs of these users are not violated.

3.2. ECA Examples

ECA policies are statements that consist of an event clause, a condition clause, and an action clause.

Network Service Management Example

Event: too many interface alarms received from an L3VPN service
Condition: alarms resolve to the same interface within a specified time period
Action: if error rate exceeds x% then put L3VPN service to Error State and migrate users to one or more new L3VPNs

Security Management Example

Event: anomalous traffic detected in network
Condition: determine the severity of the traffic
Action: apply one or more actions to affected NEs based on the type of the traffic detected (along with other factors, such as the type of resource being attacked if the traffic is determined to be an attack)

Traffic Management Examples

Event: edge link close to being overloaded by incoming traffic
Condition: if link utilization exceeds Y% or if link utilization average is increasing over a specified time period
Action: change routing configuration to other peers that have better metrics
Event: edge link close to be overloaded by outgoing traffic
Condition: if link utilization exceeds Z% or if link utilization average is increasing over a specified time period
Action: reconfigure affected nodes to use source-based routing to balance traffic across multiple links

Service Management Examples

Event: alarm received or periodic time period check
Condition: CPU utilization level comparison
Action: no violation: no action
violation:
  1) determine workload profile in time interval
  2) determine complementary workloads (e.g., whose peaks are at different times in day)
  3) combine workloads (e.g., using integer programming)

Event: alarm received or periodic time check
Condition: if DSCP == AFxy and throughput < T% or packet loss > P%
Action: no: no action
yes: remark to AFx’y’; reconfigure queuing; configure shaping to S pps; ...

Note: it is possible to construct an ECA policy rule that is directly tied to configuration parameters; this is in general not possible for declarative policy. The value of declarative policy is in expression of the goal of the policy, and the freedom in implementing that goal. The value of ECA is in more clearly specifying what needs to be done.

3.3. ECA plus Declarative Example

The fundamental reason that SUPA defines two different types of policy rules is to enable different actors to express policy in a manner conducive to their roles. The SGPIM defines concepts that are common to both the EPRIM and the SLSIM. This enables these two types of policies to be used together to provide a more powerful definition of the goals of the policy as well as how to implement those goals.

For example, compare the ECA and declarative forms of the SLA Service Management Policy:
Declarative form:
Proactively monitor Gold Service users to ensure their SLAs are not violated.

ECA form:
Event: alarm received or periodic time check
Condition: if DSCP == AFxy and
throughput < T% or packet loss > P%
Action: no: no action
yes: remark to AFx’y’; reconfigure queuing;
configure shaping to S pps; ...

The declarative policy is more abstract than its ECA counterpart, since the declarative version expresses intent without defining which specific network elements are affected and how the configuration of those network elements should be changed. The above ECA policy rule is written in a high-level form, but note that it still is specifying how to monitor the Gold Service, how to determine if the SLA is being violated, and which actions to take.

The execution of the declarative example could result in one or more ECA policy rules being triggered, such as the one above. Similarly, an ECA policy rule could trigger additional ECA policy rules to be evaluated. For example, the above ECA rule could be rewritten so that if the condition was satisfied, then each of the actions shown could be their own policy rules. This provides additional flexibility through reusing policy rules and the components of policy rules.

4. Related Work

4.1. Related Work within the IETF

4.1.1. I2RS Working Group

I2RS defines an interface that interacts with the routing system using a collection of protocol-based control or management interfaces. Users of I2RS interfaces are typically management applications and controllers. SUPA does not directly interface to the routing system. Rather, SUPA uses data produced by I2RS (e.g., topological information) to construct its policies.
4.1.2. L3SM Working Group

L3SM defines an L3 VPN service model that can be used for communication between customers and network operators. This model enables an orchestration application or customers to request network services provided by L3 VPN technologies. The implementation of network services is often guided by specific policies, and SUPA provides a tool that can help with the mapping of L3 VPN service requests to L3 VPN configurations of network elements.

4.1.3. ALTO Working Group

The ALTO working group defined an architecture for exposing topology information, more specifically the cost of paths through an infrastructure, as defined in [RFC7285]. ALTO services are able to provide network maps defined as groups of endpoints, and can therefore represent any granularity of network, from the physical to groups of networks following similar paths or restraints. Although this model can represent different levels of granularities, it is not clear if it could be adapted easily for other purposes than providing cost maps in the context of ALTO. The ALTO model is meant to be used outside of the trust domain of an ISP by external clients.

SUPA does not generate data that is similar to ALTO. Rather, SUPA could use ALTO data as part of its policies to configure services and/or resources.

4.1.4. TEAS Working Group

The Traffic Engineering Architecture and Signaling (TEAS) working group is responsible for defining MPLS- and GMPLS-based Traffic Engineering architectures that enable operators to control how specific traffic flows are treated within their networks. It covers YANG models for a traffic engineering database. In coordination with other working groups (I2RS) providing YANG models for network topologies.

Both TEAS and SUPA use YANG data models. SUPA does not generate traffic engineering (TE) data. However, SUPA could use TE data as part of its policies for configuring resources and/or services. SUPA could also define policies that define which service, path, and link properties to use for a given customer, and consequently, which protocol extensions to use. TEAS data could also be used to enable operators to define how particular traffic flows are treated in a more abstract (but still consistent) manner.
4.1.5. BESS Working Group

The BGP Enabled Services (BESS) working group defines and extends network services that are based on BGP. This includes BGP/MPLS IP provider-provisioned L3VPNs, L2VPNs, BGP-enabled VPN solutions for use in data center networking, and extensions to BGP-enabled solutions to construct virtual topologies in support of services such as Service Function Chaining. The working group is also chartered to work on BGP extensions to YANG models and data models for BGP-enabled services.

Both BESS and SUPA use YANG data models. SUPA could generate BGP configurations by using data defined by BESS as part of its policies for configuring resources and/or services.

SUPA could also define policies that govern different aspects of services defined by BESS.

4.1.6. SFC Working Group

The Service Function Chaining (SFC) working group defines a mechanism where traffic is classified; that classification is then used to select an ordered set of services to pass the traffic through.

Both SFC and SUPA use YANG data models. SUPA could define policies that augment the functionality of SFC in several different ways, including: (1) path selection based on context, (2) which set of mechanisms to use to steer traffic through which set of service functions, (3) simplify the definition of dynamic service function chains (e.g., service paths that change based upon a set of data that is discovered at runtime), and (4) scalable mechanisms to monitor and control the configuration of SFC components.

4.1.7. NVO3 Working Group

The NVO3 group proposes a way to virtualize the network edge for data centers in order to be able to move virtual instances without impacting their network configuration. This is realized through a centrally controlled overlay layer-3 network. The NVO3 work is not about defining policy information; rather, it uses policy information to perform some functions. Both NVO3 and SUPA use YANG data models. SUPA could define policies that define how the logically centralized network virtualization management entity (or entities) of NVO3 behave (e.g., the functions in the network virtualization control plane).
4.1.8. ACTN BoF (IETF-90)

The ACTN proposed work, as described in [actn] framework, has two main goals, the abstraction of multiple optical transport domains into a single controller offering a common abstract topology, and the splitting of that topology into abstract client views that are usually a fraction of the complete network. The ACTN work is therefore about unification of several physical controllers into a virtual one, and also about the segmentation, isolation and sharing of network resources. The ACTN work is not about defining policy information. Both ACTN and SUPA use YANG data models. SUPA could define policies that define the behavior of the controller.

4.1.9. Previous IETF Policy Models

SUPA is technology-neutral, previous RFCs weren’t. SUPA defines a common structure from which both ECA and declarative policies can be defined and combined; this was not possible in previous RFCs. Previous RFCs do NOT define metadata, and do NOT enable policies to formally define obligation, permission, and related concepts. Finally, SUPA uses software patterns, which previous RFCs didn’t.

4.2. Related Work outside the IETF

4.2.1. TM Forum

The TM Forum (a.k.a., the TeleManagement Forum) develops standards and best practices, research, and collaborative programs focused on digital business transformation. It consists of three major programs:

1) Agile Business and IT
2) Customer Centricity (experience)
3) Open Digital Ecosystem

Of these, the ZOOM (Zero-touch Orchestration, Operations, and Management) project, located in the Agile Business and IT project, is the main sub-project in this area that is of interest to SUPA.

Within ZOOM, the Foundational Studies project contains work on an information model and management architecture that are directly relevant to SUPA. The TMF Information Model, Policy, and Security working groups are involved in this work.
The ZOOM information model updates the existing Shared Information and Data (SID) information model to add support for the management of physical and virtual infrastructure, event- and data-driven systems, policy management (architecture and model), metadata for describing and prescribing behavior that can support changes at runtime, and access control. The policy information model defines imperative (ECA), declarative (intent-based), utility function, and promise policies. The work in [ID.draft-strassner-supageneric-policy-info-model] is based on this work. It currently extends the ZOOM ECA model and provides additional detail not currently present in ZOOM; the next version of this draft will do the same for declarative policies.

There is currently no plan to use the utility function and promise policies of ZOOM in SUPA. Finally, it should be noted that the data model work planned for SUPA is not currently planned for the ZOOM project.

4.2.2. MEF

The MEF (originally named the Metro Ethernet Forum) develops architecture, service and management specifications related to Carrier Ethernet (CE). The CE architecture includes the definition of several interfaces specific to CE like the User Network Interface (UNI) and External Network Network Interface (ENNI). Specifications developed in this space include the definitions of CE services, CE service attributes, Ethernet Access Services, Class of Service, OAM and Management interfaces, Service Activation and Test. The more recent vision of the MEF is described as The Third Network, and includes plans to develop Lifecycle Service Orchestration with APIs for existing network, NFV, and SDN implementations enabling Agile, Assured, and Orchestrated Services. This stage of the MEF activity is now in early phases with focus on architectural work.

The MEF has developed a number of Information and Data Models, and has recently started a project that used YANG to model and manage the services defined by the MEF. While the MEF has created rigorous definitions of these services, they are specific to transport technology, and they do not include and rely on policies.

4.2.3. Open Daylight

Open Daylight network controller implements a number of models through its service abstraction Layer (MD-SAL) based on draft IETF Yang models. Open Daylight is an open source project. Two of these are relevant to SUPA, and are described below.
4.2.3.1. Network Intent Composition (NIC)

The Network Intent Composition project aims at providing better flexibility by using declarative policies. It does not cover other types of policies, such as ECA policy rules. The intent-based interface aims to provide a high level of abstraction, primarily for use by an application developer. Its progress has recently stalled.

4.2.3.2. Group Based Policy

The Group Based Policy project defines an application-centric policy model for Open Daylight that separates information about application connectivity requirements from information about the underlying details of the network infrastructure. The model is positioned as declarative, but uses a relational approach to specifying policy.

4.2.4. Open Networking Foundation

The ONF created a group responsible of defining northbound interfaces, but this hasn’t lead to the publication of standards in this area so far. A blog entry on the ONF web site showed an interest in using the principle of intents at ONF, but no details were provided on the status of this project. A members-only whitepaper was recently published.

4.2.5. OpenStack

OpenStack software controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure. Few of the below mentioned OpenStack projects provides policy abstraction and better flexibility to the user.

4.2.5.1. Group-Based Policy

The Group-Based Policy project for OpenStack Neutron is built around entities assembled in Endpoints Groups (EPG) that provide or consume Contracts. Such Contracts are hierarchical entities containing policy rules. A first version was released in January 2015, based on the Juno release. This type of approach is more relational than declarative, but could be used to describe a large amount of possible scenarios. It has the advantage of providing a relatively simple policy model that covers a large applicability. From an OpenStack point of view, the scope of Group-Based Policies is limited to networking within the Neutron module.
4.2.5.2. Congress

The Congress project within OpenStack provides a way to define complex policies using extensions to the Datalog language. Datalog is entirely declarative, and its evaluation is based on first-order logic with restrictions. This gives it interesting properties, such as providing the same result no matter the order in which the statements are made. The language allows for the definition of types and for active enforcement or verification of the policies.

There is a significant body of knowledge and experience relating to declarative languages and their implementation. Congress policies aim at manipulating objects exposed by multiple OpenStack modules, and is therefore larger in scope than network element policies.

The declarative policies of SUPA are similar to those in Congress; the primary difference relies in the characteristics and behavior (in the sense of restrictions) of the underlying logic for Congress vs. SUPA. SUPA’s propositional logic statements are simpler but more limited than Congress, while SUPA’s first-order logic statements are more complex but more powerful than those of Congress. If desired, a Congress model could be easily added to SUPA.

4.2.6. The NEMO Project (not a BoF yet)

The NEMO project is a research activity aimed at defining a simple framework for "intent-based" networking. This project concentrates on creating a domain-specific language and associated API, not a model or even a rigorous definition of what a policy rule is.

The NEMO syntax defines a very simple information model that has three basic elements for network manipulation: nodes, links, and flows. A policy rule is NOT defined in this model. Rather, policy is defined as a command. The NEMO project has been successfully demonstrated at IETF-91, along with a companion graphical user interface.

NEMO declarative policies are different than SUPA declarative policies. NEMO uses a flatter, simpler object model with fewer objects to represent targets of policy. NEMO does not define a policy model, and does not support ECA policies. NEMO uses a condition-action paradigm to execute its declarative policies. In contrast, SUPA uses a richer class model to represent ECA and declarative policies. SUPA declarative policies are executed using formal logic. SUPA has not proposed a language.
4.2.7. The Floodlight Project

The Floodlight is an OpenFlow-enabled SDN controller. It uses another open source project called Indigo to support OpenFlow and manage southbound devices. The Indigo agent also supports an abstraction layer to make it easy to integrate with physical and virtual switches. It supports configuration of an abstraction layer so that it can configure OpenFlow in hybrid mode.

4.2.8. The ONOS Project

The ONOS is an SDN controller design for Service Provider networks. It uses a distributed architecture, and supports abstraction for both southbound and northbound interfaces. Its modules are managed as OSGi bundles. It is an open source project.

ONOS announced an "application-intent framework", which is similar in nature to SUPA's declarative policies. However, no object model or language has been defined yet.

5. Conclusions: the Value of SUPA

SUPA defines an interface to a network management function that takes high-level, possibly network-wide policies as input and creates element configuration snippets as output. SUPA expresses policies using a generic policy information model, and produces generic policy YANG data models. SUPA focuses on management policies that control the configuration of network elements. Management policies can be interpreted outside of network elements, and the interpretation typically results in configuration changes of collections of network elements.

Policies embedded in the configuration of network elements are not in the scope of SUPA. In contrast to policies targeted by SUPA, embedded policies are usually interpreted on network elements in isolation, and often at timescales that require the representation of embedded policies to be optimized for a specific purpose.

The SUPA information model generalizes common concepts from multiple technology-specific data models, and makes it reusable. Conceptually, SUPA can be used to interface and manage existing and future data models produced by other IETF working groups. In addition, by defining an object-oriented information model with metadata, the characteristics and behavior of data models can be better defined.
6. Security Considerations

TBD.

7. IANA Considerations

This document has no actions for IANA.

8. Contributors

The following people all contributed to creating this document:

Jun Bi, Tsinghua University
Vikram Choudhary, Huawei Technologies
Luis M. Contreras, Telefonica I+D
Georgios Karagiannis, Huawei Technologies
Hosnieh Rafiee, Huawei Technologies Duesseldorf GmbH
Dan Romascanu, Avaya
Jon Saperia, JDS Consulting
J. Schoenwaelder, Jacobs University, Germany
Qiong Sun, China Telecom
Parviz Yegani, Juniper Networks
Cathy Zhou, Huawei Technologies

9. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: J. Bi, Luis M. Contreras, G. Karagiannis, D. Romascanu, J. Saperia, J. Schoenwaelder, Q. Sun, P. Yegani, and C. Zhou.

10. References

10.1. Informative References


[SUPA-DDC] Y. Cheng, and JF. Tremblay, "Use Cases for Distributed Data Center Applications in SUPA", IETF Internet draft, draft-cheng-supa-ddc-use-cases, January 2015


Authors’ Addresses

Maxim Klyus, Ed.
NetCracker
Kozhevnicheskaya str., 7 Bldg. #1
Moscow, Russia
Phone: +7-916-8575717
E-mail: klyus@netcracker.com

John Strassner, Ed.
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95138 USA
Email: john.sc.strassner@huawei.com
Simplified Use of Policy Abstractions (SUPA) defines an interface
to a network management function that takes high-level, possibly
network-wide policies as input and creates element configuration
snippets as output. SUPA addresses the needs of operators and
application developers to represent multiple types of policy
rules, which vary in the level of abstraction, to suit the needs
of different actors. This document defines a single common
extensible framework for representing different types of policy
rules, in the form of a set of information model fragments, that
is independent of language, protocol, repository, and the level
of abstraction of the content of the policy rule. This enables a
common set of concepts defined in this set of information models
to be mapped into different data models that use different
languages, protocols, and repositories to optimize their usage.
The definition of common policy concepts also provides better
interoperability by ensuring that each data model can share a set
of common concepts, independent of its level of detail or the
language, protocol, and/or repository that it is using.

Specifically, this document defines three information models:

1. A framework for defining the concept of policy,
independent of how policy is represented or used; this is
called the SUPA Generic Policy Information Model (GPIM)
2. A framework for defining a policy model that uses the
event-condition-action paradigm; this is called the SUPA
   Eca Policy Rule Information Model (EPRIM)
3. A framework for defining a policy model that uses a
declarative (e.g., intent-based) paradigm; this is called
   the SUPA Logic Statement Information Model (LSIM)

The combination of the GPIM and the EPRIM, or the GPIM and the
LSIM, provide an extensible framework for defining policy that
uses an event-condition-action or declarative representation that
is independent of data repository, data definition language, query
language, implementation language, and protocol. A specific design
goal of SUPA is to enable ECA policy rules and declarative policy
statements to be used together or separately. This is achieved by
deriving both the EPRIM and the LSIM from the GPIM.
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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on October 26, 2015.

Copyright Notice

Copyright (c) 2014 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 ................................................. 8
2. Conventions Used in This Document ............................ 8
3. Terminology .................................................. 8
   3.1. Acronyms................................................. 8
   3.2. Definitions ............................................. 9
3.2.1. Core Terminology ........................................ 9
  3.2.1.1. Information Model ................................ 9
  3.2.1.2. Data Model ......................................... 10
  3.2.1.3. Container .......................................... 10
  3.2.1.4. PolicyContainer ................................. 10

3.2.2. Policy Terminology ...................................... 10
  3.2.2.1. SUPAPolicy ......................................... 11
  3.2.2.2. SUPAPolicyStatement ............................ 11
  3.2.2.3. SUPAECAPolicyRule ............................. 11
  3.2.2.4. SUPALogicStatement ............................. 11
  3.2.2.5. SUPAMetadata .................................. 12
  3.2.2.6. SUPAPolicyTarget ............................... 12
  3.2.2.7. SUPAPolicySubject .............................. 12

3.2.3. Modeling Terminology ................................. 13
  3.2.3.1. Inheritance ..................................... 13
  3.2.3.2. Relationship .................................... 13
  3.2.3.3. Association ..................................... 13
  3.2.3.4. Aggregation ..................................... 13
  3.2.3.5. Composition ..................................... 14
  3.2.3.6. Association Class .............................. 14
  3.2.3.7. Multiplicity .................................... 14
  3.2.3.8. Navigability ................................... 14
  3.2.3.9. Abstract Class .................................. 15
  3.2.3.10. Concrete Class ................................ 15

3.2.4. Mathematical Logic Terminology ........................ 15
  3.2.4.1. Predicate ....................................... 15
  3.2.4.2. Logic Operators ................................. 15
    3.2.4.2.1. Propositional Logic Connectives .......... 15
    3.2.4.2.2. First Order Logic Quantifiers ........... 16
  3.2.4.3. Propositional Logic ................................ 16
  3.2.4.4. First-Order Logic ................................ 16

3.3. Symbology ................................................. 17
  3.3.1. Inheritance ........................................ 17
  3.3.2. Association ........................................ 17
  3.3.3. Aggregation ........................................ 17
  3.3.4. Composition ....................................... 18
  3.3.5. Association Class .................................. 18
  3.3.6. Logical Connectives ................................ 18
  3.3.7. Quantifiers ........................................ 18

4. Policy Abstraction Architecture ........................... 19
  4.1. Motivation ............................................ 20
  4.2. SUPA Approach ......................................... 21
  4.3. SUPA Generic Policy Information Model Overview ..... 21
  4.4. Structure of SUPA Policies .......................... 24
    4.4.1. ECA Policy Rule Structure ....................... 24
    4.4.2. Logical Statement Structure ..................... 25
  4.5. GPIM Assumptions ...................................... 27
  4.6. Scope of Previous Work ............................... 28
5. GPIM Model .................................................................................. 29
  5.1. Overview .................................................................................. 29
  5.2. The Abstract Class "SUPAPolicy" ............................................. 29
    5.2.1. SUPAPolicy Attributes .................................................. 32
      5.2.1.1. The Attribute "supaObjectIDContent" ................... 32
      5.2.1.2. The Attribute "supaObjectIDFormat" ................. 32
      5.2.1.3. The Attribute "supaPolicyDescription" ............ 32
      5.2.1.4. The Attribute "supaPolicyName" ...................... 32
  5.2.2. SUPAPolicy Relationships .................................................. 33
    5.2.2.1. The Relationship "SUPAPolicyMetadata" ................. 33
    5.2.2.2. The Association Class
      "SUPAPolicyMetadataDetail" ........................................... 33
  5.3. The Abstract Class "SUPAPolicyAtomic" .................................. 33
    5.3.1. SUPAPolicyAtomic Attributes ..................................... 34
      5.3.1.1. The Attribute "supaPolicyDeployStatus" ............ 34
      5.3.1.2. The Attribute "supaPolicyExecStatus" .............. 34
  5.3.2. SUPAPolicyAtomic Relationships ..................................... 34
    5.3.2.1. The Aggregation "SUPAPAHasPolicyStmts" ............ 34
    5.3.2.2. The Association Class "SUPAPAHasPolicyStmtDetail" .... 35
  5.4. The Concrete Class "SUPAPolicyComposite" ............................. 35
    5.4.1. SUPAPolicyComposite Attributes ................................ 35
      5.4.1.1. The Attribute "supaPCFailureStrategy" ............ 36
      5.4.1.2. The Attribute "supaPCIsMatchAll" .................. 36
    5.4.2. SUPAPolicyComposite Relationships ........................... 36
      5.4.2.1. The Aggregation "HasSUPAPolicies" ................. 36
      5.4.2.2. The Association Class "HasSUPAPolicyDetail" ....... 37
  5.5. The Abstract Class "SUPAPolicyStatement" ................................ 37
    5.5.1. SUPAPolicyStatement Attributes ................................ 39
      5.5.1.1. The Attribute "supaPolStmtAdminStatus" .......... 39
      5.5.1.2. The Attribute "supaPolStmtExecStatus" .......... 40
      5.5.1.3. The Attribute "supaPolStmtIsCNF" ............... 40
    5.5.2. SUPAPolicyStatement Subclasses .................................. 40
      5.5.2.1. The Concrete Class "SUPAEncodedClause" ........... 41
        5.5.2.1.1. SUPAEncodedClause Attributes .................. 42
          5.5.2.1.1.1. The Attribute "supaClauseContent" ....... 42
          5.5.2.1.1.2. The Attribute "supaClauseFormat" ....... 42
          5.5.2.1.1.3. The Attribute "supaClauseResponse" .... 42
      5.5.3. SUPAPolicyStatement Relationships ........................ 43
        5.5.3.1. The Aggregation "SUPAPolicyTermsInStmt" ....... 43
        5.5.3.2. The Association Class "SUPAPolicyTermsInStmtDetail" .... 43
      5.5.3.2.1. SUPAPolicyStmtDetail Attributes ................ 43
    5.6. The Abstract Class "SUPAPolicySubject" ................................ 43
      5.6.1. SUPAPolicySubject Attributes ................................ 45
        5.6.1.1. The Attribute "supaPolicySubjectIsAuthenticated" .... 45
5.6.2. SUPAPolicySubject Relationships ....................... 45
   5.6.2.1. The Relationship "SUPAPolicySubjects" ............ 45
   5.6.2.2. The Association Class
            "SUPAPolicySubjectDetail" ................... 46
   5.6.2.2.1. SupaPolicySubjectDetail Attributes .......... 46
   5.6.2.2.1.1. The Attribute
            "supaPolicySubjectIsVerified" ........... 46
5.7. The Abstract Class "SUPAPolicyTarget" .................... 46
   5.7.1. SUPAPolicyTarget Attributes .......................... 47
   5.7.2. SUPAPolicyTarget Relationships ...................... 47
   5.7.2.1. The Relationship "SUPAPolicyTargets" ............ 47
   5.7.2.2. The Association Class "SUPAPolicyTargetDetail" 47
   5.7.2.2.1. SupaPolicyTargetDetail Attributes .......... 47
   5.7.2.2.1.1. The Attribute
            "supaPolicyTargetEnabled" ............ 47
5.8. The Abstract Class "SUPAPolicyTerm" ........................ 48
   5.8.1. SUPAPolicyTerm Attributes ........................... 49
   5.8.1.1 The Attribute "supaTermIsNegated" .................. 49
   5.8.2. SUPAPolicyTerm Relationships ....................... 49
   5.8.2.1. The Aggregation "SUPAPolicyTermsInStmt" .......... 49
   5.8.2.2. The Association Class
            "SUPAPolicyTermsInStmtDetail" .......... 49
   5.8.2.2.1. SUPAPolicyTermsInStmtDetail Attributes ...... 50
   5.8.2.2.1.1. The Attribute
            "supaPolTermOCLConstraint" ............ 50
5.8.3. SUPAPolicyTerm Subclasses ............................ 50
   5.8.3.1. The Concrete Class "SUPAPolicyVariable" .......... 51
   5.8.3.1.1. Problems with the RFC3460 Version of
              PolicyVariable ............................ 51
   5.8.3.1.2. SUPAPolicyVariable Attributes ............... 52
   5.8.3.1.2.1. The Attribute
            "supaAllowedValueTypes[0..n]" .......... 52
   5.8.3.1.2.2. The Attribute "supaPolVarContent" ........ 52
   5.8.3.1.2.3. The Attribute "supaPolVarType" .......... 52
   5.8.3.1.3. SUPAPolicyVariable Subclasses ................ 53
   5.8.3.2. The Concrete Class "SUPAPolicyOperator" .......... 53
   5.8.3.2.1. Problems with the RFC3460 Version .......... 53
   5.8.3.2.2. SUPAPolicyOperator Attributes ............... 53
   5.8.3.2.2.1. The Attribute "supaPolOpType" .......... 54
   5.8.3.3. The Concrete Class "SUPAPolicyValue" ........... 54
   5.8.3.3.1. Problems with the RFC3460 Version of
              PolicyValue ............................. 55
   5.8.3.3.2. SUPAPolicyValue Attributes .................. 55
   5.8.3.3.2.1. The Attribute "supaPolValContent" ....... 56
   5.8.3.3.2.2. The Attribute "supaPolValType" .......... 56
   5.8.3.3.3. SUPAPolicyValue Subclasses .................. 56
5.9. The Abstract Class "SUPAPolicyMetadata" .................. 56
   5.9.1. SUPAPolicyMetadata Attributes ..................... 56
   5.9.2. SUPAPolicyMetadata Relationships .................. 56
6. SUPA ECAPolicyRule Information Model ........................ 57
   6.1. Overview ............................................... 57
   6.2. Constructing a SUPAECAPolicyRule ....................... 58
   6.3. Working With SUPAECAPolicyRules ........................ 59
   6.4. The Concrete Class "SUPAECAPolicyRule" ............... 60
      6.4.1. SUPAECAPolicyRule Attributes ........................ 62
         6.4.1.1. The Attribute "supaECAPolicyIsMandatory" .... 62
         6.4.1.2. The Attribute "supaECAPolicyPriority" ....... 62
      6.4.2. SUPAECAPolicyRule Relationships ........................ 63
         6.4.2.1. The Aggregation "SUPAECAPolicyRules" ..... 63
            6.4.2.1.1. The Association Class "HasSUPAECAComponentDetail" .... 63
      6.4.3. SUPAECAPolicyRule Subclasses ........................ 63
         6.4.3.1. The Concrete Class "SUPAECAPolicyRuleAtomic" .... 64
            6.4.3.1.1. SUPAECAPolicyRuleAtomic Attributes .... 64
            6.4.3.1.2. SUPAECAPolicyRuleAtomic Relationships .... 64
         6.4.3.2. The Concrete Class "SUPAECAPolicyRuleComposite" .... 65
            6.4.3.2.1. SUPAECAPolicyRuleAtomic Attributes .... 65
               6.4.3.2.1.1. The Attribute "supaECAEvalStrategy" 65
               6.4.3.2.1.2. The Attribute "supaECAFailStrategy" 65
            6.4.3.2.2. SUPAECAPolicyRuleComposite Relationships .... 66
               6.4.3.2.2.1 The Aggregation "SUPAECAPolicyRules" 66
               6.4.3.2.2.2. The Association Class "SUPAECAPolicyRuleDetail" .... 66
      6.5. SUPAPolicyStatement Subclasses ........................ 66
         6.5.1. Designing SUPAPolicyStatements Using SUPABooleanClauses ................. 66
         6.5.2. The Abstract Class "SUPABooleanClause" .............. 67
            6.5.2.1. SUPABooleanClause Attributes ........................ 68
               6.5.2.1.1. The Attribute "supaBoolIsNegated" .... 69
               6.5.2.1.2. The Attribute "supaPolStmtBindValue" .... 69
            6.5.2.2. SUPABooleanClause Relationships ........................ 69
               6.5.2.2.1. The Relationship "SUPABooleanClauses" ... 69
                  6.5.2.2.1.1. The Association Class "SUPABooleanClauseDetail" .... 69
         6.5.3. SUPABooleanClause Subclasses ................................ 70
            6.5.3.1. The Abstract Class "SUPABooleanClauseAtomic" .... 70
            6.5.3.2. The Abstract Class "SUPABooleanClauseComposite" .... 70
            6.5.3.2.1. SUPABooleanClauseComposite Attributes ... 70
               6.5.3.2.1.1. The Attribute "supaBoolClauseIsCNF" 71
            6.6. The Abstract Class "SUPAECAComponent" ...................... 71
               6.6.1. SUPAECAComponent Attributes ........................ 71
               6.6.2. SUPAECAComponent Relationships ...................... 71
                  6.6.2.1. The Aggregation "SUPAECAComponentsInTerms" 71
                     6.6.2.1.1. The Association Class "SUPAECAComponentsInTermDetail" .... 72
6.6.3. SUPAECAComponent Subclasses ....................... 72
   6.6.3.1. The Concrete Class"SUPAEvent" ................ 72
   6.6.3.2. The Concrete Class"SUPACondition" ......... 72
   6.6.3.3. The Concrete Class"SUPAAction" ............ 72
7. SUPA Logic Statement Information Model ...................... 73
   7.1. Overview ........................................... 73
   7.2. Constructing a SUPAPLStatement .................... 73
   7.3. Working With SUPAPLStatements ...................... 73
   7.4. The Abstract Class "SUPAPLLogicClause" ............ 73
   7.5. The Abstract Class "SUPAPLStatement" ............ 74
      7.5.1. SUPAPLStatement Attributes ................... 74
      7.5.2. SUPAPLStatement Relationships ............... 74
      7.5.3. SUPAPLStatement Subclasses .................. 75
      7.5.3.1. The Concrete Class "SUPAPLArgument" ....... 75
      7.5.3.2. The Concrete Class "SUPAPLPremise" ...... 75
      7.5.3.3. The Concrete Class "SUPAPLConclusion" .... 75
   7.6. Constructing a SUPAFOLStatement ................... 75
   7.7. Working With SUPAFOLStatements .................... 75
      7.7.1. SUPAFOLStatement Attributes .................. 75
      7.7.2. SUPAFOLStatement Relationships ............... 75
      7.7.3. SUPAFOLStatement Subclasses .................. 75
      7.7.3.1. The Concrete Class "SUPAGoalHead" ........ 75
      7.7.3.2. The Concrete Class "SUPAGoalBody" ........ 75
   7.8. Combining Different Types of SUPAFOLStatements ....... 75
8. Examples ................................................................ 76
   8.1. SUPAECAPolicyRule Examples .......................... 76
   8.2. SUPAPLLogicStatement Examples ..................... 76
   8.3. Mixing SUPAECAPolicyRules and SUPAPLLogicStatements .. 76
9. Security Considerations ...................................... 76
10. IANA Considerations ....................................... 76
11. Acknowledgments .......................................... 76
12. References .............................................. 76
   12.1. Normative References .............................. 76
   12.2. Informative References ............................ 76
Authors’ Addresses ........................................... 7
1. Introduction

The Simplified Use Policy Abstractions (SUPA) addresses the needs of operators and application developers to represent multiple types of policy rules using a common structure. This enables policy rules to be defined independent of language, protocol, repository, and the level of abstraction of the content of the policy rule. This common framework currently takes the form of a set of three information model fragments. The SUPA Generic Policy Information Model (GPIM) defines a common set of policy management concepts that are independent of the type of policy rule, while the SUPA ECA Policy Rule Information Model (EPRIM) and SUPA Logic Statement Information Model (LSIM) define information models that are specific to the needs of Event-Condition-Action (ECA) policy rules and statements that are subsets of either Propositional Logic (PL) or First-Order Logic (FOL), respectively.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

3. Terminology

This section defines acronyms, terms, and symbology used in the rest of this document.

3.1. Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>CNF</td>
<td>Conjunctive Normal Form</td>
</tr>
<tr>
<td>DNF</td>
<td>Disjunctive Normal Form</td>
</tr>
<tr>
<td>ECA</td>
<td>Event-Condition-Action</td>
</tr>
<tr>
<td>EPRIM</td>
<td>(SUPA) ECA Policy Rule Information Model</td>
</tr>
<tr>
<td>FOL</td>
<td>First Order Logic</td>
</tr>
<tr>
<td>GPIM</td>
<td>(SUPA) Generic Policy Information Model</td>
</tr>
<tr>
<td>LSIM</td>
<td>(SUPA) Logic Statement Information Model</td>
</tr>
<tr>
<td>NETCONF</td>
<td>Network Configuration protocol</td>
</tr>
<tr>
<td>OAM&amp;P</td>
<td>Operations, Administration, Management, and Provisioning</td>
</tr>
<tr>
<td>OID</td>
<td>Object IDentifier</td>
</tr>
</tbody>
</table>
3.2. Definitions

This section defines the terminology that is used in this document.

3.2.1. Core Terminology

The following subsections define the terms "information model" and "data model".

3.2.1.1. Information Model

An information model is a representation of concepts of interest to an environment in a form that is independent of data repository, data definition language, query language, implementation language, and protocol.

Note: this definition is different than that of [RFC3198]. An information model is defined in [RFC3198] as: "An abstraction and representation of the entities in a managed environment, their properties, attributes, and operations, and the way that they relate to each other. It is independent of any specific repository, software usage, protocol, or platform." The SUPA definition is more specific, and corrects the following ambiguities:

- Most information models do not define operations; this is typically implementation-specific and a function of (at least) the language, protocol, and data repository used.
- It is unclear what the difference is between the terms "properties" and "attributes" (these are typically synonyms in modeling terminology)
- It is unclear what is meant by "software usage".
- It is unclear what is meant by "platform".
3.2.1.2. Data Model

A data model is a representation of concepts of interest to an environment in a form that is dependent on data repository, data definition language, query language, implementation language, and protocol (typically, but not necessarily, all three).

Note: this definition is different than that of [RFC3198]. A data model is defined in [RFC3198] as: "A mapping of the contents of an information model into a form that is specific to a particular type of data store or repository." The SUPA definition is more specific. For example, it takes into account differences between two implementations that use the same protocol, implementation language, and data repository, but which have different data definition and/or query protocols.

3.2.1.3. Container

A container is an object whose instances may contain zero or more additional objects, including container objects. A container provides storage, query, and retrieval of its contained objects in a well-known, organized way.

3.2.1.4. PolicyContainer

In this document, a PolicyContainer is a special type of container that provides at least the following three functions:

1. It uses metadata to define how its content is interpreted
2. It separates the content of the policy from the representation of the policy
3. It provides a convenient control point for OAMP operations

The combination of these three functions enables a PolicyContainer to define the behavior of how its constituent components will be accessed, queried, stored, retrieved, and how they operate.

3.2.2. Policy Terminology

The following terms define different policy concepts used in the SUPA Generic Policy Information Model (GPIM). Note that the prefix "SUPA" is used for all classes and relationships defined in the GPIM to ensure name uniqueness. Similarly, the prefix "supa" is defined for all SUPA class attributes.
3.2.2.1. SUPAPolicy

A SUPAPolicy is an abstract class that is a type of PolicyContainer.

SUPAPolicy is defined generically as a means to monitor and control the changing and/or maintaining of the state of one or more managed objects [1]. In this context, "manage" means that at least create, read, query, update, and delete functions are supported.

3.2.2.2. SUPAPolicyStatement

A SUPAPolicyStatement is an abstract class that contains an individual or group of related functions that are used to build different types of policies. This document defines two different types of policies: ECA policy rules and declarative policies. These different types of policies can be used to define a set of actions to take, or declaratively define a goal to be achieved and/or a set of facts to be used.

3.2.2.3. SUPAECAPolicyRule

An Event-Condition-Action (ECA) Policy (SUPAECAPolicyRule) is an abstract class that represents a policy rule as a three-tuple, consisting of an event, a condition, and an action clause. Each clause MUST be defined by at least one SUPAPolicyStatement. Optionally, the SUPAECAPolicyRule MAY contain one or more SUPAPolicySubjects, one or more SUPAPolicyTargets, and one or more SUPAPolicyMetadata objects.

3.2.2.4. SUPALogicStatement

A SUPALogicStatement is an abstract class that represents declarative (also called intent-based) policies. A SUPALogicStatement MUST contain at least one SUPAPolicyStatement. Such policies define a goal to be achieved, or a set of actions to take, but do not prescribe how to achieve the goal or execute the actions. This differentiates it from a SUPAECAPolicyRule, which explicitly defines what triggers the evaluation of the SUPAECAPolicyRule, what conditions must be satisfied in order to execute the actions of the SUPAECAPolicyRule, and what actions to execute.
3.2.2.5. SUPAMetadata

Metadata is, literally, data about data. SUPAMetadata is an abstract class that contains prescriptive and/or descriptive information about the object(s) that it is attached to. While metadata can be attached to any information model element, this document only considers metadata attached to classes and relationships.

When defined in an information model, each instance of the SUPAMetadata class MUST have its own aggregation relationship with the set of objects that it applies to. However, a data model MAY map these definitions to a more efficient form (e.g., flattening the object instances into a single object instance).

3.2.2.6. SUPAPolicyTarget

SUPAPolicyTarget is an abstract class that defines a set of managed objects that may be affected by the actions of a SUPAPolicyStatement. A SUPAPolicyTarget may use one or more mechanisms to identify the set of managed objects that it affects; examples include OIDs and URIs.

When defined in an information model, each instance of the SUPAPolicyTarget class MUST have its own aggregation relationship with each SUPAPolicy that uses it. However, a data model MAY map these definitions to a more efficient form (e.g., flattening the SUPAPolicyTarget, SUPAMetadata, and SUPAPolicy object instances into a single object instance).

3.2.2.7. SUPAPolicySubject

SUPAPolicySubject is an abstract class that defines a set of managed objects that authored this SUPAPolicyStatement. This is required for auditability. A SUPAPolicySubject may use one or more mechanisms to identify the set of managed objects that authored it; examples include OIDs and URIs.

When defined in an information model, each instance of the SUPAPolicySubject class MUST have its own aggregation relationship with each Policy that uses it. However, a data model MAY map these definitions to a more efficient form (e.g., flattening the PolicySubject, Metadata, and Policy object instances into a single object instance).
3.2.3. Modeling Terminology

The following terms define different types of relationships used in the information models of the SUPA Generic Policy Information Model (GPIM).

3.2.3.1. Inheritance

Inheritance makes an entity at a lower level of abstraction (e.g., the subclass) a type of an entity at a higher level of abstraction (e.g., the superclass). Any attributes and relationships that are defined for the superclass are also defined for the subclass. However, a subclass does NOT change the characteristics or behavior of the attributes or relationships of the superclass that it inherits from. Formally, this is called the Liskov Substitution Principle [7]. This principle is one of the key characteristics that is NOT followed in [RFC3060] and [RFC3460].

A subclass MAY add new attributes and relationships that refine the characteristics and/or behavior of it compared to its superclass.

3.2.3.2. Relationship

A relationship is a generic term that represents how a first set of entities interact with a second set of entities. A recursive relationship sets the first and second entity to the same entity. There are three basic types of relationships, as defined in the subsections below: associations, aggregations, and compositions.

3.2.3.3. Association

An association represents a generic dependency between a first and a second set of entities. In an information model, an association MAY be represented as a class.

3.2.3.4. Aggregation

An aggregation is a stronger type (i.e., more restricted semantically) of association, and represents a whole-part dependency between a first and a second set of entities. Three objects are defined by an aggregation: the first entity, the second entity, and a new third entity that represents the combination of the first and second entities. The entity owning the aggregation is referred to as the "aggregate", and the entity that is aggregated is referred to as the "part". In an information model, an aggregation MAY be represented as a class.
3.2.3.5. Composition

A composition is a stronger type (i.e., more restricted semantically) of aggregation, and represents a whole-part dependency with two important behaviors. First, an instance of the part is included in at most one instance of the aggregate at a time. Second, any action performed on the composite entity (i.e., the aggregate) is propagated to its constituent part objects. For example, if the composite entity is deleted, then all of its constituent part entities are also deleted. This is not true of aggregations or associations – in both, only the entity being deleted is actually removed, and the other entities are unaffected. In an information model, a composition MAY be represented as a class.

3.2.3.6. Association Class

A relationship may be implemented as an association class. This is used to define the relationship as having its own set of features. More specifically, if the relationship is implemented as an association class, then the attributes of the association class, as well as other relationships that the association class participates in, may be used to define the semantics of the relationship. If the relationship is not implemented as an association class, then no additional semantics (beyond those defined by the type of the relationship) are expressed by the relationship.

In an information model, a composition MAY be represented as a class.

3.2.3.7. Multiplicity

A specification of the range of allowable cardinalities that a set of entities may assume. This is always a pair of ranges, such as 1 - 1 or 0..n - 2..5.

3.2.3.8. Navigability

A relationship may have a restriction on the ability of an object at one end of the relationship to access the object at the other end of the relationship. This document defines two choices:

1. Each object is navigable by the other, which is indicated by NOT providing any additional symbology, or
2. An object A can navigate to object B, but object B cannot navigate to object A. This is indicated by an open-headed arrow pointing to the object that cannot navigate to the other object. In this example, the arrow would be pointing at object B.
3.2.3.9. Abstract Class

An abstract class is a class that cannot be directly instantiated.

3.2.3.10. Concrete Class

A concrete class is a class that can be directly instantiated.

3.2.4. Mathematical Logic Terminology

This section defines terminology for mathematical logic.

3.2.4.1. Predicate

A predicate is a Boolean-valued function (i.e., a function whose values are interpreted as either TRUE or FALSE, depending on the values of its variables).

3.2.4.2. Logic Operators

A logical connective is a symbol or word that defines how to connect two or more sentences in a language.

3.2.4.2.1. Propositional Logic Connectives

There are five propositional logic connectives, defined as follows:

- Negation, or a logical NOT operator, is an operation that, when applied to a proposition, produces a new proposition "not $p$", which has the opposite truth value of $p$.
- Conjunction, or a logical AND operator, is an operation on two logical values that produces a value of TRUE if and only if both of its operands are TRUE.
- Disjunction, or a logical OR operator, is an operation on two logical values that produces a value of FALSE if and only if both of its operands are FALSE.
- Implication, or the conditional operator, is used to form statements of the form "if $<\text{proposition}\ A\ >$ is TRUE, then $<\text{proposition}\ B\ >$ is also TRUE (i.e., this statement is FALSE only when $A$ is TRUE and $B$ is FALSE)."
- Bi-implication, or the bi-conditional operator, is used to form statements of the form "$<\text{proposition}\ A\ >$ is TRUE if and only if $<\text{proposition}\ B\ >$ is TRUE (i.e., this statement is TRUE if and only if both propositions are FALSE or if both propositions are TRUE)."
3.2.4.2.2. First Order Logic Quantifiers

This document defines two types of quantifiers for First Order Logic statements.

Quantification specifies the number of objects that satisfies a formula. This document uses two such quantifiers, which are defined as follows:

- Universal quantification asserts that a predicate within the scope of this operator is TRUE of every value of a variable of the predicate. It is commonly interpreted as "for all".
- Existential quantification asserts that a predicate within the scope of this operator is TRUE for at least one value of a variable of the predicate. It is commonly interpreted as "there exists, "there is at least one", or "for some".

3.2.4.3. Propositional Logic

Propositional Logic (PL) may be simply defined as a language consisting of a set of statements; the value of each statement is either TRUE or FALSE. More formally, a (propositional) Argument consists of a sequence of Premises and a Conclusion. An Argument is valid if the Conclusion is TRUE whenever all Premises are TRUE.

PL may be thought of as a set of declarative propositions.

3.2.4.4. First-Order Logic

First-Order Logic (FOL) may be simply defined as a language consisting of a set of statements; each statement is a predicate.

A predicate is a Boolean-valued function (i.e., the value of the function evaluates to either TRUE or FALSE, depending on the value of its variables). Predicates can also be compared.

FOL uses quantified variables. The universal quantifier and/or the existential quantifier can be used to define what values can be instantiated by the predicated variables.
3.3. Symbology

The following symbology is used in this document:

3.3.1. Inheritance

Inheritance: a subclass inherits the attributes and relationships of its superclass, as shown below:

```
+------------+
| Superclass |
+------------+  +------------+
 /          /  /          /
I           I  I           I
+------------+  +------------+
    Subclass
```

3.3.2. Association

Association: Class B depends on Class A, as shown below:

```
+---------+  +---------+  +---------+  +---------+
|         |  | Class A |  | Class B |  | Class A |  | Class B |
+---------+  +---------+  +---------+  +---------+
 association with no navigability restrictions
```

3.3.3. Aggregation

Aggregation: Class B is the part, Class A is the aggregate, as shown below:

```
+---------+  +---------+  +---------+  +---------+
|         |  | Class A |  | Class B |  | Class A |  | Class B |
+---------+  +---------+  +---------+  +---------+
 aggregation with no navigability restrictions
```

```
+---------+  +---------+  +---------+  +---------+
| Class A |  | Class B |  | Class A |  | Class B |
\ /      / \    \ /      / \    \ /      / \    \ /      / \    \ /      / \    \ /      / \  +---------+  +---------+  +---------+  +---------+
aggregation with navigability restrictions
```

Strassner           Expires January 4, 2016
3.3.4. Composition

Composition: Class B is the part, Class A is the composite, as shown below:

```
+---------+          +---------+          +---------+
|         |                |         |                |         |
| Class A | C ----| Class B |         | Class A | C ------| Class B |
|         |      \|     |         |      \     |         |      /     |
|         |                |         |                |         |                |
+---------+          +---------+          +---------+
```

composition with no navigability restrictions

composition with navigability restrictions

3.3.5. Association Class

Association Class: Class C is the association class implementing the relationship D between classes A and B

```
+---------+          +---------+
| Class A |           | Class B |
+---------+    ^     +---------+
                |
                |
+----------+----------+
| Association Class C |
+---------------------+
```

3.3.6. Logical Connectives

The following defines a mapping between the typical mathematical symbols used for logical connectives (most of which are in extended ASCII) and the symbols that will be used in this document.

<table>
<thead>
<tr>
<th>Connective</th>
<th>ASCII CODE</th>
<th>UNICODE Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation</td>
<td>172</td>
<td>U+00AC</td>
<td>&quot;NOT&quot;</td>
</tr>
<tr>
<td>Conjunction</td>
<td>8743</td>
<td>U+2227</td>
<td>&quot;AND&quot;</td>
</tr>
<tr>
<td>Disjunction</td>
<td>8744</td>
<td>U+2228</td>
<td>&quot;OR&quot;</td>
</tr>
<tr>
<td>Implication</td>
<td>8658</td>
<td>U+21D2</td>
<td>&quot;IMPLIES&quot;</td>
</tr>
<tr>
<td>Bi-implication</td>
<td>8660</td>
<td>U+21D4</td>
<td>&quot;IF AND ONLY IF&quot;</td>
</tr>
</tbody>
</table>

3.3.7. Quantifiers

The following defines a mapping between the typical mathematical symbols used for quantifiers and the symbols that will be used in this document.

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>ASCII Code</th>
<th>Unicode Code</th>
<th>Symbol Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal</td>
<td>8704</td>
<td>U+2200</td>
<td>&quot;FOR ALL&quot;</td>
</tr>
<tr>
<td>Existential</td>
<td>8707</td>
<td>U+2203</td>
<td>&quot;THERE EXISTS&quot;</td>
</tr>
</tbody>
</table>
4. Policy Abstraction Architecture

This section describes the motivation for the policy abstractions that are used in SUPA. The following abstractions are provided:

- The GPIM defines a technology-neutral information model that can express the concept of Policy.
- This version of this document restricts the expression of Policy to
  - a set of event-condition-action tuples
  - a set of PL or FOL statements
  - a combination of ECA rules and PL or FOL statements.
- Since these representations are very different in syntax and content, the content of a Policy is abstracted from its representation:
  - SUPAECAPolicyRules and SUPALogicStatements are each types of SUPAPolicies
  - SUPAECAPolicyRules and SUPALogicStatements are both constructed from SUPAPolicyStatements
  - A SUPAPolicy MAY use SUPAECAPolicyRules and/or SUPALogicStatements
  - A SUPAPolicy consists of one or more SUPAPolicyStatements, and optionally may specify one or more SUPAPolicyTarget, SUPAPolicySubject, and SUPAPolicyMetadata objects
- A SUPAPolicyStatement has three subclasses:
  - a SUPABooleanClause for building SUPAECAPolicyRules from reusable objects
  - a SUPALogicClause for building SUPALogicStatements from reusable objects
  - a SUPAENencodedClause for using attributes instead of objects to construct a SUPAECAPolicyRule or a SUPALogicStatement
- A SUPAPolicy MUST contain at least one SUPAPolicyStatement.
- A SUPAECAPolicyRule defines the set of events and conditions that are responsible for executing its actions; it MUST have an event clause, a condition clause, and an action clause.
- A SUPALogicStatement expresses facts that it believes to be true without defining how those facts are computed, and provides an efficient query mechanism for retrieving facts. Each SUPAPolicyStatement MUST be expressed as a function-free Horn clause; there are a number of additional restrictions that are covered in Section 7.
- SUPAMetadata MAY be defined for any type of SUPAPolicyStatement (as well as for individual objects that make up a SUPAPolicyStatement).
- SUPAMetadata MAY be prescriptive and/or descriptive in nature.
- A SUPAPolicyTarget is a set of managed objects that the actions of the SUPAPolicy are applied to.
- A SUPAPolicySubject is a set of managed objects that authored the SUPAPolicy.
4.1. Motivation

The power of policy management is its applicability to many different types of systems. There are many different actors that can use a policy management system, including end-users, operators, application developers, and administrators. Each of these constituencies have different concepts and skills, and use different terminology. For example, an operator may want to express an operational rule that states that only Platinum and Gold users can use streaming multimedia applications. As a second example, a network administrator may want to define a more concrete policy rule that looks at the number of dropped packets and, if that number exceeds a programmable threshold, changes the queuing and dropping algorithms used.

SUPA may be used to define other types of policies, such as for systems and operations management; an example is: "All routers and switches must have password login disabled". See section 3 of [8] for additional declarative and ECA policy examples.

All of the above examples are commonly referred to as "policy rules", but they take very different forms, since they are at very different levels of abstraction and typically authored by different actors. The first was very abstract, and did not contain any technology-specific terms, while the second was more concrete, and likely used technical terms of a general (e.g., IP address range, port numbers) as well as a vendor-specific nature (e.g., specific queuing, dropping, and/or scheduling algorithms implemented in a particular device). The third restricted the type of login that was permissible for certain types of devices in the environment.

Note that the first two policy rules could directly affect each other. For example, Gold and Platinum users might need different device configurations to give the proper QoS markings to their streaming multimedia traffic. This is very difficult to do if a common policy model does not exist.

More importantly, the users of these two policies likely have different job responsibilities. They may have no idea of the concepts used in each policy. Yet, their policies need to interact in order for the business to provide the desired service. This again underscores the need for a common policy framework.

Certain types of policy rules (e.g., ECA) may express actions, or other types of operations, that contradict each other. SUPA provides a rich object model that can be used to support language definitions that can find and resolve such problems.
4.2. SUPA Approach

The purpose of the SUPA Generic Policy Information Model (GPIM) is to define a common framework for expressing policies at different levels of abstraction. SUPA uses the GPIM as a common vocabulary for representing policy concepts that are independent of language, protocol, repository, and level of abstraction. This enables different actors to author and use policies at different levels of abstraction. This forms a policy continuum [1] [2], where more abstract policies can be translated into more concrete policies, and vice-versa.

Most systems define the notion of a policy as a single entity. This assumes that all users of policy have the same terminology, and use policy at the same level of abstraction. This is rarely, if ever, true in modern systems. The policy continuum defines a set of views (much like RM-ODP’s viewpoints [9]) that are each optimized for a user playing a specific role. SUPA defines the GPIM as a standard vocabulary and set of concepts that enable different actors to use different formulations of policy. This corresponds to the different levels in the policy continuum, and as such, can make use of previous experience in this area.

It may be necessary to translate a Policy from a general to a more specific form (while keeping the abstraction level the same). For example, the declarative policy "Every network attached to a VM must be a private network owned by someone in the same group as the owner of the VM" may be translated to more formal form (e.g., Datalog (as in OpenStack Congress) or a set of SUPALogicStatements). It may also be necessary to translate a Policy to a different level of abstraction. For example, the previous Policy may need to be translated to a form that network devices understand. This requires a common framework for expressing policies that is independent of the level of abstraction that a Policy uses.

4.3. SUPA Generic Policy Information Model Overview

Figure 1 illustrates the approach for representing policy rules in SUPA. The top two layers are defined in this document; the bottom layer (Data Models) are defined in separate documents. Conceptually, the GPIM defines a set of objects that define the key elements of a Policy independent of how it is represented or its content. As will be shown, there is a significant difference between SUPAECPolicyRules (see Section 6) and SUPALogicStatements (see Section 7). In principle, other types of SUPAPolicies could be defined, but the current charter is restricted to using these two types of SUPAPolicies as exemplars.
The GPIM defines the following concepts:

- **SUPAPolicy**: the root of the SPGIM model
- **SUPAPolicyAtomic**: a Policy that can be used in a stand-alone manner
- **SUPAPolicyComposite**: used to build hierarchies of Policies
- **SUPAPolicyStatement**: used to define the content of a SUPAPolicy
- **SUPAPolicyTerm**: used to define variables, operators, and values in a SUPAPolicyStatement
- **SUPAPolicySubject**: the author of a SUPAPolicy
- **SUPAPolicyTarget**: the managed object that a SUPAPolicy monitors and/or controls the state of
- **SUPAPolicyMetadata**: specifies descriptive and/or prescriptive information about a SUPAPolicy object

A SUPAPolicy object serves as a single root of the SUPA system (i.e., all other classes in the model are subclasses of the SUPAPolicy class). This simplifies code generation and reusability. Note that this is NOT true of either [4] or [6]; it is true of [2].

SUPA Policies are defined as either a stand-alone or a hierarchy of PolicyContainers. A PolicyContainer specifies the structure, content, and optionally, subject, target, and metadata information for the Policy.
A SUPAPolicy takes one of two forms: (1) an ECA Policy, and/or (2) a declarative set of statements. The GPIM enables both types of policies to be combined; this is also true of [2] and [5]. However, this is not true of [4] and [6], since neither define declarative policies (additional reasons are beyond the scope of this document).

The GPIM design enables a Policy of one type to invoke Policies of the other type. For example, a declarative policy may directly or indirectly invoke one or more ECA policies. Since declarative policies are at a higher level of abstraction, this is the typical case, though it is possible for the reverse to occur. For example, an ECA policy could invoke a declarative policy as one of its actions if it is more appropriate to specify a generic, as opposed to a prescriptive, course of action. This is also true of [2] and [5]; however, this is not true of [4] and [6], since neither define declarative policies (additional reasons are beyond the scope of this document).

Both a SUPAECAPolicyRule and a SUPALogicalStatement are made up of one or more SUPAPolicyStatements, which define the content of the Policy. Three types of SUPAPolicyStatements are available; one is generic, and can be used by any type of Policy, while the other two are specific to an ECA or a declarative Policy, respectively. The generic SUPAPolicyStatement, called SUPAEncodedClause, encodes the policy as an attribute. In contrast, the two specific types of SUPAPolicyStatements (SUPABooleanClause and SUPALogicClause, which are used for ECA and declarative policies, respectively), typically are made up of (reusable) SUPAPolicy objects. All three may thus be constructed at runtime by a machine. This is also true of [2] and [5]; however, this is not true of [4] and [6], since both lack the abstraction of a common PolicyStatement.

A SUPAPolicyStatement may be made up of SUPAPolicyTerms, which enables a clause to be defined in a canonical {variable, operator, value} form. In addition, specific objects for constructing ECA Policies and declarative Policies are also provided. This is provided to enable machine-driven construction of policies.

This set of classes enables each different types of Policies to be defined by an information model that refines the generic concepts of the GPIM as described above. For example, a SUPAECAPolicyRule, as well as a SUPALogicStatement, are both subclasses of the SUPAPolicyAtomic class. Therefore, both can be used as part of a hierarchy of Policies or in a stand-alone manner. As another examples, a SUPALogicClause and a SUPABooleanClause are both subclasses of SUPAPolicyStatement, and are used to create SUPALogicStatements and SUPAECAPolicyRules, respectively.
4.4. Structure of SUPA Policies

This section describes the overall design of the GPIM.

4.4.1. ECA Policy Rule Structure

A SUPAECAPolicyRule is a statement that consists of an event clause, a condition clause, and an action clause. This type of Policy explicitly defines the current and desired states of the system being managed. Conceptually, it is represented as follows:

```
+------------------+                  +---------------------+
| SUPAPolicyAtomic |                  | SUPAPolicyStatement |
+---------+--------+                  +----------+----------+
\                            / \                        / \
I                                I                           I
I                                I                           I
I                                I                           I
I                                I                           I
I                                I                           I
+-----------------+                  +---------------------+  
| SUPAECAPolicyRule |                  | SUPABooleanClause or |
|                  |                  | SUPAEncodedClause |
+-----------------+                  +---------------------+  
|                  |                  | 0..1/ \ 0..1/ \ 0..1/ \ |
A                   A                   A                   A
\                   \                   \                   \ 
/ /                   / /                   / /                   / / 
| |                   | |                   | |                   | |
I I                   I I                   I I                   I I
I I                   I I                   I I                   I I
I I                   I I                   I I                   I I
I I                   I I                   I I                   I I
|                    |                    |                     |
|                    |                    |                     |
\                    \                    \                    \  
/                     /                     /                     /  
|                     |                     |                     |  
Event                 Clause                 Condition            Action
|                     |                     |                     |  
1..n/                  1..n/                  1..n/                  1..n/  
+-----------------+                  +---------------------+  
| HasSUPAEEvents |                  | HasSUPAConditions   |
+-----------------+                  +---------------------+  
```

Figure 2: Overview of SUPA Policy Rule Abstractions

Note that the event, condition, and action clauses may be defined by using SUPABooleanClauses or SUPAEncodedClauses. See section 6.5 for the former and 5.5 (especially 5.5.2.1) for the latter.
4.4.2. Logical Statement Structure

A SUPALogicStatement is either a set of PL or FOL statements. These are called SUPAPLStatement and SUPAFOLStatement, respectively.

A SUPAPLStatement is a set of propositions that form a (single) conclusion. A proposition is either TRUE or FALSE. A proposition can be created from simpler propositions combined using Propositional Logic Connectives (see Section Propositions (see Section 3.2.4.2.1.). It may be conceptualized as follows:

```
+---------------------+
| SUPAPLogicStatement |
+---------------------+
 /
 I
 I
 /
 I
 I
```

```
+---------------------+----------------------------------------+
| SUPAPLPremise | SUPAPLConclusion | SUPAPLArgument |
+---------------------+----------------------------------------+
1..n /
 I
 I
 I
 |
 +---------------------+----------------------------------------+
|                      | HasSUPAPLConclusion                    |
+---------------------+----------------------------------------+
1 /
 | HasSUPAPLPremises |
```

Figure 3: Overview of SUPA Propositional Logic Abstractions

As shown in Figure 4, a SUPAPLArgument consists of a set of one or more SUPAPLPremises and a single SUPAPLConclusion. The multiplicity of the two aggregations is 0..1 on the aggregate side to enable SUPAPLPremises and SUPAPLConclusions to be created and stored independently of being used in a SUPAPLArgument.
In PL, each possible atomic fact requires a separate propositional symbol. This can lead to a large amount of premises required to form a conclusion.

FOL provides a richer knowledge representation by using:

- objects (i.e., terms), which define individual entities
- properties (i.e., unary predicates on terms), which distinguishes objects from each other
- relations (i.e., n-ary predicates on terms), which define facts among a set of objects, and
- functions (i.e., the mapping from one set of terms to another set of terms).

FOL may be conceptualized as follows:

![Diagram of SUPA FOL Abstractions]

Figure 4: Overview of SUPA First Order Logic Abstractions
FOL Syntax may be described using the following grammar:

Sentence
: AtomicSentence
| Sentence Connective Sentence
| (Quantifier Variable)+ Sentence
| 'NOT' Sentence
| function '( ' Sentence ')'
;

4.5. GPIM Assumptions

Most policy models (e.g., [2], [4], [5], and [6]) are built as part of an overarching model. SUPA DOES NOT assume that it is the "root class of everything". Rather, the SUPA information model is built as a single inheritance model fragment to accommodate inserting the SUPA model into another model (e.g., the root of the SUPA model becomes a subclass of the other model). This is shown in Figure 5.

```
+------------------------------------------+
|     Root Class of an Existing Model      |
+--------------------+---------------------+
/ \                     / \
I                        I
I                        I
I                        I
+-----------------+--------------+
|    A Subclass of the |       | Another Subclass of the |
|     Existing Model  |       |      Existing Model     |
+-----------------------+       +------------+------------+
/ \                     / \
I                        I
I                        I
I                        I
+-----+-----+
| SUPA Class Hierarchy  | (GPIM plus EPRIM |
|   (GPIM plus EPRIM    | and/or LSIM)    |
|      and/or LSIM)     |
+------------------------+
```

Figure 5: Integrating SUPA into an Existing Model
4.6. Scope of Previous Work

Insert intro paragraph and reference SUPA Gap Analysis [6]. Some salient points on previous policy models:

- [RFC3060] and [RFC3460] only define a policy rule that consists of a condition clause and an action clause; it does not define an ECA policy rule, nor does it define a LogicStatement
- [4] is more elaborate than [RFC3060] and [RFC3460], but suffers from the same limitations
- [5] defines four types of policies (i.e., ECA, Goal, UtilityFunction, and Promise), but does not have the detail defined in this document

Rest to be finished. Sections will include:

- Description of, and problems with, [RFC3060]
- Description of, and problems with, [RFC3460]
- Should this section also talk about CIM or SID? I personally think that this should be in the gap analysis...
5. GPIM Model

This section defines the classes and relationships of the GPIM.

5.1. Overview

The overall class definition is shown in Figure 6. SUPAPolicy is the root of the SUPA class hierarchy. For implementations, it is assumed that SUPAPolicy is subclassed from a class from another model. In Figure 6, indentation represents subclassing.

(Class of another model that SUPA is integrating into)

+-SUPAPolicy (see Section 5.2)
  |  +--SUPAPolicyAtomic (see Section 5.3)
  |  +--SUPAPolicyComposite (see Section 5.4)
  |  +--SUPAPolicyStatement (see Section 5.5)
  |  +--SUPAPolicySubject (see Section 5.6)
  |  +--SUPAPolicyTarget (see Section 5.7)
  |  +--SUPAPolicyTerm (see Section 5.8)
  |  +--SUPAPolicyMetadata (see Section 5.9)

Figure 6: Main Classes of the GPIM

The following subsections define the classes of the GPIM. If a class has attributes, those attributes are also defined. Relationships are defined according to the class that is the "owner", or primary actor, participating in the relationship.

Classes, attributes, and relationships that are marked as "mandatory" MUST be part of a conformant implementation. Classes, attributes, and relationships that are marked as "optional" SHOULD be part of a conformant implementation.

5.2. The Abstract Class "SUPAPolicy"

This is a mandatory abstract class. This class is the root of the SUPA class hierarchy. It defines the common attributes and relationships that all SUPA subclasses inherit. SUPAPolicy was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in [5].
A SUPAPolicy takes the form of an individual policy or a set of policies. The former is defined by an instance of a concrete subclass of the SUPAPolicyAtomic class. The latter consists of a set of PolicyContainers; each PolicyContainer is an instance of a SUPAPolicyComposite (or a subclass of it). Each SupaPolicyComposite can have zero or more instances of a concrete subclass of the SUPAPolicyAtomic class.

In this approach, a SUPAPolicyAtomic class is the superclass of SUPAECAPolicyRule and SUPALogicStatement. These two classes define the type of policy that is being defined. A SUPAPolicyComposite is a PolicyContainer with SUPAMetadata attached. Each PolicyContainer forms its own containment hierarchy, and SUPAMetadata can be used to describe and/or prescribe how policies behave within each PolicyContainer.

Each SUPAPolicyAtomic object (or a subclass of it) MUST have at least one SUPAPolicyStatement that is used to define the content of the policy. Some types of policies, such as an ECA policy rule, MUST have at least three SUPAPolicyStatements, one each to define its event, condition, and action clauses. SUPALogicStatements typically have multiple SUPAPolicyStatements.

A SUPAPolicy MAY be qualified (i.e., may aggregate these objects to more completely specify the behavior of the SUPAPolicy) by a set of zero or more SUPAPolicySubjects, SUPAPolicyTargets, and/or SUPAPolicyMetadata objects. Note that these three classes are defined as abstract, in order to simplify mapping to, and optimization of, data models. When defined in an information model, the SUPAPolicyAtomic and SUPAPolicyComposite classes MUST have separate aggregation relationships with the SUPAPolicySubject and SUPAPolicyTarget objects (or their subclasses), if these objects are defined. Any subclass of SUPAPolicy that wants to use metadata MUST have one or more separate aggregation relationships with a SUPAPolicyMetadata class (or its subclasses). When implemented in a data model, the set of SUPAPolicyStatement, SUPAPolicyTarget, SUPAPolicySubject, and SUPAPolicyMetadata object instances SHOULD all be part of a single PolicyContainer object. They MAY be translated to a more efficient form (e.g., flattening the objects that are participating in the above relationships into a single object instance).

Figure 7 shows the SUPAPolicy class, and two of its subclasses (SUPAPolicyAtomic and SUPAPolicyComposite). This is an implementation of the composite pattern [3], which enables a SUPAPolicy to be made up of a stand-alone object (an instance of a SUPAPolicyAtomic class) or a hierarchy of objects (i.e., instances of one or more SUPAPolicyAtomic and SUPAPolicyComposite classes). The use of this software pattern enables SUPA Policies to be designed as individual objects and/or hierarchies of objects.
Note that a SUPAPolicy, as well as a SUPAPolicyAtomic and a SUPAPolicyComposite, are all PolicyContainer objects. SUPAPolicy was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

In figure 7:

- Both SUPAPolicyComposite and SUPAPolicyAtomic inherit from SUPAPolicy
- The diamond with an enclosed "A" represents an aggregation (see Section 3.2.3.4)
- The HasSUPAPolicies aggregation is implemented as an association class (see Section 3.2.3.6)
- The multiplicity of the HasSUPAPolicies aggregation is 0..1 - 1..n (zero or one SUPAPolicyComposite object instances can aggregate one or more SUPAPolicy object instances, see Section 3.2.3.7)
- The arrow pointing at SUPAPolicy restricts the navigability of this aggregation (see Section 3.2.3.8)
5.2.1. SUPAPolicy Attributes

This section defines the attributes of the SUPAPolicy class. These attributes are inherited by all subclasses of the SUPAPolicy class.

5.2.1.1. The Attribute "supaObjectIDContent"

This is a mandatory attribute that represents part of the object identifier of an instance of this class. It is a string attribute, and defines the content of the object identifier. It works with another class attribute, called supaObjectIDFormat, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of an object identifier for the object instance of this class. This is based on the DEN-ng class design [2].

One of the goals of SUPA is to be able to generate different data models that support different types of protocols and repositories. This means that the notion of an object ID must be generic. In this way, different naming schemes, such as those depending on URIs, FQDNs, primary key – foreign key relationships, and UUIDs can all be accommodated.

5.2.1.2. The Attribute "supaObjectIDFormat"

This is a mandatory attribute that represents part of the object identifier of an instance of this class. It is a string attribute, and defines the format of the object identifier. It works with another class attribute, called supaObjectIDContent, which defines the content of the object ID. These two attributes form a tuple, and together enable a machine to understand the syntax and value of an object identifier for the object instance of this class. This is based on the DEN-ng class design [2].

5.2.1.3. The Attribute "supaPolicyDescription"

This is an optional string attribute that defines a free-form textual description of this object.

5.2.1.4. The Attribute "supaPolicyName"

This is an optional string attribute that defines the name of this Policy. This enables any existing generic naming attribute to be used for generic naming, while allowing this attribute to be used to name Policy entities in a common manner. Note that this is NOT the same as the commonName attribute of the Policy class defined in RFC3060 [RFC3060], as that attribute is intended to be used with just X.500 cn attributes.
5.2.2. SUPAPolicy Relationships

This section defines the relationships of the SUPAPolicy class.

5.2.2.1. The Relationship "SUPAPolicyMetadata"

This is a mandatory aggregation that defines the set of SUPAPolicyMetadata that are aggregated by this particular SUPAPolicy object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAPolicy) side, and 0..n on the part (SUPAPolicyMetadata) side. This means that this relationship is optional, but if it is instantiated, then one or more SUPAPolicy objects are contained in this particular SUPAPolicyComposite object. The semantics of this aggregation are implemented using the SUPAPolicyMetadataDetail association class.

5.2.2.2. The Association Class "SUPAPolicyMetadataDetail"

This is a mandatory concrete association class that defines the semantics of the SUPAPolicyMetadata aggregation. This enables the attributes and relationships of the SUPAPolicyMetadataDetail class to be used to constrain which SUPAPolicyMetadata objects can be aggregated by this particular SUPAPolicy object instance.

Attributes will be added to this class at a later time.

5.3. The Abstract Class "SUPAPolicyAtomic"

This is a mandatory abstract class. This class is a type of PolicyContainer. SUPAPolicyAtomic was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

A SUPAPolicyAtomic class represents a SUPA Policy that can operate as a single, stand-alone, manageable object. Put another way, a SUPAPolicyAtomic object can NOT be modeled as a set of hierarchical SUPAPolicy objects; if this functionality is required, then a SUPAPolicyComposite object must be used.

The SUPAPolicyAtomic class is the superclass for the different types of SUPA Policies that are defined. In this release, both a SUPAECAPolicyRule (see Section 6) as well as a SUPALogicStatement (see Section 7) are defined as its subclasses.
5.3.1. SUPAPolicyAtomic Attributes

This section defines the attributes of the SUPAPolicyAtomic class.

Care must be taken in adding attributes to this class, because the behavior of SUPALogicStatements is very different than the behavior of SUPAECAPolicyRules.

5.3.1.1. The Attribute "supaPolicyDeployStatus"

This is an optional attribute, which is an enumerated, non-negative integer. It defines the current deployment status of this SUPAECAPolicyRule. Both operational and test mode values are included in its definition. Values include:

0: undefined
1: deployed and enabled
2: deployed and in test
3: deployed but not enabled
4: ready to be deployed
5: not deployed

5.3.1.2. The Attribute "supaPolicyExecStatus"

This is an optional attribute, which is an enumerated, non-negative integer that defines the current execution status of this SUPAECAPolicyRule. Both operational and test mode values are included in its definition. Values include:

0: undefined
1: executed and SUCEEDED (operational mode)
2: executed and FAILED (operational mode)
3: currently executing (operational mode)
4: executed and SUCEEDED (test mode)
5: executed and FAILED (test mode)
6: currently executing (test mode)

5.3.2. SUPAPolicyAtomic Relationships

This section defines the relationships of the SUPAPolicyAtomic class.

5.3.2.1. The Aggregation "SUPAPAHasPolicyStmts"

This is a mandatory aggregation that defines the set of SUPAPolicyStatements that MUST be aggregated by this particular SUPAPolicyAtomic object. The multiplicity of the aggregation is 0..n on the aggregate (SUPAPolicyAtomic) side, and 1..n on the part (SUPAPolicyStatement) side.
The "0..n" part of the multiplicity enables SUPAPolicyStatements to be stored in a repository for later use, while the "1..n" part of the multiplicity mandates that at least one SUPAPolicyStatement MUST be aggregated by this particular SUPAPolicyAtomic object. More specifically, a cardinality of "3..n" on the part side is incorrect, since that would also apply to SUPALogicStatements. The semantics of this aggregation are defined by the SUPAPAHasPolicyStmtDetail association class.

5.3.2.2. The Association Class "SUPAPAHasPolicyStmtDetail"

This is an association class, and defines the semantics of the SUPAPAHasPolicyStmts aggregation. This enables the attributes and relationships of the SUPAPAHasPolicyStmtDetail class to be used to constrain which SUPAPolicyStatements objects can be aggregated by this particular SUPAPolicyAtomic object instance. Attributes will be added to this class at a later time.

5.4. The Concrete Class "SUPAPolicyComposite"

This is a mandatory concrete class. This class is a type of PolicyContainer. SUPAPolicyComposite was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

A SUPAPolicyComposite class represents a SUPA Policy as a hierarchy of Policy objects, where the hierarchy contains instances of a SUPAPolicyAtomic and/or SUPAPolicyComposite object. Each of the SUPA Policy objects, including the outermost SUPAPolicyComposite object, are separately manageable. More importantly, the SUPAPolicyComposite object can aggregate any SUPAPolicy subclass. Hence, it can be used to form hierarchies of SUPAPolicies as well as associate SUPAPolicySubjects and/or SUPAPolicyTargets to a given SUPAPolicy.

5.4.1. SUPAPolicyComposite Attributes

This section defines the attributes of the SUPAPolicyComposite class. The combination of these two attributes provides a more flexible and powerful solution compared to [RFC3060] and [RFC3460].

Care must be taken in adding attributes to this class, because the behavior of SUPALogicStatements is very different than the behavior of SUPAECAPolicyRules.
5.4.1.1. The Attribute "supaPCFailureStrategy"

This is an optional non-negative enumerated integer attribute, whose values are used to define what action(s) should be taken if a failure occurs when executing a SUPAPolicy object that is contained in this SUPAPolicyComposite object. Values include:

- 0: undefined
- 1: stop execution
- 2: attempt rollback on failed policy
- 3: attempt rollback on all policies
- 4: ignore failure and continue

A value of 0 can be used as an error condition. A value of 1 means that ALL execution is stopped, and that other SUPAPolicies that otherwise would have been executed are ignored. A value of 2 means that execution is stopped, and a rollback of that SUPAPolicy (and ONLY that SUPAPolicy) is attempted. A value of 3 means that execution is stopped, and all SUPAPolicies that have been previously executed (including the one that just failed) are rolled back. A value of 4 means that any failure will be ignored, and all SUPAPolicies contained in this SUPAPolicyComposite object will be executed.

5.4.1.2. The Attribute "supaPCIsMatchAll"

This is an optional Boolean attribute. If its value is TRUE, then ALL SUPAPolicies that are contained in this SUPAPolicyComposite object will be evaluated, regardless of whether a SUPAPolicy fails to execute correctly or not. If its value is FALSE, then only the FIRST SUPAPolicy contained in this SUPAPolicyComposite object will be evaluated. The default value is TRUE.

5.4.2. SUPAPolicyComposite Relationships

This section defines the relationships of SUPAPolicyComposite.

5.4.2.1. The Aggregation "HasSUPAPolicies"

This is a mandatory aggregation that defines the set of SUPAPolicies that are contained in the instance of this particular SUPAPolicyComposite object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAPolicyComposite) side, and 1..n on the part (SUPAPolicy) side. This means that this relationship is optional, but if it is instantiated, then one or more SUPAPolicy objects are contained in this particular SUPAPolicyComposite object. The semantics of this aggregation are implemented using the HasSUPAPolicyDetail association class.
5.4.2.2. The Association Class "HasSUPAPolicyDetail"

This is a mandatory concrete association class that defines the semantics of the HasSUPAPolicies aggregation. This enables the attributes and relationships of the HasSUPAPolicyDetail class to be used to constrain which SUPAPolicy objects can be aggregated by this particular SUPAPolicyComposite object instance. Attributes will be added to this class at a later time.

5.5. The Abstract Class "SUPAPolicyStatement"

This is a mandatory abstract class that separates the representation of a SUPAPolicy from its implementation. This abstraction is missing in [RFC3060], [RFC3460], [4], and [6]. SUPAPolicyStatement was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

A SUPAPolicyStatement contains an individual or group of related functions that are used to build different types of policies. This is implemented using the composite pattern [3], so hierarchies of SUPAPolicyStatements can be constructed. Note that the hierarchy is available for two of its three subclasses: SUPABooleanClause (which is for forming ECA policy rules) and SUPALogicClause (which is for forming declarative policies). SUPAEncodedClause does not need to use this pattern, since it already encoded the clause.

This document defines three different types of policies: ECA policy rules, declarative policies, and encoded policies. These different types of policies can be used to define a goal to be achieved, a set of facts to be used, or a set of actions to take. Examples of actions include getting data from network elements or other sources, stating facts about the system being managed, writing a change to a configuration of one or more managed objects, and querying information about one or more managed objects.

SUPAPolicyStatements are objects in their own right, which facilitates their reuse. SUPAPolicyStatements can aggregate a set of SUPAPolicyTerms and/or a set of SUPAECAComponents. The former enables a SUPAPolicyStatement to be constructed from a set of SUPAPolicy objects, which emphasizes reusability and facilitates machine-directed policy authoring. The latter enables a higher level of abstraction to be applied to SUPAECAPolicyRules (i.e., the Event, Condition, and Action clauses can be filled by the subclasses of SUPAECAComponents (SUPAEvent, SUPACondition, and SUPAAction).
SUPAPolicyStatements can also be aggregated by a SUPAPolicyAtomic object - this enables both ECA policy rules and declarative policy statements to use SUPAPolicyStatements to form their policy rules.

A class diagram showing SUPAPolicyStatement is shown in Figure 8.

Figure 8: SUPAPolicyStatements and SUPAPolicy Classes

Note that the SUPAPAHasPolicyStmts aggregation was defined in Section 5.3.2.1.
Note that in Figure 8:

- SUPAPolicyStatement is a subclass of SUPAPolicy
- SUPAPolicyStatements are aggregated by SUPAPolicyAtomic, which is a subclass of SUPAPolicy
- SUPAPolicyComposite aggregates SUPAPolicyAtomic as well as SUPAPolicyComposite instance
- A SUPAEncodedClause is a subclass of SUPAPolicyStatement, and may be used by either a SUPAECAPolicyRule or a SUPALogicStatement
- SUPAPolicyStatement has two subclasses that are not shown in Figure 8: SUPABooleanClause and SUPALogicClause (they are discussed in Sections 6.5 and 7.4, respectively)
- Both the HasSUPAPolicyStatements and the HasSUPAPolicies aggregations are implemented as association classes

When defined in an information model, a SUPAPolicyStatement MUST be represented as a separate object that aggregates its constituent components. However, a data model MAY map this definition to a more efficient form (e.g., flattening the SUPAPolicyStatement and its aggregated object instances into a single object instance).

5.5.1. SUPAPolicyStatement Attributes

This section defines the attributes of the SUPAPolicyStatement class. These attributes are inherited by all subclasses of the SUPAPolicyStatement class.

5.5.1.1. The Attribute "supaPolicyStmtAdminStatus"

This is an optional attribute, which is an enumerated non-negative integer. It defines the current administrative status of this SUPAPolicyStatement.

This attribute can be used to place this particular SUPAPolicyStatement into a specific administrative state, such as enabled, disabled, or in test. Note that since a SUPAPolicy (e.g., a SUPAECAPolicyRule or a SUPALogicStatement) is made up of SUPAPolicyStatements, this enables all or part of a SUPAPolicy to be administratively controlled. Values include:

- 0: Unknown (an error state)
- 1: Enabled
- 2: Disabled
- 3: In Test (i.e., no operational traffic can be passed)

Value 0 denotes an error that prevents this SUPAPolicyStatement from being used. Values 1 and 2 mean that this SUPAPolicyStatement is administratively enabled or disabled, respectively. A value of 3 means that this SUPAPolicyStatement is in a special test mode and SHOULD NOT be used as part of an OAM&P policy.
5.5.1.2. The Attribute "supaPolStmtExecStatus"

This is an optional attribute, which is an enumerated non-negative integer. It defines whether this SUPAPolicyStatement is currently in use and, if so, what its execution status is.

This attribute can be used to place this particular SUPAPolicyStatement into a specific execution state, such as enabled, disabled, or in test. Note that since a SUPAPolicy (e.g., a SUPAECAPolicyRule or a SUPALogicStatement) is made up of SUPAPolicyStatements, this enables all or part of a SUPAPolicy to be administratively controlled. Values include:

- 0: Unknown (an error state)
- 1: Completed (i.e., successfully executed, but now idle)
- 2: Working (i.e., in use and no errors reported)
- 3: Not Working (i.e., in use, but errors have been reported)
- 4: In Test (i.e., cannot be used as part of an OAM&P policy)
- 5: Available (i.e., could be used, but currently isn’t)
- 6: Not Available (i.e., not available for use)

Value 0 denotes an error that prevents this SUPAPolicyStatement from being used. Value 1 means that this SUPAPolicyStatement has successfully finished execution, and is now idle. Values 1-3 mean that this SUPAPolicyStatement is in use; in addition, this SUPAPolicyStatement is working correctly, not working correctly, or in a special test state, respectively. A test state signifies that it SHOULD NOT be used to evaluate OAM&P policies. Values 4-5 mean that this SUPAPolicyStatement is not currently in use; a value of 4 means that it is available and could be used, while a value of 5 means that it is unavailable.

5.5.1.3. The Attribute "supaPolStmtIsCNF"

This is an optional Boolean attribute. If its value is true, then this SUPAPolicyStatement is in Conjunctive Normal Form (CNF). In Boolean logic, a formula that is in CNF is conceptually an AND of ORs. Otherwise, it is in Disjunctive Normal Form (i.e., an OR of ANDs).

5.5.2. SUPAPolicyStatement Subclasses

As stated before, the primary purpose of SUPAPolicyStatement is to define a common type of Policy statement that can be used to represent policy content regardless of the type of SUPAPolicy that is being used (e.g., it is independent of the requirements of a SUPAECAPolicyRule or a SUPALogicStatement).
The GPIM currently defines one subclass of SUPAPolicyStatement, called a SUPAEncodedClause, which can be used by both SUPAECAPolicyRules as well as SUPALogicStatements. Note that subclasses dedicated to the specific use of a SUPAECAPolicyRule and a SUPALogicStatement are defined in Sections 6.5 and 7.4, respectively.

There are three principal subclasses of SUPAPolicyStatement:

- SUPAEncodedClause, which is a mechanism to directly encode the content of the SUPAPolicyStatement into a set of attributes; this is described in more detail in Section 5.5.2.1.
- SUPABooleanClause, which defines a SUPAPolicyStatement as a set of one or more clauses; multiple clauses may be combined with Boolean AND and OR operators. This defines a SUPAPolicy as a completely reusable set of SUPAPolicy objects that are structured in an ECA form, and is described in more detail in Section 6.5.
- SUPALogicClause, which defines a SUPAPolicyStatement as either a fact or a clause; both are expressed in first-order logic. This defines a SUPAPolicy as a completely reusable set of SUPAPolicy objects that are structured in FOL, and is described in more detail in Section 7.4.

A SUPAPolicy MAY be constructed using any combination of the above three subclasses.

Both SUPAECAPolicyRules (see Section 6) and SUPALogicStatements (see section 7) MUST use SUPAPolicyStatements to define their content. This enables the content of these different types of Policies to be represented in a common manner.

Both SUPAECAPolicyRules and SUPALogicStatements MAY use a SUPAEncodedClause to define their content.

SUPAECAPolicyRules SHOULD also use a SUPABooleanClause to define its content, while SUPALogicStatements SHOULD also use a SUPALogicClause to define its content.

5.5.2.1. The Concrete Class "SUPAEncodedClause"

This is a mandatory concrete class that specializes (i.e., is a subclass of) a SUPAPolicyStatement. It defines a generalized extension mechanism for representing SUPAPolicyStatements that have not been modeled with other SUPAPolicy objects. Rather, the Policy Clause is directly encoded into the attributes of the SUPAEncodedClause. Note that other subclasses of SUPAPolicyStatement use SUPAPolicy objects to define their content.
This class provides the developer a tradeoff of efficiency vs. reusability. SUPAEncodedClause was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

This class uses two of its attributes (supaPolicyClauseContent and supaPolicyClauseFormat) for defining the content and format of a vendor-specific policy statement. This allows direct encoding of the policy statement, without having the "overhead" of using other objects. However, note that while this method is efficient, it does not reuse other SUPAPolicy objects. Rather, it can be thought of as a direct encoding of the policy statement. SUPAEncodedClause was abstracted from DEN-ng [2].

5.5.2.1.1. SUPAEncodedClause Attributes

This section defines the attributes of the SUPAEncodedClause class.

5.5.2.1.1.1. The Attribute "supaClauseContent"

This is a mandatory string attribute, and defines the content of this encoded clause of this clause. It works with another attribute of the SUPAEncodedClause class, called supaClauseFormat, which defines how to interpret this attribute. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the encoded clause for the object instance of this class. This is based on the DEN-ng class design [2].

5.5.2.1.1.2. The Attribute "supaClauseFormat"

This is a mandatory string attribute, and defines the format of this encoded clause. It works with another attribute of the SUPAEncodedClause class, called supaClauseContent, which defines the content (i.e., the value) of the encoded clause. These two attributes form a tuple, and together enable a machine to understand the syntax and value of the encoded clause for the object instance of this class. This is based on the DEN-ng class design [2].

5.5.2.1.1.3. The Attribute "supaClauseResponse"

This is an optional Boolean attribute that emulates a Boolean response of this clause, so that it may be combined with other subclasses of the SUPAPolicyStatement that provide a status as to their correctness and/or evaluation state.
5.5.3. SUPAPolicyStatement Relationships

This section defines the relationships of the SUPAPolicyStatement class.

5.5.3.1. The Aggregation "SUPAPolicyTermsInStmt"

This is a mandatory aggregation that defines the set of SUPAPolicyTerms that are aggregated by this instance of this SUPAPolicyStatement. This enables a SUPAPolicyStatement to be constructed from a (variable, operator, value) 3-tuple. The multiplicity of this relationship is defined as 0..n on the aggregate (SUPAPolicyStatement) side, and 0..n on the part (SUPAPolicyTerm) side. This means that this relationship is optional, but if it is instantiated, then a given SUPAPolicyStatement may contain zero or more SUPAPolicyTerms, and a given SUPAPolicyTerm may be contained in zero or more SUPAPolicyStatements. This enables SUPAPolicyTerms to be stored in a PolicyRepository before they are bound to one or more SUPAPolicyStatements (and vice-versa). The semantics of this aggregation are implemented using the SUPAPolicyTermsInStmtDetail association class.

5.5.3.2. The Association Class "SUPAPolicyTermsInStmtDetail"

This is a mandatory abstract association class that defines the semantics of the SUPAPolicyTermsInStmt aggregation. This enables the attributes and relationships of the SUPAPolicyTermsInStmt class to be used to constrain which SUPAPolicyStatement objects can aggregate which SUPAPolicyTerm objects.

Attributes will be added to this class at a later time.

5.5.3.2.1 SUPAPolicyStmtDetail Attributes

No attributes are currently defined for the SUPAPolicyStmtDetail association class.

5.6. The Abstract Class "SUPAPolicySubject"

This is an optional class that defines the set of managed entities that authored, or are otherwise responsible for, this SUPAPolicyStatement. Note that a SUPAPolicySubject does NOT evaluate or execute SUPAPolicies. Its primary use is for auditability.
A SUPAPolicySubject SHOULD be mapped to a role (e.g., using the role-object pattern, as DEN-ng does). This enables role-based access control to be used to restrict which entities can author a given policy. Note that Role is a type of Metadata.

SUPAPolicySubject was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5]. A class diagram is shown in Figure 9.

```
+------------------------+
| HasSUPAPolicyTgtDetail |
+-+------------------------+
|                     ^   |
|                     0..n+-+----+
|                     ^   \    \
|                     SUPAPolicyTargets/    SUPAPolicy |
|                     0..n \                         |
|                     ^   |
|                     /\
|                     HasSUPAPolicySubjDetail I
|                     0..1 +-------------------------+
|                     /\
|                     | SUPAPolicySubject I
|                     0..1 +---------+----------+
|                     / \
|                     | SUPAPolicy Targets I
|                     +-------------------+
|                     I
|                     I
|                     I
|                     I
|                     I
|                     I
|                     I
|                     (other SUPAPolicy subclasses)
```

Figure 9. SUPAPolicySubject and SUPAPolicyTarget
In Figure 9:

- SUPAPolicySubject and SUPAPolicyTarget are both subclasses of SUPAPolicy.
- Both the SUPAPolicyTargets and the SUPAPolicySubjects aggregations are implemented as association classes.
- The multiplicity of both of the above aggregations are 0..1 on the aggregate (SUPAPolicySubject or SUPAPolicyTarget) side and 0..n on the target (i.e., SUPAPolicy) side, respectively. This means that both aggregations are optional. If either is instantiated, then a SUPAPolicySubject or a SUPAPolicyTarget MAY contain zero or more SUPAPolicy object instances.

This model assumes that appropriate SUPAPolicySubject and SUPAPolicyTarget objects are added as subclasses of SUPAPolicy. If this is not the case, then the part side of both aggregations MUST be changed to where those objects are identified.

5.6.1. SUPAPolicySubject Attributes

This section defines the attributes of the SUPAPolicySubject class.

5.6.1.1. The Attribute "supaPolicySubjectIsAuthenticated"

This is an optional Boolean attribute. If the value of this attribute is true, then this SUPAPolicySubject has been authenticated and can be used to verify the authorship of a SUPAPolicy.

5.6.2. SUPAPolicySubject Relationships

This section defines the relationships of the SUPAPolicySubject class.

5.6.2.1. The Relationship "SUPAPolicySubjects"

This is an optional association that defines the set of SUPAPolicySubjects that are associated with this particular SUPAPolicy object. The multiplicity of this relationship is defined as 0..1 on the SUPAPolicySubject side, and 0..n on the SUPAPolicy side. This means that this relationship is optional, but if it is implemented, then this particular SUPAPolicy object was authored by this set of SUPAPolicySubjects. The semantics of this association are implemented using the SUPAPolicySubjectDetail association class.
5.6.2.2. The Association Class "SUPAPolicySubjectDetail"

This is an optional concrete association class that defines the semantics of the SUPAPolicySubjects association. This enables the attributes and relationships of the SUPAPolicySubjectDetail class to be used to constrain which SUPAPolicySubject objects can be used to author this particular SUPAPolicy object instance.

5.6.2.2.1. SUPAPolicySubjectDetail Attributes

Currently, one attribute is defined for the SUPAPolicySubjectDetail association class.

5.6.2.2.1.1. The Attribute "supaPolicySubjectIsVerified"

This is an optional Boolean attribute. If the value of this attribute is true, then this SUPAPolicySubject has been verified to be the author of this particular SUPAPolicy object instance.

5.7. The Abstract Class "SUPAPolicyTarget"

SUPAPolicyTarget was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5]. Figure 9 shows a class diagram of the SUPAPolicyTarget.

A PolicyTarget is a set of managed entities that a SUPAPolicy is applied to. This is determined by two conditions.

First, the set of managed entities that are to be affected by the SUPAPolicy must all agree to play the role of a SUPAPolicyTarget. In general, a managed entity may or may not be in a state that enables SUPAPolicies to be applied to it to change its state; hence, a negotiation process may need to occur to enable the SUPAPolicyTarget to signal when it is willing to have SUPAPolicies applied to it.

Second, a SUPAPolicyTarget must be able to either process (either directly or with the aid of a proxy) SUPAPolicies or receive the results of a processed SUPAPolicy and apply those results to itself. If a proposed SUPAPolicyTarget meets both of these conditions, it SHOULD set its supaPolicyTargetEnabled Boolean attribute to a value of TRUE.

A SUPAPolicyTarget SHOULD be mapped to a role (e.g., using the role-object pattern). Note that Role is a type of Metadata.
5.7.1. SUPAPolicyTarget Attributes

There are currently no attributes defined for the SUPAPolicyTarget class.

5.7.2. SUPAPolicyTarget Relationships

This section defines the relationships of the SUPAPolicyTarget class.

5.7.2.1. The Relationship "SUPAPolicyTargets"

This is an optional aggregation that defines the set of SUPAPolicyTargets that are contained in the instance of this particular SUPAPolicy object. This defines the set of entities that will be operated on by this particular SUPAPolicy object. The multiplicity of this relationship is defined as 0..1 on the aggregate (SUPAPolicy) side, and 0..n on the part (SUPAPolicyTarget) side. This means that this relationship is optional, but if it is implemented, then this particular SUPAPolicy object will operate on this set of SUPAPolicyTargets. The semantics of this aggregation are implemented using the HasSUPAPolicyTgtDetail association class.

5.7.2.2. The Association Class "SUPAPolicyTargetDetail"

This is an optional concrete association class that defines the semantics of the SUPAPolicyTargets aggregation. This enables the attributes and relationships of the SUPAPolicyTargetDetail class to be used to constrain which SUPAPolicyTarget objects can be operated on by this particular SUPAPolicy object instance.

5.7.2.2.1. SUPAPolicyTargetDetail Attributes

Currently, one attribute is defined for the SUPAPolicyTargetDetail association class

5.7.2.2.1.1. The Attribute "supaPolicyTargetEnabled"

This is an optional Boolean attribute. If its value is TRUE, then this indicates that this SUPAPolicyTarget is currently able to have SUPAPolicies applied to it. Otherwise, this SUPAPolicyTarget is not able to have SUPAPolicies applied to it.
5.8. The Abstract Class "SUPAPolicyTerm"

This is a mandatory abstract class that is the parent of SUPAPolicy objects that can be used to define a standard way to test or set the value of a variable. It does this by defining a 3-tuple, in the form {variable, operator, value}, where each element of the 3-tuple is defined by a concrete subclass of the appropriate type (i.e., SUPAPolicyVariable, SUPAPolicyOperator, and SUPAPolicyValue, respectively). For example, a generic test of the value of a variable is expressed as {variable, operator, value}.

SUPAPolicyTerm was abstracted from DEN-ng [2].

SUPAPolicyTerm is defined as an abstract class for two reasons:

1. This enables a single aggregation (SUPAPolicyTermsInStmt; see section 5.8.2.1) to be used to specify which object instances of which SUPAPolicyTerm subclasses are contained by a particular SUPAPolicyStatement object instance. Otherwise, a set of three aggregations would be required.

2. This enables a single class (SUPAPolicyTermsInStmtDetail; see section 5.8.2.2) to be used as a superclass to define which one of its subclasses participates in this relationship. The advantage of this design is that as more SUPAPolicyTerm subclasses are added in the future, the SUPAPolicyStatement object is not affected.

This design emphasizes flexibility and genericity of the model. Specifically, this means that the concept of creating a SUPAPolicyStatement can take a generic form, consisting of the tuple (PolicyVariable, PolicyOperator, PolicyValue). Note that this is only one option for constructing SUPAPolicyStatements, and is not mandatory; hence, the multiplicity of the SUPAPolicyTermsInStmt aggregation (see Section 5.8.2.) is 0..n - 0..n.

This design is in marked contrast to most existing designs. For example, [RFC3060], [RFC3460], and [4] do not define an ECA Policy Rule; rather, they are limited to a Policy Rule that only has a condition clause and an action clause. This means that there is no mechanism for the system to trigger when a Policy Rule should be evaluated (because there is no event clause). This makes it very difficult to simulate what will happen when a policy rule is executed. In addition, [RFC3060], [RFC3460], and [4] do not define any type of logic statement (or, for that matter, any other type of Policy Rule).
5.8.1. SUPAPolicyTerm Attributes

Currently, SUPAPolicyTerm defines a single attribute, as described in the following subsection. Constraints on the subclasses of SUPAPolicyTerm are applied to individual classes as attributes, or through the SUPAPolicyTermsInStmtDetail association class (see section 5.8.2.2).

5.8.1.1 The Attribute "supaTermIsNegated"

This is an optional Boolean attribute. If the value of this attribute is true, then this particular SUPAPolicyTerm subclass is negated; otherwise, it is not. This is based on the DEN-ng class design [2].

5.8.2. SUPAPolicyTerm Relationships

Currently, SUPAPolicyTerm participates in a single relationship, as described in the following subsection.

5.8.2.1. The Aggregation "SUPAPolicyTermsInStmt"

This is a mandatory aggregation that defines the set of SUPAPolicyTerms that are contained in this SUPAPolicyStatement. The multiplicity of this relationship is defined as 0..n on the aggregate (SUPAPolicyStatement) side, and 0..n on the part (SUPAPolicyTerm) side. This means that a SUPAPolicyStatement does not have to contain a SUPAPolicyTerm; this is typically true for SUPALogicStatement. However, if a SUPAPolicyStatement does require one or more SUPAPolicyTerms, then those may be defined using this aggregation. The semantics of this aggregation are implemented using the SUPAPolicyTermsInStmtDetail association class.

5.8.2.2. The Association Class "SUPAPolicyTermsInStmtDetail"

This is a mandatory abstract association class that defines the semantics of the SUPAPolicyTermsInStmt aggregation. This enables the attributes and relationships of the SUPAPolicyTermsInStmtDetail class to be used to constrain which SUPAPolicyTerm objects can be aggregated by this particular SUPAPolicyStatement object instance.
The preferred design is to keep this association class abstract, and create three subclasses from it that constrain the set of SUPAPolicyVariables, SUPAPolicyOperators, and SUPAPolicyValues that are used with this particular SUPAPolicyStatement. This provides a direct and simple mapping to optimized data models. Alternatively, appropriate attributes could be added to this association class to define the constraint, but such attributes would also have to take into account the type of PolicyTerm subclass that is being constrained.

5.8.2.2.1. SUPAPolicyTermsInStmtDetail Attributes

Currently, SUPAPolicyTermsInStmtDetail defines a single generic attribute, as described below.

5.8.2.2.1.1. The Attribute "supaPolTermOCLConstraint"

This is an optional string attribute that specifies a constraint to be applied. The constraint is defined in OCL 2.0. It is independent of the type of SUPAPolicyTerm that it applies to.

5.8.3. SUPAPolicyTerm Subclasses

The following three subsections define three subclasses of the SUPAPolicyTerm class.

5.8.3.1. The Concrete Class "SUPAPolicyVariable"

This is a mandatory concrete class that defines information that forms a part of a SUPAPolicyStatement. It specifies a concept or attribute that should be compared to a value, as specified in this SUPAPolicyStatement. If it is used in a SUPAECAPolicyRule, then its value MAY be able to be changed at any time. However, if it is used in a SUPALogicStatement, then it is typically bound to an expression, and keeps a single value during its entire lifetime. SUPAPolicyVariable was abstracted from DEN-ng [2].

The value of a SUPAPolicyVariable is typically compared to the value of a SUPAPolicyValue using the type of operator defined in a SUPAPolicyOperator.

SUPAPolicyVariables are used to abstract the representation of a SUPAPolicyRule from its implementation. Therefore, the design of SUPAPolicyVariables depends on two important factors.
First, some SUPAPolicyVariables are restricted in the values and/or the data type that they may be assigned. For example, port numbers cannot be negative, and they cannot be floating-point numbers. Thus, any SUPAPolicyVariable can have a set of constraints associated with it that restrict the value, data type, and other semantics of the SUPAPolicyVariable when used in a particular SUPAPolicyStatement. Second, there is a high likelihood that specific applications will need to use their own variables that have specific meaning to a particular application.

In general, there are two ways to apply constraints to an object instance of a SUPAPolicyVariable:

- use a specific subclass of PolicyVariable, which has these constraints already applied to the attribute to be used
- define constraints in the supaPolTermConstraints attribute of the SUPAPolicyTermsInStmtDetail association class (see Section 5.8.2.2.2.1.)

The former approach was used in [RFC3460]. The problem with this approach is that it requires two additional subclasses (called PolicyImplicitVariable and PolicyExplicitVariable) with two different semantics, as well as three different associations. It also leads to class explosion. This draft keeps the idea of the class hierarchy for backwards compatibility, but streamlines the implementation.

The latter approach is recommended, due to the use of established software patterns that can be used to populate the attribute(s) of the SUPAPolicyTermsInStmtDetail association class, or any subclass that is defined to refine its semantics.

5.8.3.1.1. Problems with the RFC3460 Version of PolicyVariable

First, [RFC3460] says: "Variables are used for building individual conditions". While this is true, variables can also be used for building individual actions. This is reflected in the SUPAPolicyVariable definition.

Second, [RFC3460] says: "The variable specifies the property of a flow or an event that should be matched when evaluating the condition." While this is true, variables can be used to test many other things than "just" a flow or an event. This is reflected in the SUPAPolicyVariable definition.

Third, the [RFC3460] definition requires the use of associations in order to properly constrain the variable (e.g., define its data type, the range of its allowed values, etc.). This is both costly and inefficient.
Fourth, in [RFC3460], defining constraints for a variable is limited to associating the variable with a PolicyValue. This is both cumbersome (because associations are costly; for example, they equate to a join in a relational database management system), and not scalable, because it is prone to proliferating PolicyValue classes for every constraint (or range of constraints) that is possible. Therefore, in SUPA, this mechanism is replaced with using an association to a generic SUPAConstraint object.

Fifth, [RFC3460] is tightly bound to the DMTF CIM schema [4]. The CIM is a data model (despite its name), because:

- It uses keys and weak relationships, which are both concepts from relational algebra and thus, not technology-independent
- It has its own proprietary modeling language
- It contains a number of concepts that are not defined in UML (including overriding keys for subclasses)

Finally, the class hierarchy has two needless classes, called SUPAImplicitVariable and SUPAExplicitVariable. These classes do not define any attributes or relationships, and hence, do not add any semantics to the model.

5.8.3.1.2. SUPAPolicyVariable Attributes

Currently, SUPAPolicyVariable defines three generic attributes, as described below.

5.8.3.1.2.1. The Attribute "$supaAllowedValueTypes[0..n]"

This is an optional array of string attributes. Each attribute specifies a constraint to be applied using OCL 2.0. This provides a more rigorous and flexible treatment of constraints than is possible in [RFC3460].

5.8.3.1.2.2. The Attribute "$supaPolVarContent"

This is a string attribute that contains the value of the SUPAPolicyVariable object instance. Its data type is defined by the supaPolVarType class attribute.

5.8.3.1.2.3. The Attribute "$supaPolVarType"

This is a string attribute that contains the data type of the SUPAPolicyVariable object instance. Its value is defined by the supaPolVarContent class attribute.
5.8.3.1.3. SUPAPolicyVariable Subclasses

A set of SUPAPolicyVariable subclasses will be defined in the next version of this document. These are included for backwards compatibility with existing designs based on [RFC3460]. This is a more complex approach, and is not recommended.

5.8.3.2. The Concrete Class "SUPAPolicyOperator"

This is a mandatory concrete class for modeling different types of operators that are used in a SUPAPolicyStatement.

A SUPAPolicyOperator is a mandatory concrete class that defines the type of operator to be applied to a SUPAPolicyStatement that is created from a set of SUPAPolicyTerms. This class is based on a similar class defined in [2].

The restriction of the type of operator used in a SUPAPolicyStatement restricts the semantics that can be expressed in that SUPAPolicyStatement.

5.8.3.2.1. Problems with the RFC3460 Version

Note that this class is NOT present in either RFC[3060] or [RFC3460]; instead, both hardwire the operator to a "MATCH" function. Quoting from [RFC3460]:

"A simple condition models an elementary Boolean expression of the form 'variable MATCHes value". However, the formal notation of the SimplePolicyCondition, together with its associations, models only a pair, (<variable>, <value>). The 'MATCH' operator is not directly modeled -- it is implied. Furthermore, this implied 'MATCH' operator carries overloaded semantics [sic]."

In stark contrast to this, SUPA defines a SUPAPolicyOperator as a formal subclass of SUPAPolicyTerm. A single attribute, called supaPolOpType, carries the operator to be applied to the SUPAECAPolicyRule. This has the important advantage of enabling ECA policy rules of varying functionality to be created by a human or a machine.

5.8.3.2.2. SUPAPolicyOperator Attributes

Currently, SUPAPolicyOperator defines a single generic attribute, as described below.
5.8.3.2.2.1. The Attribute "supaPolOpType"

This is a mandatory non-negative enumerated integer that specifies the various types of operators that are allowed to be used in this particular SUPAPolicyStatement. Values include:

0: Unknown
1: Match
2: Greater than
3: Greater than or equal to
4: Less than
5: Less than or equal to
6: Equal to
7: Not equal to
8: IN
9: NOT IN
10: SET
11: CLEAR

Note that 0 is an unacceptable value. Its purpose is to support dynamically building a SUPAPolicyStatement by enabling the application to set the value of this attribute to a standard default value if the real value is not yet known.

5.8.3.3. The Concrete Class "SUPAPolicyValue"

The SUPAPolicyValue class is a mandatory concrete class for modeling different types of values and constants that occur in a PolicyStatement. SUPAPolicyValue was abstracted from DEN-ng [2].

The value of a SUPAPolicyVariable is typically compared to the value of a SUPAPolicyValue using the type of operator defined in a SUPAPolicyOperator.

SUPAPolicyValues are used to abstract the representation of a SUPAPolicyRule from its implementation. Therefore, the design of SUPAPolicyValues depends on two important factors. First, just as with SUPAPolicyVariables (see Section 5.8.3.1), some types of SUPAPolicyValues are restricted in the values and/or the data type that they may be assigned. Second, there is a high likelihood that specific applications will need to use their own variables that have specific meaning to a particular application.

In general, there are two ways to apply constraints to an object instance of a SUPAPolicyValue:
use a specific subclass of PolicyValue, which has these
constraints already applied to the attribute to be used

- define constraints in the supaPolTermConstraints attribute
  of the SUPAPolicyTermsInStmtDetail association class (see
  Section 5.8.2.2.2.1.)

The former approach was used in [RFC3460]. The problem with this
approach is that it generates a set of classes, each having a
single data type, to represent a primitive type. Hence, this
approach may lead to class explosion. This draft keeps the idea
of the class hierarchy for backwards compatibility, but
streamlines the implementation.

The latter approach is recommended, due to the use of established
software patterns that can be used to populate the attribute(s)
of the SUPAPolicyTermsInStmtDetail association class, or any
subclass that is defined to refine its semantics.

5.8.3.3.1. Problems with the RFC3460 Version of PolicyValue

First, [RFC3460] says: It is used for defining values and
constants used in policy conditions”. While this is true,
variables can also be used for building individual actions. This
is reflected in the SUPAPolicyVariable definition.

Second, the [RFC3460] definition requires the use of associations
in order to properly constrain the variable (e.g., define its
data type, the range of its allowed values, etc.). This is both
costly and inefficient.

Third, in [RFC3460], there is no generic facility for defining
constraints for a PolicyValue.

Fourth, [RFC3460] is tightly bound to the DMTF CIM schema [4].
The CIM is a data model (despite its name), because:

- It uses keys and weak relationships, which are both concepts
  from relational algebra and thus, not technology-independent
- It has its own proprietary modeling language
- It contains a number of concepts that are not defined in UML
  (including overriding keys for subclasses)

5.8.3.3.2. SUPAPolicyValue Attributes

Currently, SUPAPolicyValue defines two generic attributes, as
described below.
5.8.3.3.2.1. The Attribute "supaPolValContent"

This is a string attribute that contains the value of the SUPAPolicyValue object instance. Its data type is defined by the supaPolValType class attribute.

5.8.3.3.2.2. The Attribute "supaPolValType"

This is a string attribute that contains the data type of the SUPAPolicyValue object instance. Its value is defined by the supaPolValContent class attribute.

5.8.3.3.3. SUPAPolicyValue Subclasses

A set of SUPAPolicyValue subclasses will be defined in the next version of this document. These are included for backwards compatibility with existing designs based on [RFC3460]. This is a more complex approach, and is not recommended.

5.9. The Abstract Class "SUPAPolicyMetadata"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

SUPAPolicyMetadata was abstracted from DEN-ng [2]. A more complete representation of metadata, as defined in [2], is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

5.9.1. SUPAPolicyMetadata Attributes

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

5.9.2. SUPAPolicyMetadata Relationships

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.
6. SUPA ECAPolicyRule Information Model

This section defines the classes, attributes, and relationships of the SUPA ECAPolicyRule Information Model (EPRIM).

6.1. Overview

Conceptually, the EPRIM is a set of subclasses that specialize the concepts defined in the GPIM for representing the components of a Policy that uses ECA semantics. This is shown in Figure 10.

(Class of another model that SUPA is integrating into)

---SUPAPolicy (see Section 5.2)
  |---SUPAPolicyAtomic (see Section 5.3)
  |    |---SUPAECAPolicyRule (see Section 6.4)
  |    |---SUPAECAComponent (see Section 6.6)
  |    |    |---SUPAECAEvent (see Section 6.6.3.1)
  |    |    |---SUPACondition (see Section 6.6.3.2)
  |    |    |---SUPAAction (see Section 6.6.3.3)
  |---SUPAPolicyComposite (see Section 5.4)
  |---SUPAPolicyStatement (see Sections 5.5 and 6.5)
  |    |---SUPAEncodedClause (see Section 5.5.2.1)
  |    |---SUPABooleanClause (see Section 6.5.2)
  |---SUPAPolicySubject (see Section 5.6)
  |---SUPAPolicyTarget (see Section 5.7)
  |---SUPAPolicyTerm (see Section 5.8)
  |---SUPAPolicyMetadata (see Section 5.9)
...

Figure 10: The EPRIM Refining the GPIM
Specifically, the EPRIM specializes the SUPAPolicyAtomic class to create a SUPAECAPolicyRule; it also specializes the SUPAPolicy class to create a SUPAECAComponent, and the SUPAPolicyStatement to create a SUPABooleanClause. The SUPAECAPolicyRule uses the rest of the GPIM infrastructure to define a complete Policy model according to ECA semantics.

The overall strategy for refining the GPIM is as follows:

- SUPAECAPolicyRule is defined as a subclass of the GPIM SUPAPolicyAtomic class
- A SUPAECAPolicyRule has event, condition, and action clauses; each of these are created by either a SUPABooleanClause or a SUPAEncodedClause (subclasses of SUPAPolicyStatement)
- Both a SUPABooleanClause and a SUPAEncodedClause inherit the HasSUPAECAComponents aggregation, so both of these types of clauses can use SUPAECAComponents in their construction
- A SUPAECAComponent defines SUPAEvent, SUPACondition, and SUPAAction objects that can optionally be used to create the event, condition, and action clauses of a SUPAECAPolicyRule
- Both a SUPABooleanClause and a SUPAEncodedClause inherit the SUPAPolicyTermsInStmt aggregation, so both of these types of clauses can use SUPAPolicyTerms in their construction
- An optional set of GPIM SUPAPolicySubjects can be defined to represent the authoring of a SUPAECAPolicyRule
- An optional set of GPIM SUPAPolicyTargets can be defined to represent the set of managed entities that will be affected by this SUPAECAPolicyRule
- An optional set of SUPAPolicyMetadata can be defined for any of the objects that make up a SUPAECAPolicyRule and/or a SUPAECAComponent

6.2. Constructing a SUPAECAPolicyRule

There are several different ways to construct a SUPAECAPolicyRule. The simplest approach is as follows:

- Define three types of SUPABooleanClauses (see Section 6.7), one each for the event, condition, and action clauses that make up a SUPAECAPolicyRule (see Section 6.4)
- Define a set of SUPAEvent, SUPACondition, and SUPAAction objects (see Section 6.5.1, 6.5.2, and 6.5.4, respectively), and associate each with the SUPABooleanClause that represents the event, condition, and action clauses, respectively, of the SUPAECAPolicyRule
- Define a SUPAECAPolicyRule, which is a subclass of the GPIM SUPAPolicyAtomic class (see Section 5.3)
o Aggregate the three SUPABooleanClauses into the SUPAECAPolicyRule
o Optionally, define a set of SUPAPolicySubjects and SUPAPolicyTargets, and aggregate them into the SUPAECAPolicyRule
o Optionally, define SUPAPolicyMetadata for any of the above objects, and aggregate them to the SUPAPolicy objects that the SUPAPolicyMetadata applies to

6.3. Working With SUPAECAPolicyRules

A SUPAECAPolicyRule is a type of SUPAPolicy. It is a tuple that MUST have three clauses, defined as follows:

- The event clause defines a Boolean expression that, if TRUE, triggers the evaluation of its condition clause (if the event clause is not TRUE, then no further action for this policy rule takes place).
- The condition clause defines a Boolean expression that, if TRUE, enables the actions in the action clause to be executed (if the condition clause is not TRUE, then no further action for this policy rule takes place).
- The action clause is a set of actions, whose execution MAY be controlled by the SUPAPolicyMetadata of the policy rule.

Note that in theory, the condition clause of an ECA policy rule could also include multiple clauses (like, for example, a switch statement). Similarly, multiple action clauses (i.e., one for each distinct condition clause) could be included as well. This was done on a limited basis in DEN-ng. However, this complicates the overall design, so at this time, SUPA is not providing this facility.

Each of the three clauses can be constructed from either a SUPAEncodedClause or a SUPABooleanClause. The advantage of using SUPAEncodedClauses is simplicity, as the content of the clause is encoded directly into the attributes of the SUPAEncodedClause. The advantage of using SUPABooleanClauses is reusability, since each term in each clause is potentially a reusable object.

Since a SUPABooleanClause is a subclass of a SUPAPolicyStatement (see Section 5.5), it can aggregate SUPAPolicyTerm objects as well as SUPAECAClause objects. Therefore, a SUPAECAPolicyRule can be built entirely from objects defined in the GPIM. As will be shown in Section 7.4, this is also true for SUPALogicStatements.
The construction of a SUPAECAPolicyRule is shown in Figure 11, and is explained in Section 6.4.

Figure 11. SUPAECAPolicyRule Clauses

NOTE: This is a simplified design, inspired from [2]. The HasSUPAECACOMPONENTS aggregation is implemented using the HasSUPAECACOMPONENTDETAIL association class. This is an abstract class further described in Section 6.4.2. It has three concrete subclasses, one each that correspond to the three subclasses of SUPAECACOMPONENT (i.e., SUPAEVENT, SUPACONDITION, and SUPAACTION), which are all concrete. This enables one aggregation to define a set of constraints between a SUPAPOLICYSTATEMENT and the set of Events, Conditions, and/or Actions that it can contain.

6.4. The Concrete Class "SUPAECAPolicyRule"

This is a concrete mandatory class. In keeping with the original DEN-ng model [1], this class is a PolicyContainer that contains PolicyEvents, PolicyConditions, PolicyActions, and optionally, PolicySubjects, PolicyTargets, and PolicyMetadata. As such, it does not have an inherent relationship with PolicySubject or PolicyTarget; these all represent the specific semantics for a particular SUPAPolicy. Hence, such semantics are defined in an instance of the SUPAPOLICYComposite class that contains a SUPAECAPolicyRule, if they are required.
An Event-Condition-Action (ECA) Policy (SUPAECAPolicyRule) is an abstract class that represents a policy rule as a three-tuple, consisting of at least one event clause, one condition clause, and one action clause. Each clause MUST be defined by at least one SUPAPolicyStatement. Optionally, the SUPAECAPolicyRule MAY contain one or more SUPAPolicySubjects, one or more SUPAPolicyTargets, and one or more SUPAPolicyMetadata objects.

There are two main ways that the event, condition, and action clauses of a SUPAECAPolicyRule can be populated:

- Use a SUPAEncodedClause
- Use a SUPABooleanClause using SUPAECAComponents and/or SUPAPolicyTerms

Regardless of which approach is taken, the operation of a SUPAECAPolicyRule is identical:

- the event clause specifies what triggers the evaluation of the SUPAECAPolicyRule
- the condition clause specifies whether the condition clause has evaluated to TRUE or FALSE, and hence, whether the action clause should be executed or not
- the action clause specifies the set of actions to be executed

If there is more than one term, then these terms MUST be combined using any combination of logical AND, OR, and NOT operators to form a Boolean clause (i.e., a clause whose value is either TRUE or FALSE). For example, a valid event clause could be: "three events of type A AND NOT an event of type B".

This behavior differentiates a SUPAECAPolicyRule from a SUPALogicStatement, which specifies the set of actions to perform, but not how to implement or execute them.

The behavior of the event, condition, and action clauses MAY be specified using one or more SUPAMetadata objects that have been aggregated by the SUPAECAPolicyRule. Note that one or more SUPAMetadata objects may also be aggregated by any of the components of a SUPAECAPolicyRule.

A SUPAECAPolicyRule MAY specify a set of SUPAPolicySubjects that have authored the SUPAECAPolicyRule. A SUPAECAPolicyRule MAY specify a set of SUPAPolicyTargets that define a set of managed objects that the actions of the SUPAECAPolicyRule MAY monitor and/or change their state.
When defined in an information model, each of the event, condition, and action clauses MUST be represented as an aggregation between a SUPAECAPolicyRule (the aggregate) and a set of event, condition, or action objects (the components). However, a data model MAY map these definitions to a more efficient form (e.g., by flattening these three types of object instances, along with their respective aggregations, into a single object instance).

The semantics of a SUPAECAvPolicyRule may be conceptualized as follows:

ON RECEIPT OF <policy-event-clause>
  IF <policy-condition-clause> EVALUATES TO TRUE
  THEN EXECUTE <policy-action-clause>
END

In the above, a policy-event-clause, policy-condition-clause, and a policy-action-clause are each instances of either a SUPAEncodedClause or a SUPABooleanClause.

SUPAECAvPolicyRule was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

6.4.1. SUPAECAvPolicyRule Attributes

Currently, the SUPAECAvPolicyRule defines three attributes, as described in the following subsections.

6.4.1.1. The Attribute "supaECAPolicyIsMandatory"

This is an optional Boolean attribute. If the value of this attribute is true, then this SUPAECAvPolicyRule MUST be executed (i.e., its Event and Condition clauses are irrelevant, and the Action(s) specified in the Action clause MUST be executed). These actions will use the inherited supaPolicyExecStrategy attribute to govern which of the Actions in this SUPAECAvPolicyRule will be executed or not.

6.4.1.2. The Attribute "supaECAPolicyPriority"

This is a mandatory non-negative integer attribute that defines the priority of this particular SUPAECAvPolicyRule. A larger value indicates a higher priority. A default value of 0 MAY be assigned.
6.4.2. SUPAECAPolicyRule Relationships

Currently, the SUPAECAPolicyRule defines a single aggregation between it and SUPAECAComponent, as described below. Note that the remaining functionality that SUPAECAPolicyRule requires is provided by other relationships within the GPIM and EPRIM. For example, the aggregation SUPAHasPolicyStmts (see section 5.3.2.1) defines the set of SUPAPolicyStatements that MUST be aggregated by this particular SUPAPolicyAtomic object.

6.4.2.1. The Aggregation "SUPAECAPolicyRules"

This is an optional aggregation that defines the set of SUPAEEvents, SUPAConditions, and SUPAActions that are aggregated by this particular SUPAECAPolicyRule. The multiplicity of this aggregation is 0..1 on the aggregate (SUPAECAPolicyRule) side and 1..n on the part (SUPAECAComponent) side. This means that if this aggregation is defined, then at least one SUPAECAComponent must also be instantiated and aggregated by the SUPAECAPolicyRule. However, a SUPAECAComponent does not have to be instantiated when a SUPAECAComponent is instantiated; this enables SUPAECAComponent objects to be stored in a repository, for use as a library. The semantics of this aggregation are defined by the SUPAECADetail association class.

6.4.2.1.1. The Association Class "HasSUPAECAComponentDetail"

This is an optional association class, and defines the semantics of the HasSUPAECAComponent aggregation. This enables the attributes and relationships of the SUPAPAHasPolicyStmtDetail class to be used to constrain which SUPAPolicyStatements objects can be aggregated by this particular SUPAPolicyAtomic object instance. Attributes will be added to this class at a later time.

6.4.3. SUPAECAPolicyRule Subclasses

The composite pattern [3] is applied to the SUPAECAPolicyRule class, enabling it to be used as either a stand-alone policy rule or as a hierarchy of policy rules. This is shown in Figure 12.

SUPAECAPolicyRuleComposite and SUPAECAPolicyRuleAtomic both inherit from SUPAECAPolicyRule. This means that they are both a type of SUPAECAPolicyRule. Hence, the HasSUPAECAPolicyRules aggregation enables a particular SUPAECAPolicyRuleComposite object to aggregate both SUPAECAPolicyRuleComposite as well as SUPAECAPolicyRuleAtomic objects. In contrast, a SUPAECAPolicyRuleAtomic can NOT aggregate either a SUPAECAPolicyRuleComposite or a SUPAECAPolicyRuleAtomic.
6.4.3.1. The Concrete Class "SUPAECAPolicyRuleAtomic"

This is a mandatory concrete class. This class is a type of PolicyContainer. SUPAECAPolicyRuleAtomic was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

A SUPAECAPolicyRuleAtomic class represents a SUPA ECA Policy Rule that can operate as a single, stand-alone, manageable object. Put another way, a SUPAECAPolicyRuleAtomic object can NOT be modeled as a set of hierarchical SUPAECAPolicyRule objects; if this is required, then a SUPAECAPolicyRuleComposite object should be used instead.

6.4.3.1.1. SUPAECAPolicyRuleAtomic Attributes

No attributes are currently defined for the SUPAECAPolicyRule class.

6.4.3.1.2. SUPAECAPolicyRuleAtomic Relationships

No relationships are currently defined for the SUPAECAPolicyRule class.
6.4.3.2. The Concrete Class "SUPAECAPolicyRuleComposite"

This is a mandatory concrete class. This class is a type of PolicyContainer. SUPAECAPolicyRuleComposite was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

A SUPAECAPolicyRuleComposite class represents a SUPA ECA Policy Rule as a hierarchy of Policy objects, where the hierarchy contains instances of a SUPAECAPolicyRuleAtomic and/or SUPAECAPolicyRuleComposite object. Each of the SUPA Policy objects, including the outermost SUPAECAPolicyRuleComposite object, are separately manageable. More importantly, the SUPAECAPolicyRuleComposite object can aggregate any SUPAECAPolicyRule subclass. Hence, it can be used to form hierarchies of SUPAECAPolicyRules as well as associate SUPAPolicySubjects and/or SUPAPolicyTargets to a given SUPAECAPolicyRule.

6.4.3.2.1. SUPAECAPolicyRuleAtomic Attributes

Currently, the SUPAECAPolicyRule defines two attributes, as described in the following subsections.

6.4.3.2.1.1. The Attribute "supaECAEvalStrategy"

This is a mandatory, non-zero, integer attribute that enumerates a set of allowable alternatives that define how the actions in a SUPAPolicyRuleComposite are evaluated. Values include:

0: undefined
1: execute all actions regardless of their execution status
2: execute all actions until one or more actions fail
3: execute only the highest priority action(s)

6.4.3.2.1.2. The Attribute "supaECAFailStrategy"

This is a mandatory, non-zero, integer attribute that enumerates a set of allowable alternatives that define how actions that do not execute successfully should be handled. Values include:

0: undefined
1: rollback just the failed action
2: rollback all actions that have been executed in this SUPAECAPolicyRule
3: ignore failures and continue
6.4.3.2.2. SUPAECAPolicyRuleComposite Relationships

Currently, a single aggregation is defined for the
SUPAECAPolicyRuleComposite class, which is defined below.

6.4.3.2.2.1. The Aggregation "SUPAECAPolicyRules"

This is a mandatory aggregation that defines the set of
SUPAECAPolicyRule objects (i.e., instances of either the
SUPAECAPolicyRuleAtomic or SUPAECAPolicyRuleComposite classes)
that are contained in this particular SUPAECAPolicyRuleComposite
object instance. The semantics of this aggregation are defined in
the SUPAECAPolicyRuleDetail association class.

6.4.3.2.2.2. The Association Class "SUPAECAPolicyRuleDetail"

This is a mandatory association class, and defines the semantics
of the SUPAECAPolicyRules aggregation. This enables the
attributes and relationships of the SUPAECAPolicyRuleDetail
class to be used to constrain which SUPAECAPolicyRule objects
can be aggregated by this particular SUPAPolicyComposite object
instance. Attributes will be added to this class at a later time.

6.5. SUPAPolicyStatement Subclasses

Section 5.5.2 defines a common subclass of SUPAPolicyStatement,
called SUPAEncodedClause, which any SUPAPolicy (rule or predicate)
can use. This section describes another specialization of the
GPIM SUPAPolicyStatement class for use in constructing (only)
SUPAECAPolicyRule objects.

The SUPAPolicyStatement class, and its subclasses, are based on
similar classes in [2].

6.5.1. Designing SUPAPolicyStatements Using SUPABooleanClauses

A SUPABooleanClause specializes a SUPAPolicyClause, and defines a
Boolean statement consisting of a standard structure in the form
of a PolicyVariable, a PolicyOperator, and a PolicyValue. This
design is based on the DEN-ng model [2]. For example, this enables
the following Boolean clause to be defined:

    Foo >= Bar AND Baz
where Foo is a PolicyVariable, \( \geq \) is a PolicyOperator, and Bar is a PolicyValue. Note that in this approach, each of these three terms (i.e., the PolicyVariable, PolicyOperator, and PolicyValue) are subclasses of the SUPAPolicyTerm class, which is defined in Section 5.8). This enables the EPRIM, in conjunction with the GPIM, to be used as a reusable class library. This encourages interoperability, since each element of the clause is itself an object defined by SUPA.

The addition of a negation in the above statement is provided by the supaTermIsNegated Boolean attribute in the SUPAPolicyTerm class. An entire clause is indicated as negated using the supaBoolIsNegated Boolean attribute in the SUPABooleanClause class.

A PolicyStatement is in Conjunctive Normal Form (CNF) if it is a conjunction (i.e., a sequence of ANDed terms), where each term is a disjunction (i.e., a sequence of ORed terms). Every statement that consists of a combination of AND, OR, and NOT operators can be written in CNF.

A PolicyStatement is in Disjunctive Normal Form (DNF) if it is a disjunction (i.e., a sequence of ORed terms), where each term is a conjunction (i.e., a sequence of ANDed terms). Every statement that consists of a combination of AND, OR, and NOT operators can be written in DNF.

The supaBoolISCNF Boolean attribute of the SUPABooleanClause class is TRUE if this SUPABooleanClause is in CNF, and FALSE otherwise.

The construction of more complex clauses, which consist of a set of simple clauses in conjunctive or disjunctive normal form (as shown in the above example), is provided by using the composite pattern [3] to construct two subclasses of SUPABooleanClause. These are called SUPABooleanClauseAtomic and SUPABooleanClauseComposite, and are defined in Sections 6.5.2.1 and 6.5.2.2, respectively. This enables instances of either a SUPABooleanClauseAtomic and/or a SUPABooleanClauseComposite to be aggregated into a SUPABooleanClauseComposite object.

6.5.2. The Abstract Class"SUPABooleanClause"

This is a mandatory abstract class that defines a clause as the following three-tuple:

\[
\text{(PolicyVariable, PolicyOperator, PolicyValue)}
\]
The composite pattern [3] is used in order to construct complex Boolean clauses from a set of SUPABooleanClause objects. This is why SUPABooleanClause is defined to be abstract - only instances of the SUPABooleanAtomic and/or SUPABooleanComposite classes can be used to construct a SUPABooleanClause.

SUPAECAPolicyRuleComposite and SUPAECAPolicyRuleAtomic both inherit from SUPAECAPolicyRule. This means that they are both a type of SUPAECAPolicyRule. Hence, the HasSUPAECAPolicyRules aggregation enables a particular SUPAECAPolicyRuleComposite object to aggregate both SUPAECAPolicyRuleComposite as well as SUPAECAPolicyRuleAtomic objects. In contrast, a SUPAECAPolicyRuleAtomic can NOT aggregate either a SUPAECAPolicyRuleComposite or a SUPAECAPolicyRuleAtomic.

The advantage of a SUPABooleanClause is that it is formed entirely from SUPAPolicy objects. This enhances both reusability as well as interoperability. Since this involves compositing a number of objects, data model implementations MAY optimize a SUPABooleanClause according to their application-specific needs (e.g., by flattening the set of classes that make up a SUPABooleanClause object into a single object).

Figure 13 below shows the composite pattern applied to the SUPABooleanClause class.

```
  1..n +-------------------+
     \                   |
      +--------------- + SUPABooleanClause |
          \               /                     |
           |             +--------+----------+
           |                        / \
           | HasSUPABooleanClauses   I \
           |                         I
           |                         I
           +--------------------------+
     0..1 \ /       I             I
      +-------+--------+---------+    +-----------+-----------+
      |SUPABooleanClauseComposite|    |SUPABooleanClauseAtomic|
      +--------------------------+    +-----------------------+
```

Figure 13. The Composite Pattern Applied to a SUPABooleanClause

6.5.2.1. SUPABooleanClause Attributes

The following sections define attributes of a SUPABooleanClause.
6.5.2.1.1. The Attribute "supaBoolIsNegated"

This is a mandatory Boolean attribute. If the value of this attribute is TRUE, then this SUPABooleanClause is negated.

6.5.2.1.2. The Attribute "supaPolStmtBindValue"

This is an optional non-zero integer attribute, and defines the order in which terms bind to a clause. For example, the Boolean statement "((A AND B) OR (C AND NOT (D or E))) has the following binding order: terms A and B have a bind value of 1; term C has a binding value of 2, and terms D and E have a binding value of 3.

6.5.2.2. SUPABooleanClause Relationships

The following subsections define the relationships of a SUPABooleanClause.

6.5.2.2.1. The Relationship "SUPABooleanClauses"

This is a mandatory aggregation that defines the set of SUPABooleanClauses that are aggregated by this SUPABooleanClauseComposite. This will either form a complete SUPABooleanClause from multiple clauses (which can be made up of SUPABooleanClauseAtomic and/or SUPABooleanClauseComposite object instances) or define another level in the SUPABooleanClause object hierarchy.

The multiplicity of this relationship is 0..1 on the aggregate (SUPABooleanClauseComposite) side, and 1..n on the part (SUPABooleanClause) side. This means that one or more SUPABooleanClauses are aggregated and used to define this SUPABooleanClauseComposite object. The 0..1 cardinality on the SUPABooleanClauseComposite side is necessary to enable SUPABooleanClauses to exist (e.g., in a PolicyRepository) before they are used by a SUPABooleanClauseComposite. The semantics of this aggregation is defined by the SUPABooleanClauseDetail association class.

6.5.2.2.1.1. The Association Class "SUPABooleanClauseDetail"

This is a mandatory association class that defines the semantics of the SUPABooleanClauses aggregation. This enables the attributes and relationships of the SUPABooleanClauseDetail class to be used to constrain which SUPABooleanClause objects can be aggregated by this particular SUPABooleanClauseComposite object instance. Attributes will be added to this class at a later time.
6.5.3. SUPABooleanClause Subclasses

SUPABooleanClause defines two subclasses, as shown in Figure 13. They are both described in the following subsections.

6.5.3.1. The Abstract Class "SUPABooleanClauseAtomic"

This is a mandatory concrete class that represents a SUPABooleanClause that can operate as a single, stand-alone, manageable object. Put another way, a SUPABooleanClauseAtomic object can NOT be modeled as a set of hierarchical clauses; if this functionality is required, then a SUPABooleanClauseComposite object must be used.

SUPAPolicyAtomic was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

No attributes are currently defined for the SUPABooleanClauseAtomic class. Its primary purpose is to aggregate SUPAPolicyVariable, SUPAPolicyOperator, and SUPAPolicyValue objects to form a complete SUPABoolean clause. As such, this class is defined as abstract to simplify data model optimization and mapping.

6.5.3.2. The Abstract Class "SUPABooleanClauseComposite"

This is a mandatory concrete class that represents a SUPABooleanClause that can operate as a hierarchy of PolicyClause objects, where the hierarchy contains instances of a SUPABooleanClauseAtomic and/or SUPABooleanClauseComposite object. Each of the SUPABooleanClauseAtomic and SUPABooleanClauseComposite objects, including the outermost SUPABooleanClauseComposite object, are separately manageable. More importantly, the SUPABooleanClauseComposite object can aggregate any SUPABooleanClause subclass. Hence, it can be used to form hierarchies of SUPABooleanClauses.

SUPABooleanClauseComposite was abstracted from DEN-ng [2], and a version of this class is in the process of being added to the policy framework defined in the TM Forum ZOOM model [5].

6.5.3.2.1. SUPABooleanClauseComposite Attributes

A single attributes is currently defined for the SUPABooleanClauseComposite class, and is described in the following subsection.
6.5.3.2.1.1. The Attribute "supaBoolClauseIsCNF"

This is a mandatory Boolean attribute. If its value is true, then this SUPABooleanClauseComposite is defined in Conjunctive Normal Form (i.e., AND of ORs). Otherwise, it is defined in Disjunctive Normal Form (i.e., OR of ANDs).

6.6. The Abstract Class "SUPAECAComponent"

This is a mandatory abstract class that defines three subclasses, one each for the event, condition, and action portions of a SUPAECAPolicyRule. They are called SUPAEvent, SUPACondition, and SUPAAction, respectively. Its primary purpose is to serve as a convenient aggregation point, and thus has two uses:

1. This enables a single aggregation (SUPAECACOMPONENTSIN TERMS, see section 6.6.2.1) to be used to specify which object instances of which SUPAPolicyTerm subclasses are contained by a particular SUPAECACOMPONENT object instance. Otherwise, a set of three aggregations would be required.
2. This enables a single class (SUPAECACOMPONENTSIN TERMSDETAIL, see section 6.6.2.1.1) to be used as a superclass to define which the specific semantics required by this combination of SUPAPolicyTerm and SUPAECACOMPONENT subclasses.

6.6.1. SUPAECACOMPONENT Attributes

No attributes are currently defined for this class.

6.6.2. SUPAECACOMPONENT Relationships

A single aggregation is defined for this class, as specified below.

6.6.2.1. The Aggregation "SUPAECACOMPONENTSIN TERMS"

This is a mandatory aggregation that defines the set of SUPAPolicyTerms that are aggregated by this SUPAECACOMPONENT. This enables complex combinations of SUPAPolicyTerms to be defined as SUPAEvents, SUPAConditions, or SUPAActions, which increases reusability. It also simplifies forming more complex Boolean clauses as combinations of SUPAEvents, SUPAConditions, or SUPAActions.
The multiplicity of this relationship is 0..1 on the aggregate (SUPAECAComponent) side, and 0..n on the part (SUPAPolicyTerm) side. This means that this aggregation is completely optional. However, if it is instantiated, then zero or more SUPAPolicyTerms can be aggregated by a particular SUPAECAComponent. Similarly, a given SUPAPolicyTerm can be used by zero or more SUPAECAComponents. The 0..1 cardinality on the SUPAECAComponent side is necessary to enable SUPAPolicyTerm objects to exist (e.g., in a PolicyRepository) before they are used by a SUPAECAComponent. The semantics of this aggregation is defined by the SUPAECAComponentHasTermDetail association class.

6.6.2.1.1. The Association Class "SUPAECAComponentsInTermDetail"

This is a mandatory association class that defines the semantics of the SUPAECAComponentsInTerms aggregation. This enables the attributes and relationships of the SUPAECAComponentsInTermDetail class to be used to constrain which SUPAPolicyTerm objects can be aggregated by this particular SUPAECAComponent object instance. Attributes will be added to this class at a later time.

6.6.3. SUPAECAComponent Subclasses

There are three concrete subclasses of SUPAECAComponent; they are described in the following subsections.

6.6.3.1. The Concrete Class "SUPAEEvent"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6.6.3.2. The Concrete Class "SUPACondition"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.

6.6.3.3. The Concrete Class "SUPAAction"

THIS WILL BE DEFINED IN THE NEXT VERSION OF THIS DOCUMENT.
7. SUPA Logic Statement Information Model

This section defines the classes, attributes, and relationships of the SUPA Logic Statement Information Model (LSIM).

7.1. Overview

A Goal policy rule (also called a declarative policy rule, or an intent-based policy rule) is a declarative statement that defines what the policy should do, but not how to implement the policy. In this draft, such rules are called SUPA Logic Statements.

This Section, and the following Sections, will be finished in the next version of this document.

7.2. Constructing a SUPAPLStatement

This section will be finished in the next version of this document.

7.3. Working With SUPAPLStatements

This section will be finished in the next version of this document.

7.4. The Abstract Class "SUPALogicClause"

A SUPALogicStatement is an abstract class that represents declarative (also called intent-based) policies. A SUPALogicStatement MUST contain at least one SUPAPolicyStatement. Such policies define a goal to be achieved, or a set of actions to take, but do not prescribe how to achieve the goal or execute the actions. This differentiates it from a SUPAECAPolicyRule, which explicitly defines what triggers the evaluation of the SUPAECAPolicyRule, what conditions must be satisfied in order to execute the actions of the SUPAECAPolicyRule, and what actions to execute.

This document defines two forms of a SUPALogicStatements. The first uses Propositional Logic (PL, see Section 3.2.4.2), while the second uses First-Order Logic (FOL, see Section 3.2.4.3).

Note that this document does not refer to a SUPALogicStatement as a "rule", since both types of SUPALogicStatements defined in this document are technically not "rules". Rather, they are types of zero-order and first-order logic statements.
If the SUPALogicStatement is expressed in PL, then it MUST consist of only the propositional connectives (i.e., negation, conjunction, disjunction, implication, and bi-implication (see Section 3.2.4.1)). Furthermore, statements in a PL are limited to simple declarative propositions that MUST NOT use quantified variable or predicates.

If the SUPALogicStatement is expressed in FOL, then it MUST consist of a set of logical predicates (i.e., a Boolean-valued function). The predicate can use all propositional connectives as well as two additional quantifiers (i.e., the universal quantifier and the existential quantifier).

A logical predicate MUST consist of a head clause, and MAY also contain a body clause. This enables the semantics of a SUPALogicStatement to be clearly differentiated from the semantics of other types of SUPAPolicies that use SUPAPolicyStatements (and other parts of the SPGIM), such as SUPAECAPolicyRules. While in principle higher order logics can be defined, this document is limited to defining a SUPALogicStatement using either PL or FOL.

When implemented in an information model, each PL or FOL statement MUST be defined as objects (i.e., a subclass of the SUPAPolicyStatement class; see Section 7). When an FOL statement is implemented in an information model, both the head and body clauses MUST be defined as objects (or sets of objects). However, a data model MAY map either a PL statement or an FOL statement to a more efficient form (e.g., by flattening the head and body objects into a single object).

7.5. The Abstract Class "SUPAPLStatement"

This section will be finished in the next version of this document.

7.5.1. SUPAPLStatement Attributes

This section will be finished in the next version of this document.

7.5.2. SUPAPLStatement Relationships

This section will be finished in the next version of this document.
7.5.3. SUPAPLStatement Subclasses

This section will be finished in the next version of this document.

7.5.3.1. The Concrete Class "SUPAArgument"

7.5.3.2. The Concrete Class "SUPAPLPremise"

7.5.3.3. The Concrete Class "SUPAPLConclusion"

7.6. Constructing a SUPAFOLStatement

7.7. Working With SUPAFOLStatements

7.7.1. SUPAFOLStatement Attributes

7.7.2. SUPAFOLStatement Relationships

7.7.3. SUPAFOLStatement Subclasses

7.7.3.1. The Concrete Class "SUPAGoalHead"

7.7.3.2. The Concrete Class "SUPAGoalBody"

7.8. Combining Different Types of SUPAFOLStatements
8. Examples

8.1. SUPAECAPolicyRule Examples

8.2. SUPALogicStatement Examples

8.3. Mixing SUPAECAPolicyRules and SUPALogicStatements

9. Security Considerations

This will be defined in the next version of this document.

10. IANA Considerations

This document has no actions for IANA.

11. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Andy Bierman, Bob Natale, Fred Feisullin, Liu (Will) Shucheng, Marie-Jose Montpetit.

12. References

This section defines normative and informative references for this document.

12.1. Normative References


12.2. Informative References


Authors’ Addresses

John Strassner
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95138 USA
Email: john.sc.strassner@huawei.com
Abstract

SUPA will define a generic policy model, an imperative (Event-Condition-Action, ECA) policy information model and a declarative (intent-based) policy information model which is the extension of the generic model, and a set of policy data models which will make use of the common concepts defined in the generic model. This memo will explore some typical use cases and demonstrate the applicability of SUPA policy models.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."
The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 19, 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 ................................................ 3
2. Conventions ................................................. 3
3. Terminology ................................................... 3
4. Framework ................................................... 4
  4.1. Network Manager/Controller ................................ 6
5. SUPA Examples ............................................... 9
  5.1. SES Use Case ........................................... 9
    5.1.1. Scenario .......................................... 9
    5.1.2. Generic Policy Models ............................ 10
    5.1.3. Programmatic approach - SUPA modeling .......... 11
    5.1.4. SUPA Data Model for SES Use Case ............... 12
  5.2. VPC Use Case .......................................... 17
    5.2.1. Generic .......................................... 17
    5.2.2. Example1 ......................................... 18
    5.2.3. Example2 ......................................... 20
  5.3. DC Link Use Case ...................................... 21
  5.4. Virtual SP Use Case ................................... 22
  5.5. Instant VPN Use Case .................................. 25
6. Security Considerations .................................... 26
7. IANA Considerations ........................................ 26
8. Acknowledgments ............................................ 26
1. Introduction

One of the ways for network service automation is using network management and operation software applications. The applications should not directly communicate with each network element; a hierarchical and extensible framework should be considered to hide the protocol specific and/or vendor specific details, high level network and service abstraction, and standardized programming API will be necessary.

SUPA will define policy generic models and data models, for service management and operation applications. [I-D.strassner-supa-generic-policy-info-model] defines a common set of concepts for various data models which may use different languages, protocols, and repositories.

Three generic models are defined in [I-D.strassner-supa-generic-policy-info-model]: Generic Policy Model, Eca Policy Rule Model, Logic Statement Model. The ECA information model is intended for dynamic service automation; while the Logic Statement Model is intended for expressing high requirements without being involved in network details.

Data models can be defined by developers/operators or by any third party, as long as they follow the common concepts defined in SUPA generic model. [I-D.chen-supa-eca-data-model] defines a policy data model of Event-Condition-Action (ECA), which is an example.

The generic data models will be used for domain or service specific data model. And there is no interoperability requirement for domain specific data models. The interoperability is guaranteed at the generic data model level via the common concepts.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

3. Terminology

DC Data Center
4. Framework
As shown in Figure 1, SUPA will define generic policy models, which are independent of services and use cases. Policy data models can be derived from the generic models. The data model will define high level, maybe network-wide policies. Policy data model will be used in conjunction with service data models to generate configurations for network elements. The service data model is use case specific and will be developed by operators or third parties, which is out the scope of SUPA.

The service management applications will send SUPA data models to the service management system, where policy making and automated policy enforcement will be performed, and the data models will be mapped to configuration of network elements. Configuration of network elements is vendor specific, using various protocols, such Netconf, Restconf, etc.

SUPA also make use of information collected from network elements. The information may include warning or fault event, load status, traffic statistics, etc, which can be used to adjust network configurations. This kind of automation is done through ECA data models.
4.1. Network Manager/Controller

The internal details of the network manager / controller may be out of the scope of SUPA, but explaining how it works may help people to understand and implement SUPA.

Network administrator can send service deployment and management request to network manager / controller via SUPA data models. The data models will be converted into network elements configuration snippets. The configuration change may be performed instantly, or later triggered by events. The network manager / controller has the intelligence to decide which network devices should be configured,
and what the configuration will be, which is derived from the actions specific in the data models explicitly or implicitly.

Network management related resources and information are stored in the network manager/controller, which contains the network topology (physical and virtual interconnection of network elements, etc), inventory (database of network elements, ports, device type, capabilities, etc.), protocol specific information, etc.

SUPA will make use of the existing work of other IETF WGs and other SDOs, such as if the topology data model is already defined in another IETF WG, SUAP will reference it rather than trying to define it again.

The network manager / controller will find out the list of network devices which should be configured for a specific demand or service.

For example, there is a configuration request:

All edge routers shall have SSH disabled.

An edge router is a router with connection to network(s) outside of the current network domain. The controller will query the topology database and find out all the routers with the attribute of "device-role == edge", or the controller may use more complicated algorithms to find out if a router is an edge route, which is implementation specific.

Similarly, another example is, the controller can make use of PCE engine to plan the links between DCs, and make sure the links are disjoint for better availability in case of failure. The PCE engine will be used in conjunction with the topology database to find out possible disjoint links.

The network manager / controller will also have other information, such as protocol specific information, traffic with TCP destination port 22 is SNMP traffic.
The network manager / controller also collect information from the
network device, such as events, logs, statistics, etc. The information
may come from SNMP TRAP, Syslog, NETCONF notification, and other
sources such as vendor specific protocols or extensions. The
collected information may be used in conjunction with SUPA ECA data
models for dynamic configuration change. An example use of the
information is, if the load on a link between two DC exceeds a
threshold, and there are multiple disjoint links between the two DCs,
traffic steering will be triggered.

Event: link_load > threshold
Condition: there are disjoint links
Action: perform traffic steering

Some of the events are already standardized, such SNMP TRAP and
NETCONF notification; some are implementation specific.

SUPA data models explicitly or implicitly specify network actions,
and the actions may be expanded into more detail actions if
necessary, and finally converted into protocol specific, vendor
specific network element configuration snippets.

In the previous example shown below again:

All edge routers shall have SSH disabled.

The action in this case is "disable SSH traffic", the network
manager / controller should converted this action into configuration
"disable traffic on TCP port 22" in the IP stack, or an ACL rule
which will drop traffic with TCP destination port 22.

The network manager / controller can support various types of
southbound interface, such as NETCONF, RESTCONF, SNMP, OpenFlow, etc,
which make it possible to support devices from different vendors.
This is implementation specific and out of the scope of SUPA.
5. SUPA Examples
5.1. SES Use Case

5.1.1. Scenario

Switched Ethernet services (SES) to Small and Medium Businesses business is a growing business segment of the service provider. As the Enterprise’s applications grow in demands in terms of the bandwidth and richness of applications, WAN optimization is needed to improve the service quality. SUPA policy data models can be used for maximizing the WAN performance by analyzing the traffic and performing application management and acceleration tools for the network.
In the use case below, Service Manager (SM) is used for service and policy definition and Network Manager (Controller) is used for network topology maintenance and mapping data models to detail network configurations.

While speed and bandwidth are at the forefront of the WAN Optimization there need to be tools in place to detect, diagnose, remedy and report application performance to ensure the SLAs for a customer are enforced.

The service is modeled in terms of what kind of service (Ethernet, VLAN), bandwidth (10Mbps- 10 Gbps), service package (platinum, gold, silver) etc.

Policy models are based on an Event condition action like:

1. Bandwidth usage alarm triggers data caching
2. Latency alarm triggers reduction of re transmission
3. WAN outage at a specific site can trigger geographic redundancy (provided the service is setup for GR)

The above are 3 of the primitives (Event condition action - ECA) on which the run time operations could be based on. When the service model is comprehensively designed with more possibilities (variables), more policy models could be implemented

5.1.2. Generic Policy Models

Requirements and configurations derived from above application scenarios can be described by service data model and policy data models as below:

Service data model can be used to describe attributes for the SES, including service package type (Platinum, gold etc), bandwidth bought by the subscriber (100Mbps, 10Gbps), connection name -copper/ GigE, latency, etc.

Policy data model describes a condition when the link capacity reaches 90%, Service prioritization and WAN optimization need to be enforced based on the customers service package. Event is the link utilization and condition is the usage and action is the WAN optimization. The actions could trigger multiple actions like data compression, protocol acceleration (like streaming gets priority) which are beyond the scope of SUPA.
ECA Policy:

Event: link_load > 90%

Condition: acceleration for service available

Action: data compression; protocol acceleration

It is assumed that the network management/controller module has the network topology and monitors the load on links in the topology.

When translating and processing the SUPA data model, the link information, including link attributes and load, will be provided by the network management/controller. If the load on a specific link exceeds a threshold, the network manager/controller will trigger actions specified in the model.

The actual actions may be vendor specific, network management/controller specific or device specific. The actions will be mapped into configuration for network devices. The network management/controller also need to figure out the set of network devices which need to be configured based on network topology together with some other information, such as service specific information. This is the internal functions of network management/controller, which is out of the scope of SUPA.

5.1.3. Programmatic approach - SUPA modeling

The advantage of the programmatic approach can be maximized by defining as many SUPA ECA models as possible in a top down approach.

In this use case, since this is a switched service, point to point traffic can be identified (by IP Address and port number) and segmented and whole bandwidth can be utilized by many applications simultaneously. Examples are: Print jobs, backups etc..

The benefit of the SUPA is in creating many policies upfront. As the operations grow in complexity SUPA can expand an existing policy by adding more variables. This is how reusable policies can be developed upfront and configuration and maintenance operations can be dealt by modeling and programmatic approach.
Logic Statement Model can also be called as declarative or intent model. This type of model will describe the service intention without specifying low level details, such protocol level or network device level detail, but just the service requirements itself.

5.1.4. SUPA Data Model for SES Use Case

The following model segment is based on [I-D.chen-supa-eca-data-model].

In the model, the event can be expressed using some standardized names, such as the SNMP TRAP (linkDown, linkup, Failure, etc), or "link-load > 90%".

The condition(s) can be expressed using script, such as Python script hasAcceleration("ses") or Python script hasDisjointLinks(DC1, DC2). The script is supposed to be interpreted by a script tool and there are various script tools, the implementer can use any one as they like, either an existing one like Python or a new one. The script itself is out the scope of SUPA; a simple value will be return by the script tool. Some complex combination of conditions can be expressed using script which will give more flexibility.

When handling the condition script, the script tool will be called to process the script. In this case, the script will communicate with service management system and/or the tenant database to find out if any optimization is available for this service or tenant.

Script can also be used for actions.

An example of the script using Python is:

```python
service-name="ses"
// input: service-name, type: string
// output: enhancement, type: string or None if no enhance
def queryEnhancementInCapability(service-name):
    for i in range(len(capability-models)):
        if getServiceName(capability-models[i]) == service-name:
            return getEnhancement(capability-models[i])
```

Vadrevu et al. Expires April 19, 2016 [Page 12]
return None

// input: service-name, type: string
// output: True/False, type: boolean
def hasAcceleration(service-name):
    if queryEnhanceinCapability(service-name) == None:
        return False
    else:
        return True

The capability data models are supposed to contain the following:
<capability-data-model>
...
<services>
    <service>
        <service-name>ses</service-name>
        <service-enhance>compression</service-enhance>
    </service>
    <service>
        ...
    </service>
<services>
</capability-data-model>
The SUPA XML example is shown below:

```xml
<supa-policy>
  <supa-policy-name>ses-policy</supa-policy-name>
  <supa-policy-priority>0</supa-policy-priority>
  <supa-policy-validity-period>
    <start>00-00-0000</start>
    <end>00-00-0000</end>
  </supa-policy-validity-period>

  <supa-policy-target>
    <profileType>domain</profileType>
    <asDomainName>operatorA-domain1</asDomainName>
    <businessTypeName>ses</businessTypeName>
    <instance>
      <instanceName>// detail to be provided by controller
      <flow-filter>
        <src-ip-addr>10.1.1.0/24</src-ip-addr>
        <dst-ip-addr>20.1.1.0/24</dst-ip-addr>
      </flow-filter>
      ....... // more filters
      <flow-filter>
      </instanceName>
    </instance>
  </supa-policy-target>

  <supa-policy-atomic>
    <supa-ECA-policy-rule>
      <policy-rule-deploy-status>...... // to be provided by controller
      <policy-rule-exec-status>...... // to be provided by controller
    </supa-ECA-policy-rule>
    <supa-ECA-component>
      <supa-policy-events>
        <has-policy-events>YES</has-policy-events>
      </supa-policy-events>
      <supa-policy-conditions>
        <has-policy-conditions>YES</has-policy-conditions>
        <conjunctive-type>and</conjunctive-type>
      </supa-policy-conditions>
      <supa-policy-actions>
        <action-execution>YES</action-execution>
      </supa-policy-actions>
    </supa-ECA-component>
  </supa-policy-atomic>
</supa-policy>
```
The data model can be augmented according to developers’ need. The developers can add vendor specific events, conditions and actions via "augment" Yang function in [RFC6020], as suggested in [I-D.chen-supap-eca-data-model]. An example of augmented model is shown below:

```xml
// ------- yang model snippet start -------
augment ""/supa:supa-policy/supa:supa-policy-statement/supa:event-list" {
    leaf my-event {
        description "customized event";
        type bool;
    }
}
// entity or script or boolean
```

The data model can be augmented according to developers’ need. The developers can add vendor specific events, conditions and actions via "augment" Yang function in [RFC6020], as suggested in [I-D.chen-supap-eca-data-model]. An example of augmented model is shown below:

```xml
// ------- yang model snippet start -------
augment ""/supa:supa-policy/supa:supa-policy-statement/supa:event-list" {
    leaf my-event {
        description "customized event";
        type bool;
    }
}
// entity or script or boolean
```
augment "supa:supa-policy/supa-policy-statement/supa:condition-list" {
  container my-condition{
    description "The bandwidth threshold, unit is Mbps";
    type uint32;
  }
}

augment "supa:supa-policy/supa-policy-statement/supa:action-list" {
  container my-action-drop{
    description "drop packets";
    type string;
  }
}

// ------- yang model snippet end -------

// ------ xml model snippet start ------

// assume the above augmentation is in a name space "mymodel"
<supa-policy>
  ...... // others
  <supa-policy-statement>
    <event-list>
      <event-name>
        ...... // other events
      </event-name>
      <mymodel:my-event>true</mymodel:my-event> // added event
    </event-list>

    <condition-list>
      <condition-linkThreshold>
        ...... // other conditions
      </condition-linkThreshold>
      <mymodel:my-condition>32</mymodel:my-condition> // added condition
    </condition-list>

    <action-list> // added action
      <actionName>
        Vadrevu et al. Expires April 19, 2016 [Page 16]
5.2. VPC Use Case
5.2.1. Generic

In practice, a public cloud operator can virtualize the cloud resources into multiple isolated virtualized private clouds and provide them to tenants. Such a virtualized private cloud is referred to as a VPC. In a typical VPC provided by, e.g., Alibaba or Amazon, through a control portal, a tenant can establish and manage the network easily, for instance, deploying or removing virtualized network devices (e.g., virtualized routers and virtualized switches), adjusting the topology of VPC networks, specifying packet forwarding policies, and deploying or removing virtual services (e.g., load balancers, firewalls, databases, DNS, etc.). The network functionalities that the tenant can access are virtualized and actually performed by the VMs located on the servers connected through physical or overlay networks. Note that the servers may be located in different data centers which are geographically distributed.

The manipulation of the virtualized VPC network may also affect the configuration of physical networks. For instance, when a tenant newly deploys two VMs in the VPC which are located in different DCs, the VPC control mechanism may have to generate a VPN between two DCs for the internal VPC communication. Therefore, the control mechanism for a VPC should be able to adjust the underlying network when a tenant changes the network or service deployment of the virtual VPC network.

In many cases, a tenant may need to specify how the VPCs are connected to its enterprise cloud networks. For instance, a tenant may want to deploy multiple VPNs to connect the VPC with its private
cloud networks and specify the policies to steer the traffics through different VPNs in different conditions. Note that the VPCs that the tenant may be located in different geographic regions and the VPNs to those VPCs may need to be generated at run time.

In addition, a VPC, often provides other value added services (e.g., database Services, DNS) for VMs in certain VPCs. The VMs and the value added services could be located in different DCs, or even provided by different vendors. VPNs are configured for the VPCs to provide connection to the internal services in tenant’s own DC or organization, and to create and manage VPNs to internal services. The access of VMs to data resources should be controlled. For instance, the VMs in a VPC can access the database services only when the tenant has deployed database into its VPC through the control portal.

5.2.2. Example1
When cloud / DC operator signs a contract with customer, resource information such as network bandwidth, storage size, number of CPU, memory size, etc, will be specified.

But in deployment, the resources may be located in multiple distributed data centers, and tunnels will be created to interconnect these resources, which will make it look like one seamless entity - a virtual DC. There could be quite a number of tunnels, and the tunnels are dynamic, either for the reason of load balancing purpose or VM migration, or other reasons. This will make it difficult to configure the service statically or manually, service automation is very necessary.

The service management system will have a repository of available resources, including the topology. And also the management system will have the customer specific information (location, SLA, agreed resources, etc).

The administrator can send the service requirement to the management system by a high level data model, which can further be mapped to low level detail data models, then finally mapped to configurations of network devices.

Target: Provide VPC service to customer A with specified resources and function (storage, computing, DNS, etc)

Declarative policy:

1. Allocate the required services on DCs according to a user’s profile

2. Services located in multiple distributed DCs must be interconnected via VPNs
3. The VPNs associated to the services provided for a user must match the user’s profile in terms of latency, speed and bandwidth.

5.2.3. Example2

As shown in the above figure, when a VPC tenant move from one location to another, where it is near to another DC, and the network load between the new DC and the previous DC is low, the tenant’s VM should be migrated to the new DC in order for better user experience.

After the VM is moved to the new DC, the network related to the VM must be updated accordingly.

Target: Perform VM migration when user location changed and the network load between the DCs is low

ECA Policy:

Event: a VPC user’s location is changed (near to another DC)

Condition: network_load(DC_old, DC_new) < threshold

Action:

1. Migrate the VM to the new data center (DC_new)
2. Update the VPNs connecting the user’s services

In the above model it is assumed that the network management/controller has the network topology, including attributes of the links, such as bandwidth. The network management/controller also monitors the real-time load on the links in the network topology.

The user’s location can be identified by the user’s IP address. When a user login, the network management/controller will check the user’s IP address against an IP address database, such as the IP address assignments by IANA.

The network management/controller also maintain a mapping of DCs and IP address segments, say, a DC should serve users in a near location which can be identified by IP address segments. Though this is not always the case, sometimes the geographical distribution of network resource will also need to be considered besides the location (IP address). But, anyway, a mapping of DC and the IP address it should serve should be maintained.

If the controller detects a location change and a new DC is possible for the user, and the network load between the new DC and the old DC is low, then VM migration will be triggered and related network configuration will be performed.

5.3. DC Link Use Case

DCs usually have multiple external links, either to other DCs or to the internet. Because of the dynamic nature of network traffic, the load on a link may vary at different times of a day, e.g. link mainly carries enterprise traffic may have a high load in the working hours but less traffic in the night. Some events may also impact the load of links, such as one link is physically damaged and the load in it will go to another link.

In order to make full use of the bandwidth of the links, dynamic traffic steering is necessary for SLA meanwhile with full use of network resource.

```
+--------+ +--------+
|        | |        |
+--------+ +--------+

Vadrevu et al. Expires April 19, 2016 [Page 21]```
Figure 6 Multiple Disjoint Links Between DCs

Target: DC have multiple external links; when the load on a link is too high, perform traffic steering for better bandwidth resource usage.

ECA Policy
Event: load on a DC link exceeds threshold
Condition: multiple disjoint links between DCs
Action: steer some traffic to link with low load

In the above model it is assumed that the network management/controller has the network topology, including attributes of the links, such as bandwidth. The network management/controller also monitors the real-time load on the links in the network topology.

The network topology also contains the connections between network devices. The network management/controller will be able to figure out if there are multiple disjoint links between two DCs. The algorithm for finding out disjoint links is out of the scope of this SUPA.

When the network management/controller detects the load on a link exceeds a threshold, it can check if there are multiple disjoint links, and if yes, it will further perform necessary actions specified in the model.

5.4. Virtual SP Use Case

Virtual network operators usually do not have a complete network, including access network, metro network, and backbone network. They
need to rent network from other operators. An example is, a virtual
operator do not have the access network, traffic of broadband
network subscriber will go through other operators access network,
and then be directed to the virtual operators network from the BNG
via tunnels. In some other cases, the virtual operators may not have
the backbone network, the network islands and DCs will be connected
by tunnels.

The problem in this case is, virtual network operators have no
control over the tunnels and they cannot decide the exact path that
the tunnel should go through. In some scenarios, if the tunnel goes
through the border of two network operators, or the tunnel goes
through an area where network load is too high, the SLA will be a
problem. Virtual network operators who run the business in a large
geographical region often run into this problem. Due to cost issue,
virtual network operators cannot buy service from other operators
with critical SLA.

A possible solution is, the virtual network operator rent or put
some routers in network operators’ DCs, and then configure tunnels
between the routers and perform traffic steering. In this way,
virtual network operators can have control over the tunnels, pin
down the path. When a problem is detected, such as QoS of a tunnel
is below a threshold, virtual network operator can perform "network
wide" optimization, reconfigure the tunnels and/or perform traffic
steering.

```
+------------+                             +------------+
| vNetwork 1 |                             | vNetwork 2 |
+------------+                             +------------+
/                                    /
/                                    /
+--------------+              +--------------+
| +----------+ |   tunnel 1   | +----------+ |
| | Router 1 | |--------------| | Router 2 | |
| +----------+ |              | +----------+ |
| Operator DC1 |              | Operator DC2 |
+--------------+              +--------------+
```

Vadrevu et al.          Expires April 19, 2016                [Page 23]
If direct tunnel is built between virtual operator’s networks (e.g. vNetwork1-to-vNetwork3), route is out of control -- the route may go through network node with problems, or with high load, or cross border of different operators where QoS cannot be guaranteed.

In this case, the virtual network operator can configure three tunnels rather than one to connect vNetwork1 to vNetwork3: vNetwork1-to-Router1, Router1-to-Router2, Router2-to-vNetwork3.

After the initial network configuration is finished, if any problem is detected in any tunnel, the network management system can perform network wide optimization, taking all the routers into account and working out another set of tunnels if necessary.

ECA Policy:

Event: QoS parameters < threshold

Condition: multiple disjoint tunnels available

Action: Network wide tunnel optimization + traffic steering

In this case, the virtual SP can monitor the real-time QoS parameters between the virtual networks and the rented routers. If the QoS parameters exceed a threshold, and the virtual has deployed multiple rented routers which can provide multiple disjoint tunnels, then the network management/controller can trigger network wide tunnel optimization and/or perform traffic steering.

When performing the tunnel optimization, the network management/controller may terminate the tunnel(s) which go through specific network area with problems, and/or build new tunnels, and/or perform network wide traffic steering. This will give the operator a lot of flexibility in controlling the network.
The traffic steering may need to be combined with the network topology, and dynamically distribute traffic in the whole network.

5.5. Instant VPN Use Case

Traditionally, when an operator needs to deploy VPN service for an enterprise customer, they will send a service staff to the customer site and make the wire connection between the CE and PE; the service staff will also collect the configuration information, e.g. port/frame/slot of PE, PE ID, etc, and then send the information back to the management system, and the management system will configure the network according to this information together with the customer’ information (such as bandwidth, SLA, etc). The problem of this approach is that the service staff needs to collect the connection information and feedback to the management system, and MUST make sure the information matches the actual connection. This operation is error prone.
New approach should not count on the physical / geographical information feedback by the service staff, minimize the operation procedures. The CE should send authentication (with credentials) request to the PE, and PE should forward the request to the management system together with port/frame/slot on which the request is received, the PE ID etc.

Target: Configure VPN for an enterprise customer to connect its enterprise network with VPC

ECA Policy:

Event: service management system receive a CE request for VPN creation (forwarded by PE)

Condition: Authentication OK

Action: Configure VPN based on received request, including user grade and physical info (port/slot/frame/route id, etc, from which the request is received)

6. Security Considerations

Since SUPA models can be used to generate configurations for network elements, the management applications which send models to service management system must go through authentication and authorization.

7. IANA Considerations

This memo does not have any requirement to IANA.

8. Acknowledgments

This document has benefited from reviews, suggestions, comments and proposed text provided by the following members, listed in alphabetical order: Juergen Schoenwaelder, John Strassner, James Huang
9. References

9.1. Normative References


9.2. Informative References


Authors' Addresses

Narasimha Vadrevu
VN Telecom Consultancy
Cupertino, California

Email: vadrevun@von20.com