ETSI GR NFV-IFA 042 V4.1.1 (2021-11)

Network Functions Virtualisation (NFV) Release 4 Management and Orchestration; Report on policy information and data models for NFV-MANO

Reference

DGR/NFV-IFA042

Keywords

data, MANO, model, NFV

Contents

Intellectual Property Rights

Foreword

This Group Report (GR) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document performs a study on policy models in NFV-MANO and conducts following research:

- Enriching examples of NFV-MANO policy representations based on use cases described in ETSI GR NFV-IFA 023 [i.6].
- Determining concepts that could be considered by policy model design, such as policy expression paradigm, policy conflict, policy classification.
- Deriving recommendations on policy model design.
- Gap analysis on the existing policy models from other organizations and feasibility analysis on whether it could be referenced by the policy model used by NFV‑MANO.
- Deriving recommendations on the subsequent normative work on the policy model applicable to NFV-MANO.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] TM Forum GB922 Policy (R 18.0.1): "Framework Standard Information Common Business Entities-Policy".

[i.2] TM Forum TR234 (R 14.0.11): "ZOOM Information Model Snapshot".

[i.3] IETF draft-ietf-supa-generic-policy-data-model-03 (May 30, 2017): "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)".

[i.4] IETF draft-ietf-supa-generic-policy-info-model-04 (June 18, 2017): "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)".

[i.5] IETF RFC 8328: "Policy-Based Management Framework for the Simplified Use of Policy Abstractions (SUPA)".

[i.6] ETSI GR NFV-IFA 023: "Network Functions Virtualisation (NFV); Management and Orchestration; Report on Policy Management in MANO; Release 3".

[i.7] ETSI GS NFV-SOL 012: "Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; RESTful protocols specification for the Policy Management Interface".

[i.8] ONAP: "Policy Administration Point (PAP) Architecture".

NOTE: Available at https://docs.onap.org/projects/onap-policy-parent/en/latest/pap/pap.html#pap-label.

[i.9] ONAP: "Policy Framework Architecture".

NOTE: Available at https://docs.onap.org/projects/onap-policy-parent/en/latest/architecture/architecture.html.

[i.10] ETSI GS NFV-IFA 011: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; VNF Descriptor and Packaging Specification".

[i.11] Void.

[i.12] ETSI GR NFV-IFA 041: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Report on enabling autonomous management in NFV-MANO".

[i.13] ETSI GS NFV-IFA 013: "Network Functions Virtualisation (NFV) Release 4; Management and Orchestration; Os-Ma-nfvo reference point - Interface and Information Model Specification".

[i.14] ONAP: "A short Introduction to APEX".

NOTE: Available at A short Introduction to APEX — onap master documentation.

[i.15] ETSI GR NFV 003: "Network Functions Virtualisation (NFV); Terminology for main concepts in NFV".

[i.16] IETF draft-wwx-netmod-event-yang-10: "A YANG Data model for ECA Policy Management" November 1, 2020.

[i.17] PlantUML tool download.

NOTE: Available at https://sourceforge.net/projects/plantuml/.

[i.18] Pyang tool download.

NOTE: Available at https://pypi.org/project/pyang/1.7.4/.

[i.19] ETSI GS NFV-SOL 001: "Network Functions Virtualisation (NFV) Release 4; Protocols and Data Models; NFV descriptors based on TOSCA specification".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR NFV 003 [i.15] apply.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GR NFV 003 [i.15] and the following apply:

NOTE: An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in ETSI GR NFV 003 [i.15].

APEX	Adaptive Policy EXecution
APEX PDP	APEX Policy Deployment Point
ECA	Event-Condition-Action
EPRIM	ECA Policy Rule Information Model
GPIM	Generic Policy Information Model
MEDA	Match-Establish-Decide-Act
OODA	Observation-Orientation-Decision-Action

PAP	Policy Administration Point
PF	Policy Function
PV	Policy Variable
TMF	Telemanagement Forum
TMF PDP	TMF Policy Decision Point
TMF PEP	TMF Policy Enforcement Point
TMF PXP	TMF Policy eXecution Point

4 Background and overview

Policy is one of the important managed objects of NFV-MANO. Policy can facilitate the implementations of several NFV-MANO functions, such as auto-scaling, auto-healing, root cause analysis, energy optimization, etc. Moreover, policy plays an important role in the realization of NFV-MANO automation, even autonomy.

Release 3 FEAT07 work has completed the study and specifications on many aspects of policy management in NFV-MANO, such as:

1) Analysis on policy management use cases and recommendations on policy management functional framework.

2) Specifications on functional requirements for policy management.

3) Specifications on policy management interfaces.

However, the structure and content of policy are not specified by the existing specifications, that is, the policy information model and the policy data model applicable for NFV-MANO are not specified. The lack of the standardization of policy models causes several issues:

- The current policy interactions and operations of various entities related to policy (NFV-MANO and the OSS/BSS) can only be realized in a proprietary manner.
- Operators and vendors have ambiguities in the management of policies.
- The existing specifications of stage 2 and stage 3 cannot support the actual deployment of policy management entity functions.

The present document is targeted to remove the above gaps on NFV-MANO policy model.

5 NFV-MANO policy model potential requirements analysis

5.1 NFV-MANO policy representations

5.1.1 Overview

Clause 5.1 of the present document describes NFV-MANO policy application scenarios with more details related to policy content and framework, which will lead more features and requirements to be extracted for defining policy information model and policy data model applicable for NFV-MANO. Clause 5.8 of ETSI GR NFV-IFA 023 [i.6] lists several use cases on applying policy management in NFV-MANO, covering the scenarios on how NFV-MANO policies can be applied to the lifecycle management of specific NS, VNF or virtualized resource instances.

In ETSI GR NFV-IFA 023 [i.6] use cases, some kinds of policies are pre-defined in VNFD and some kinds of policies are transferred over NFV-MANO reference points without defining the policy expression format.

In ETSI GS NFV-SOL 012 [i.7], a RESTful protocol and data model fulfilling the requirements related to policy managements interfaces over NFV-MANO reference points are specified. ETSI GS NFV-SOL 012 [i.7] regards a policy as an artefact.

Determination of policy expression format depends on the policy information model and policy data model that is used for creating the policy.

To express the content of a policy, ECA (Event, Condition, Action) is used as an example of policy expression format. This policy model can be implemented as imperative policies, as ECA is also popular in policy management specifications from other SDOs.

NOTE 1: The present document uses ECA expression format to determine basic elements of a policy, but it is not restricted to use other policy expression formats.

NOTE 2: TOSCA imperative workflows can be used to implement the LCM operations in the NSD and VNFD, as described in Annex E of ETSI GS NFV-SOL 001 [i.19]. These imperative workflows are not regarded as "imperative policies", but instead as a way to sequence a set of tasks and operations based on some triggers performed on the topologies defined in TOSCA templates.

5.1.2 NFV-MANO policy representation for VNF auto-scaling

5.1.2.1 Analysis description

The policy representation analysis in this clause references use case 'Auto-scaling policy pre-defined in the VNFD' described in clause 5.8.5 of ETSI GR NFV-IFA 023 [i.6], in which the auto-scaling policy is pre-defined in the VNFD.

The analysis describes an informative process on defining policy format and content of VNF auto-scaling policy.

As an example, possible events, conditions and actions associated to VNF auto-scaling policy can be modelled as an ECA policy:

- Event: (the event that activates the trigger's action):
    1) The VNFM receives certain virtualized resources performance data from the VIM or VNF indicator notifications reported by multiple VNFs.
- Condition: (logical combination of certain specific conditions):
    1) The performance notification and corresponding data are reported within a period of time T1 in the recent period (data validity period).
    2) A period of time T2 has passed since the last time the policy was successfully triggered (policy cool down period).
    3) One or more VNF performance indicators (such as CPU occupancy, memory usage) have increased or decreased by a specific value or ratio.
    4) The value of the "autoScalable" attribute of the "VnfConfigurableProperties" (if true, auto-scaling can be performed).
    5) The current date of the system when the policy is being executed within a specified time interval, e.g. between 0:00 and 06:00 AM of each day.
- Action: (perform different types of actions according to different combinations that satisfy the conditions):
    1) Scale a single aspect of a VNF (a specific VDU type).
    2) Scale multiple aspects of a VNF (multiple VDU types) simultaneously.
    3) Wait a certain amount of time (e.g. 30 minutes) and scale a single/multiple aspects of a VNF.

NOTE: The conditions in the example above are illustrative and do not represent a final and complete list of conditions. For instance, other aspects regarding the check of network connectivity changes for the scaling or impacts on other VNFs when scaling can also be considered.

According to ETSI GS NFV IFA 011 [i.10], there are two ways to scale a VNF. As shown in figure 5.1.2.1-1, one is to scale the individual scaling aspect of a VNF, that is, different aspects of a VNF can be individually scaled to a predefined number, and the corresponding operation is "ScaleVnf operation"; The other is to scale a VNF to a pre-defined target size, that is, multiple aspects of a VNF can be simultaneously scaled to a pre-defined level, and the corresponding operation is "ScaleToLevel operation".

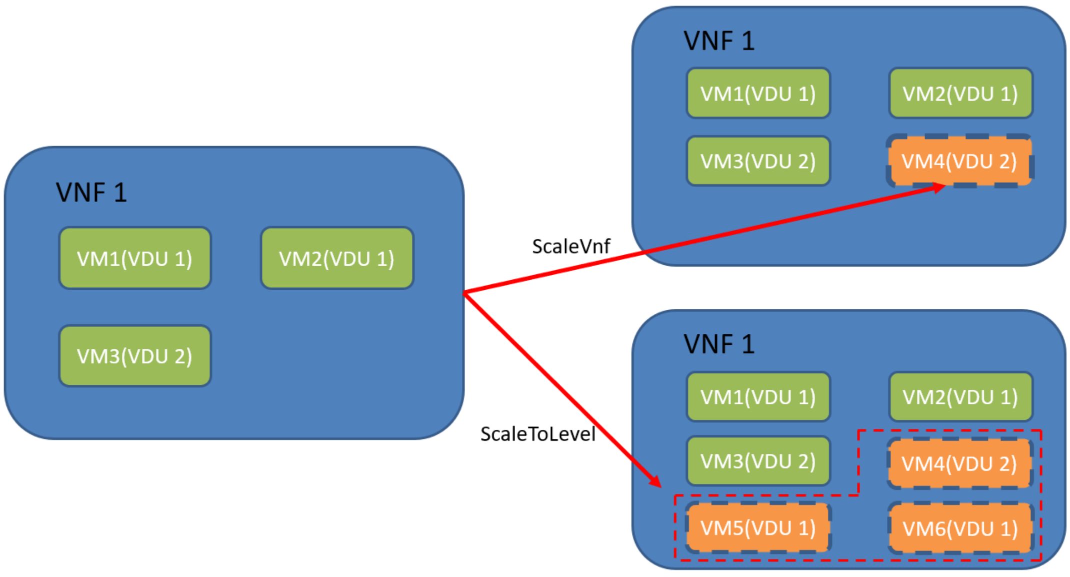

Figure 5.1.2.1-1: Two scenarios of VNF scaling operations

5.1.2.2 An example of imperative policy paradigm

Based on the representation of VNF auto-scaling policy described in clause 5.1.2.1, a simple example of imperative auto-scaling policy paradigm using formal English is described in this clause:

- **WHEN** a VNF Indicator Notification is received.
- **IF** {Data Validity Period < 3 min **AND** Time Since Last Trigger > 6 min **AND** CPU Occupancy > 80 % **AND** autoScalable == True}.
- **THEN** execute and action to scale out a VNF by adding one VM based on a type of VDU.

5.1.3 NFV-MANO policy representation for NS self-healing

5.1.3.1 Analysis description

The policy representation analysis in this clause references use case "Transferring NS healing policy to the NFVO" described in clause 5.8.4 of ETSI GR NFV-IFA 023 [i.6]. High-level NS self-healing policy information description is given in ETSI GR NFV-IFA 023 [i.6] and for the NS self-healing policy, the PAP is the OSS/BSS and the PF is the NFVO. In this analysis, additional details on the representation of NS self-healing policy content are provided.

The NS self-healing actions to be performed in the policy depend on the detected failures. In some simple situations, it is effective to trigger the healing policy at the same level where the cause of the failure is detected. For example, if the VNFM determines that the root cause of a VNF function failure is because of high usage of a VNF, the VNFM executes the action defined in a VNF healing policy to scale up/out the VNF. However, in a different situation, if the VNF function failures are caused by VM failures, the VNFM can not handle this failure alone as the VNFM cannot determine the root cause of such failures. In such a case, the NFVO has a better perspective to handle this type of failures and takes the role to perform root cause analysis by accessing corresponding information from the VNFM and VIM. In this case, the NFVO executes NS self-healing or VNF healing policies. In complex scenarios, it is possible that multiple policies have to be triggered at different entities to resolve the failure. From the perspective of NFV-MANO, one or more functional blocks (NFVO, VNFM, VIM) can execute actions defined in healing policies to handle system failures.

Considering different scenarios, the NS self-healing policy can be represented in two ways as listed below:

- One policy composed of one main policy and many sub-policies.
- One logical policy composed of many independent policies.

The NS self-healing policy representation depends on the requirements from policy designers and production networks. Representation #1 is suitable if one main policy cooperates with many sub-policies to act as a chain to perform NS healing operations. Representation #2 is suitable if an NS self-healing logical policy is composed of many independent policies and these policies form a policy chain to act as a single logical policy.

5.1.3.2 NS self-healing policy representation #1

In this representation, NS self-healing policy is composed of one main policy and many sub-policies.

Figure 5.1.3.2-1 shows the relationship between the main policies and the sub-policies. An NS self-healing policy includes one main policy and one or multiple sub-policies. According to the requirements from policy designers and production networks, one main policy cooperates with multiple sub-policies to act as a chain to perform NS healing operations. The main policy is the only starting point of the NS self-healing policy. Sub-policies are triggered by main policy only. A sub-policy could be triggered by more than one main policy if the sub-policy is part of more than one main policy. According to the combination of the conditions satisfied in the main policy, different sub-policies will be triggered explicitly by the main policy.

Therefore, there are some pre-conditions for NS self-healing policy representation #1:

- The sub-policies associated with the main policy are designed before or at the same time as the main policy to ensure the main policy can trigger the sub-policies successfully.
- The sub-policies are on-boarded before they are triggered by the main policy.

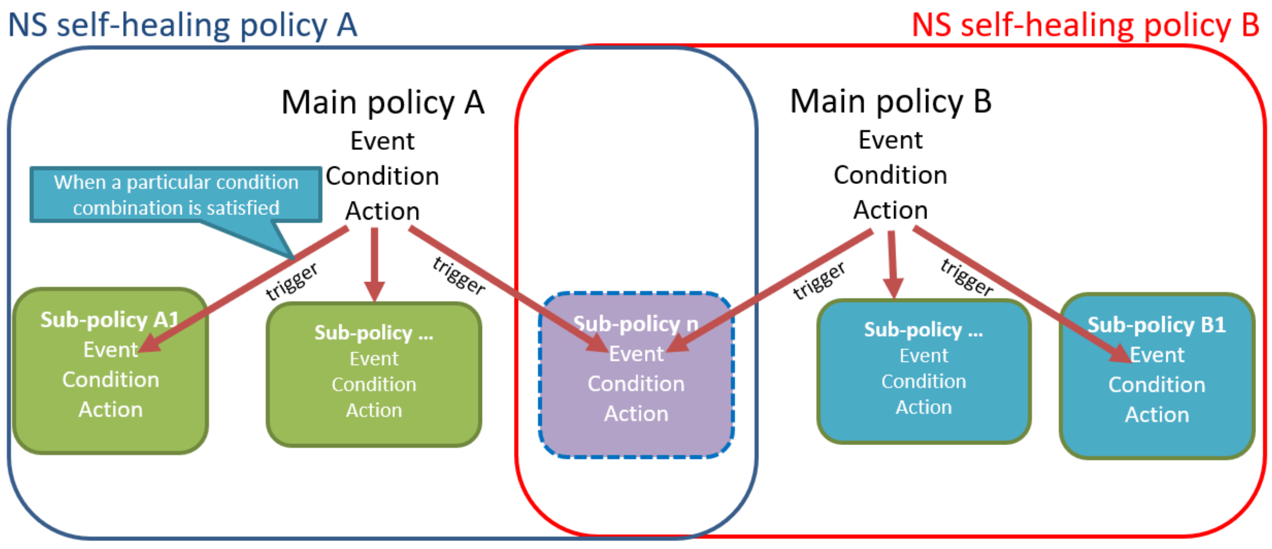

Figure 5.1.3.2-1: Relationship between main policies and sub-policies

In summary, the execution of the main policy depends on the cooperation of the sub-policies. Sub-policies can be independently designed and activated. However, the design, activation and execution of the main policy depends on the availability of the sub-policies. The main policy is not executed successfully until the relevant constituent sub-policies that are triggered are executed also successfully.

Below is an example of NS self-healing policy with representation #1. In this example, policy A is the main policy and policy B and policy C are two sub-policies within policy A. In this example, NS self-healing policy is designed to heal the failures of VNFs using one main policy. The root cause of the failure of VNFs comes from the faults of VNFs or VMs. The NFVO performs root cause analysis and determines which sub-policy to trigger. The NFVO requests the VNFM to trigger the VNF healing sub-policy if the faults are due to VNF specific faults. The NFVO triggers the VM healing sub-policy if the faults are due to the faults of VMs. The first trigger event to the main policy is expected to be fault information and data reported by the VNFM/VIM.

NOTE: The conditions in the example are illustrative and do not represent a final and complete list of conditions.

In this representation, possible events, conditions and actions associated to NS self-healing policy A can be modelled as an ECA policy:

- Event: (the event that activates the trigger's action):
    1) The NFVO receives fault information and data reported by the VNFM/VIM.
- Condition: (logical combination of certain specific conditions):
    1) The fault notification and corresponding data are reported within a period of time T1 in the recent period (data validity period).
    2) A period of time T2 has passed since the last time the policy was successfully triggered (policy cool down period).
    3) The source of the fault: only from the VIM, only from the VNFM, from the VIM and the VNFM.
    4) The content and correlation of the fault: the standby MPU offline problem of the VNF failure type is perhaps caused by VM hardware error (disconnected or unable to operate normally, etc.), which indicates that there is a correlation between VNF/VM failures; the problem that the CPU occupancy rate fluctuates frequently is perhaps caused by the abnormality of the VM OS, which indicates that there is a VNF/VM failure correlation.
- Action: (perform different types of actions according to different combinations that satisfy the conditions):
    1) According to the specific combinations of conditions, the NFVO determines the fault type as a VNF independent fault, and sends a notification to the VNFM to explicitly trigger the VNF healing sub-policy B.
    2) According to the specific combinations of conditions, the NFVO determines the fault type as a VM independent fault or a VNF fault caused by VM fault, and explicitly triggers VM healing sub-policy C.

In this representation, possible events, conditions and actions associated to VNF healing sub-policy B can be modelled as an ECA policy:

- Event: (the event that activates the trigger's action):
    1) The main policy A determines to trigger the sub-policy B.
- Condition: (logical combination of certain specific conditions):
    1) The performance data are reported within a period of time T3 in the recent period (data validity period).
    2) A period of time T4 has passed since the last time the policy was successfully triggered. (policy cool down period).
    3) Fault information: such as VNF offline, VNF function failure, VNF load anomalies, etc.
- Action: (perform different types of actions according to different combinations that satisfy the conditions):
    1) The VNFM determines to perform healing operations on the VNFs, such as reconnection and restart of the VNFs.
    2) The VNFM determines to heal the VNF(s) by allocating other VMs or re-allocating existing ones.

In this representation, possible events, conditions and actions associated to VM healing sub-policy C could be modelled as ECA policy:

- Event: (the event that activates the trigger's action):
    1) The main policy A determines to trigger the sub-policy C.
- Condition: (logical combination of certain specific conditions):
    1) The performance data are reported within a period of time T5 in the recent period (data validity period).

2) A period of time T6 has passed since the last time the policy was successfully triggered (policy cool down period).

3) Fault information: anomalies in VM usage rate, VM OS failure, VM hardware failure, etc.

- Action: (perform different types of actions according to different combinations that satisfy the conditions):

  1) The NFVO determines to perform VM healing operations, including but not limited to VM migration, rebuild, shutdown/start, etc.

5.1.3.3 NS self-healing policy representation #2

In this representation, NS self-healing policy is a logical policy composed of many independent policies.

Figure 5.1.3.3-1 shows the relationship between policies. According to the requirements from policy designers and production networks, a NS self-healing logical policy can be composed of multiple independent policies. These policies form a policy chain to act as a single logical policy. It is assumed that, Policy designers have a global perspective to design policies to form different policy chains. These policies could be stored in the NFVO and executed by the NFVO.

NOTE 1: Policies included/associated in the NS self-healing policy can be stored in other FBs and executed by other FBs. Policy management interface operations as defined in ETSI GS NFV-SOL 012 [i.7], can be used to coordinate the policy transfer and execution.

In this representation, policies included by a specific logical policy can be triggered independently by other policies. Therefore, there are pre-conditions for NS self-healing policy representation #2:

- The policies included by a logical policy are in activation status.

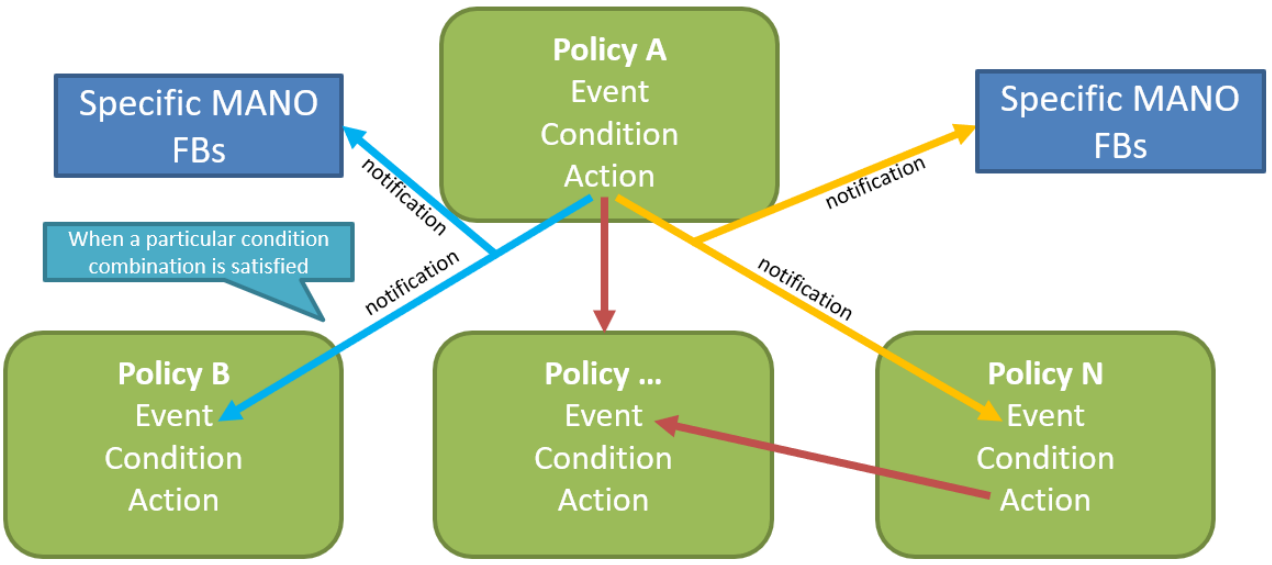

Figure 5.1.3.3-1: Relationship between independent policies

In summary, there is no dependency relationship between policies. Every policy can run and be triggered individually. There is an unspecified association relationship between policies to form a policy chain, thus forming a logical NS self-healing policy (e.g. there is no specific policy description).

Below is an example of NS self-healing policy with representation #2. In this example, policy A, policy B and policy C are three independent policies. NS self-healing policy is designed to heal the failures of VNFs by chaining different policies. The root cause of the failure of VNFs comes from the faults of VNFs or VMs. The NFVO performs root cause analysis and determines which policy is to trigger. The NFVO sends additional failure information details (if not present with the VNFM) and a notification and requests the VNFM to trigger the VNF healing policy if the faults are due to VNF specific faults. The NFVO determines to trigger the VM healing policy if the faults are due to the faults of VMs. When compared to the other representation, in representation #2, the first trigger event can be fault information and data reported by the VNFM/VIM or other specific types of fault information.

NOTE 2: The conditions in the example are illustrative and do not represent a final and complete list of conditions.

In this representation, possible events, conditions and actions associated to NS root cause analysis policy A can be modelled as an ECA policy:

- Event: (the event that activates the trigger's action):
    1) The NFVO receives fault information and data reported by the VNFM/VIM.
    2) The NFVO receives other specific types of fault information.
- Condition: (logical combination of certain specific conditions):
    1) The fault notification and corresponding data are reported within a period of time T1 in the recent period (data validity period).
    2) A period of time T2 has passed since the last time the policy was successfully triggered (policy cool down period).
    3) The source of the fault: only from the VIM, only from the VNFM, from the VIM and the VNFM.
    4) The content and correlation of the fault: the standby MPU offline problem of the VNF failure type is maybe caused by VM hardware error (disconnected or unable to operate normally, etc.), which indicates that there could be a correlation between VNF/VM failures; the problem that the CPU occupancy rate fluctuates frequently could be caused by the abnormality of the VM OS, which indicates that there is a VNF/VM failure correlation.
- Action: (perform different types of actions according to different combinations that satisfy the conditions):
    1) According to the specific combinations of conditions, the NFVO determines the fault type as a VNF independent fault, and determines to send the fault notification that the fault is an independent VNF fault to the OSS/BSS.
    2) According to the specific combinations of conditions, the NFVO determines the fault type as a VM independent fault or a VNF fault caused by VM fault, and determines to send the fault notification that the fault is an independent VM fault or a VNF fault caused by a VM fault to the OSS/BSS.

NOTE 3: Whether the fault notification is sent to other functions blocks depends on which functional block executes the next policy. If the VNFM is responsible for executing the next policy, sending of notification is necessary. If the NFVO is responsible for executing the next policy, sending of notification is not necessary.

In this representation, possible events, conditions and actions associated to VNF healing policy B can be modelled as an ECA policy:

- Event: (the event that activates the trigger's action):
    1) The NFVO determines that the fault of the NS instance is an independent VNF fault and sends the notification to the OSS/BSS.
    2) The NFVO receives other specific types of VNF fault information.
- Condition: (logical combination of certain specific conditions):
    1) The performance data are reported within a period of time T3 in the recent period (data validity period).
    2) A period of time T4 has passed since the last time the policy was successfully triggered (policy cool down period).
    3) Fault information: such as VNF offline, VNF function failure, anomalies in VNF load, etc.
- Action: (perform different types of actions according to different combinations that satisfy the conditions):
    1) The NFVO determines to request VNFM to perform self-healing operations on the VNFs, such as reconnection and restart of the VNFs.

2) The NFVO determines to migrate the VNF(s) to other VMs.

In this representation, possible events, conditions and actions associated to VM healing policy C can be modelled as an ECA policy:

- Event: (the event that activates the trigger's action):
    1) The NFVO determines that the fault of the NS instance is an independent VM fault or a VNF fault caused by a VM fault.
    2) The NFVO receives other specific types of VM fault information.
- Condition: (logical combination of certain specific conditions):
    1) The performance data are reported within a period of time T5 in the recent period (data validity period).
    2) A period of time T6 has passed since the last time the policy was successfully triggered (policy cool down period).
    3) Fault information: VM usage rate anomalies, VM OS failure, VM hardware failure, etc.
- Action: (perform different types of actions according to different combinations that satisfy the conditions):
    1) The NFVO determines to perform VM healing operations, including but not limited to VM migration, rebuild, shutdown/start, etc.

5.1.3.4 An example of imperative policy paradigm

Based on the representation of VM healing policy C described in clause 5.1.3.3, a simple example of imperative VM healing policy paradigm using formal English is described in this clause:

- **WHEN** a VM Fault Notification is received.
- **IF** {Data Validity Period < 5 min **AND** Time Since Last Trigger > 10 min **AND** VM Usage Rate Fluctuation Times > 5}.
- **THEN** execute an action to restart the VM of anomaly.

5.1.4 NFV-MANO policy representation for virtualized resource optimization

5.1.4.1 Analysis description

The policy representation analysis in this clause references use case 'Optimization of NFVI-PoPs resource utilization' described in clause 5.8.3 of ETSI GR NFV-IFA 023 [i.6].

The analysis describes an informative process of the optimization policy of NFVI-PoPs resource utilization.

There are two resource utilization criteria specified in clause 5.8.3 of ETSI GR NFV-IFA 023 [i.6]. The base flow is specified too. Based on the descriptions of ETSI GR NFV-IFA 023 [i.6], an optimization policy is shown as an example to give more details.

As an example, possible events, conditions and actions associated to optimization policy of NFVI-PoPs resource utilization can be:

- Event: The NFVO receives performance information regularly reported by the VIM.
- Condition: (logical combination of certain specific conditions):
    1) Performance information is reported within a period of time T1 in the recent period (data validity period).
    2) A period of time T2 has passed since the last time the policy was successfully triggered (policy cool down period).

3) One or more VNF performance metrics (such as CPU occupancy, memory usage) have increased or decreased by a specific value or ratio.

4) The remaining capacity of the high-load NFVI-PoP is greater than the used capacity of the low-load NFVI-PoP (for the load consolidation approach).

5) The remaining capacity of the low-load NFVI-PoP is greater than a specific threshold (for the load re-balancing approach).

- Action: (perform different types of actions according to different combinations that satisfy the conditions):

  1) Determines to execute load re-balancing. Migrates a subset of VNFs on high-load NFVI-PoP to low-load NFVI-PoP. The selection of the migrated VNFs and the migration sequence can be carried out in accordance with specific pre-defined rules.

  2) Determines to execute load consolidation. Migrates all VNFs on low-load NFVI-PoP to high-load NFVI-PoP. The migration sequence of the migrated VNFs can be performed in accordance with specific pre-defined rules.

NOTE 1: The frequency of information reported by the VIM and the valid time of data determined by the NFVO can be determined based on historical data or manual experience.

NOTE 2: The selection and migration sequence of migrated VNFs can be determined based on various types of information within NFV-MANO and manual experience.

NOTE 3: The operation of migrating a VNF is neither described in the present document, nor in the referred documentation.

5.1.4.2 An Example of imperative policy paradigm

Based on the representation of virtualized resource optimization described in clause 5.1.4, a simple example of imperative VR policy paradigm using formal English is described in this clause:

- **WHEN** a type of VR performance information is received.
- **IF** {Data Validity Period < 7 min **AND** Time Since Last Trigger > 10 min **AND** Remaining Capacity of NFVI-PoP1 > Capacity In Use of NFVI-PoP2}.
- **THEN** execute load consolidation to migrate all VNFs on NFVI-PoP2 to NFVI-PoP1.

5.1.5 Policy conflict

As described in clauses 5.8.8 and 5.8.9 of ETSI GR NFV-IFA 023 [i.6], situations related to policy conflict are introduced: the two auto-scaling policies in conflict have the same policy events and conditions but different actions. This will cause NFV-MANO to be unable to perform the actions defined by the two policies together.

The conflict described above is explicit. NFV-MANO functional blocks can detect the conflict by parsing policy events, conditions and actions of the policies.

However, the policy conflicts can be more complex and implicit. The conflicts exist between policies with different events, different conditions and different actions, e.g. an implicit conflict between the VNF auto-scaling policy described in clause 5.1.2.1 and the VNF auto-haling policy B described clause 5.1.3.3.

The simplified example of the VNF auto-scaling policy is shown below:

- Event: the VNFM receives an indicator which represents the capacity available for VNF1.
- Condition: the number of CPUs of VNF1 is less than 3; the memory of VNF1 is less than 2G.
- Action: scale a single aspect of VNF1.

The simplified example of VNF auto-healing policy B is shown below:

- Event: the VNFM receives an indicator which represents the fault information of the VM of VNF1.

- Condition: the fault is the anomalies in VNF1 load.
- Action: migrate VNF1 to other VMs.

Although the two policies are different, a conflict can exist. A situation is assumed that there is a fault in a VM of VNF1. This fault can lead the capacity available to decrease and lead the fault information of the VM to be reported. If the two policies described above are both activated, the policy events of the two policies will be triggered. Then, the conditions of the two policies will be evaluated to true, resulting therefore in NFV-MANO having to execute two conflicting actions: scaling VNF1 and migrating VNF1.

In conclusion, the resolution of policy conflict should be considered in NFV-MANO. The resolutions can be achieved by different approaches such as policy model design, AI or digital twin technology.

NOTE: The solutions for policy conflict resolution is out of the scope of the present document.

5.2 Policy types involved

5.2.1 Introduction

From the description of the policy representations described in clause 5.1 of the present document and use cases listed in clause 5.8 of ETSI GR NFV-IFA 023 [i.6], NFV-MANO uses policies acting on different dimensions in its management and orchestration operations. Automated policy execution can also help reduce the involvement of O&M personnel in NFV-MANO processes.

The classifications of policy types contribute to the design of the policy model. Functional perspective is a good classification dimension, but the policy types can be classified on more dimensions, to help better design of the attributes, contents, qualifiers and other elements required by the policy model.

Clause 5.2.2 describes the classification dimensions of policy types. These dimensions and methods do not have conflict with each other and can be used in combinations to better design the NFV-MANO policy model.

5.2.2 Policy type classification

5.2.2.1 Function-based

This classification of policy types is based on functionality which the policy acts on. For example, this function-based classification method can classify the policies into the following types (the types/sub-types presented here are not exhaustive):

- Granting: policies of this type are used to assist functional blocks to determine when to accept the granting requests or not. The granting policies could be used in the granting processes involved in the lifecycle management processes. An example of this type of policies is from the use case in clause 5.8.2 of ETSI GR NFV-IFA 023 [i.6].
- Resources management: policies of this type are used to assist functional blocks to optimize the utilization of resources. Optimization aspects include but not limited to: resource utilization, resource distribution, energy, costs, etc. An example of this type of policies is from the use case in clauses 5.8.3, 5.8.12, 5.8.13 of ETSI GR NFV-IFA 023 [i.6].
- Lifecycle Management (LCM): policies of this type are used to assist functional blocks to manage lifecycle of NS instances, VNF instances and virtualized resources. Sub-types of this type of policy can include:
    - Healing: policies of this type are used to assist functional blocks to heal instances and virtualized resources. The healing aspects include but not limited to: NS instances, VNF instances, virtualized resources, etc. An example of this type of policies is from the use case in clauses 5.8.4 and 5.8.11 of ETSI GR NFV-IFA 023 [i.6].
    - Scaling: policies of this type are used to assist functional blocks to scale in/out or scale up/down the instances and virtualized resources. An example of this type of policies is from the use case in clause 5.8.5 of ETSI GR NFV-IFA 023 [i.6].

- Assurance: policies of this policy type are used to monitor the Fault and Performance metrics and in selecting policies or assurance configuration to meet the specific targets. Sub-types of this type of policy could include:
    - Root cause analysis related policies: policies of this type are used to assist functional blocks to analyse the root cause of the faults to judge the severity of the faults.

5.2.2.2 Layer-based

This classification is based on layers the policies can be used. For example, this layer-based classification method can classify the policies into the following types (the types presented here are not exhaustive):

- Network service-level: policies of this type are used to assist NS management. An example of this type of policies is from the use case in clause 5.8.4 of ETSI GR NFV-IFA 023 [i.6].
- Network function-level: policies of this type are used to assist VNF management. An example of this type of policies is from the use cases in clauses 5.8.2, 5.8.5 and 5.8.11 of ETSI GR NFV-IFA 023 [i.6].
- Resource-level: policies of this type are used to assist virtualized resource management and container management. An example of this type of policies is from the use cases in clauses 5.8.3, 5.8.12, 5.8.13 of ETSI GR NFV-IFA 023 [i.6].

There could be associations between policies of different levels. As described in clause 5.8.10 of ETSI GR NFV-IFA 023 [i.6], when an NS scaling policy is applied to a certain NS instance, multiple policies of different layers can be applied to the NS scaling workflow that can trigger other policies at network function level and virtualized resource level. Based on the scenario, there could be combinations of different policies of different policy types.

5.2.2.3 Form and delivery based

This classification of policy types is based on where the policy is defined and how it is delivered. For example, this form and delivery based classification method can classify the policies into the following types (the types presented here are not exhaustive):

- Pre-defined in NSD: policies of this type are pre-defined in the NSD and used in the LCM operations of the NS instance.
- Pre-defined in VNFD: policies of this type are pre-defined in the VNFD and used in the LCM operations of the VNF instance. An example of this type of policies is from the use case in clause 5.8.5 of ETSI GR NFV-IFA 023 [i.6].
- Policies artifacts: policies of this type are defined in independent policy artifacts and transferred using the NFV-MANO policy management framework.

5.2.2.4 Target-based

This method classifies policies based on the objects that the policy targets. For example, this target-based classification method can classify the policies into the following types (the types presented here are not exhaustive):

- Common: policies of this type target any object of a kind. Policies of this type will not be associated to specific instances of NS, VNF or virtualized resources in NFV-MANO.
- Specific: policies of this type target a specific object. Policies of this type will be associated to specific instances of NS, VNF or virtualized resources in NFV-MANO. For example, a scaling policy of specific policy type could only be applied to VNF instances that are instantiated by NSD A and NSD B. VNF instances instantiated by other NSDs will not apply this policy even though the policy events and conditions are satisfied.

5.3 Context-aware policies

Currently, context-aware policies are not applied in NFV-MANO. However, there exists a relationship between policy and context.

For example, policy representations described in clause 5.1 evaluate the cool down time in policy conditions. The cool down time can be considered as a context item which is used to avoid the policies being triggered repeatedly.

It is assumed that there are two VNF auto-scaling policies A and B which are only applied to VNF1. There is a possible situation that policy A and policy B use the same cool down time T. Based on the above assumptions, some issues related to cool down time are introduced:

- Can other NFV-MANO policies besides policy A and B also read this cool down time T? (readable).
- Whether both policy A and B can modify cool down time T, or only one of them can modify it? (writable).

To resolve the issues, context-awareness is considered by the policy model to describe the cool down time T and other correlated context items. For example, how to reference context items into policy events, conditions and actions, how to define the readability and writability of context items?

Another important aspect for context-awareness, is that the policy is expected to also be able to process contextual information on the policy execution itself with relevant events, conditions and actions in the policy which can also be used to control the execution of the policy. For instance, the policy expression can formulate ways to check the status of the execution of the policy such as whether some failure happens during its execution, or whether the execution of the policy is running over a defined limit, or whether some condition checking fails, etc. and with such information then the policy can decide whether to continue or cancel its execution depending on the characteristic of the policy.

NOTE: Context-awareness can be used in other automation aspects of NFV-MANO, such as automatic control loop, intent management. More studies are needed in the future.

5.4 Recommendations for policy model requirements

5.4.1 General recommendations

Table 5.4.1-1 provides the general recommendations for NFV-MANO policy model, according to clause 5.1 NFV-MANO policy representations and clause 5.3 context-aware policies.

Table 5.4.1-1: General recommendations for NFV-MANO policy model

Rec Number	Recommendation Description	Comments
POLICY.MODEL. GEN.001	It is recommended that a requirement be specified for the NFV-MANO policy model to support the imperative policy paradigm.	See note.
POLICY.MODEL. GEN.002	It is recommended that a requirement be specified for the NFV_MANO policy model to support the ECA policy expression form.	See note.
POLICY.MODEL. GEN.003	It is recommended that a requirement be specified for the NFV_MANO policy model to support a policy to include one or more policy events, policy conditions, policy actions to form a complete policy.	See note. Attributes, relationships, methods and constraints are defined in the model to create relations among the policy events, conditions, actions and other necessary objects to form a complete policy.
POLICY.MODEL. GEN.004	It is recommended that a requirement be specified for the NFV_MANO policy model to specify the mapping relationships among policy events, conditions and actions to allow different combinations of policy events, conditions and actions to achieve the goals of the policy.	See note.
POLICY.MODEL. GEN.005	It is recommended that a requirement be specified for the NFV-MANO policy model to support context-aware policies.	Derived from clause 5.3. In NFV-MANO, the context is dynamic. Therefore, it is necessary to change the active set of policies dynamically to follow the changes of context. In addition, context-awareness of the policy can relate to the actual execution of the policy.

Rec Number	Recommendation Description	Comments
POLICY.MODEL. GEN.006	It is recommended that a requirement be specified for the NFV-MANO policy model to align with semantics and structures of the NFV Information Model and be capable of using the specified NFV design and runtime information and interfaces.	The policy information model of NFV-MANO should not be defined separately and thus is expected to leverage the semantics and requirements expressed in ETSI NFV-IFA documentation, such as being able to process runtime information of VNF (e.g. configuration of the VNF instance from VnfInfo), information collected via notifications and events, and make use of specified interface operations.
NOTE: Derived from all representations in clause 5.1 of the present document and use cases in clause 5.8 of ETSI GR NFV-IFA 023 [i.6].		

5.4.2 Recommendations for policy model information elements

Table 5.4.2-1 provides the general recommendations for NFV-MANO policy model information elements.

Table 5.4.2-1: General recommendations for NFV-MANO policy model information elements

Rec Number	Recommendation Description	Comments
POLICY.MODEL. INFO.001	It is recommended that a requirement on information elements be specified for the NFV-MANO policy model to support the imperative policy paradigm.	See note.
POLICY.MODEL. INFO.002	It is recommended that a requirement on information elements be specified for the NFV-MANO policy model to support the ECA policy expression form.	See note.
POLICY.MODEL. INFO.003	It is recommended that a requirement on information elements be specified for the NFV-MANO policy model to support a policy to include one or more policy events, policy conditions, policy actions to form a complete policy.	See note. Attributes, relationships, methods and constraints are defined in the model to create relations among the policy events, conditions, actions and other necessary objects to form a complete policy.
POLICY.MODEL. INFO.004	It is recommended that a requirement on information elements be specified for the NFV-MANO policy model to specify the mapping relationships among policy events, conditions and actions to allow different combinations of policy events, conditions and actions to achieve the goals of the policy.	See note.
POLICY.MODEL. INFO.005	It is recommended that a requirement on information elements be specified for the NFV-MANO policy model to support context-aware policies.	Derived from clause 5.3. In NFV-MANO, the context is dynamic. Therefore, it is necessary to change the active set of policies dynamically to follow the changes of context. In addition, context-awareness of the policy can relate to the actual execution of the policy.

Rec Number	Recommendation Description	Comments
POLICY.MODEL. INFO.006	It is recommended that a requirement on information elements be specified for the NFV-MANO policy model to align with semantics and structures of the NFV Information Model and be capable of using the specified NFV design and runtime information and interfaces.	The policy information model of NFV-MANO should not be defined separately and thus is expected to leverage the semantics and requirements expressed in ETSI NFV-IFA documentation, such as being able to process runtime information of VNF (e.g. configuration of the VNF instance from VnfInfo), information collected via notifications and events, and make use of specified interface operations.
NOTE: Derived from all representations in clause 5.1 of the present document and use cases in clause 5.8 of ETSI GR NFV-IFA 023 [i.6].		

5.4.3 Recommendations for policy model attributes

Table 5.4.3-1 provides the recommendations for potential requirements related to NFV-MANO policy model attributes.

Table 5.4.3-1: Recommendations for NFV-MANO policy model attributes

Rec Number	Recommendation Description	Comments
POLICY.MODEL. ATTR.001	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) to support the types of triggers the policy events in a policy.	Derived from representations in clause 5.1. Triggers could be various types of notifications, various types of data received from functional blocks, an explicit trigger by another policy, etc.
POLICY.MODEL. ATTR.002	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) to support the description of source of information the policy events and conditions applicable to the policy.	Derived from representations in clause 5.1. For example, there is an NS healing policy in NFVO. A condition in it is used to check the status of a VNF instance of this NS instance. To guide the NFVO to identify the indicators needed, this recommended attribute could be used as a VNF ID or the name of the indicator. Therefore, the VNFM could send back the indicator based on this attribute.
POLICY.MODEL. ATTR.003	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) to support the description of the relationships between policy events or policy conditions, such as and, or, xor, etc.	Derived from representations in clause 5.1. This recommendation for a requirement allows a subset of events to trigger the evaluation of policy conditions.
POLICY.MODEL. ATTR.004	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) needed by the statement form used by the policy conditions and policy actions.	Derived from representations in clause 5.1. This recommendation for a requirement is an extension of POLICY.MODEL.INFO.004. The object(s) should have these attribute(s).

Rec Number	Recommendation Description	Comments
POLICY.MODEL. ATTR.005	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) to support defining the execution sequence, the context of the actual execution, and strategies of the policy actions.	Derived from representations in clause 5.1. There could be many policy actions. Therefore, there should be rules to specify how to execute these actions. For example, the actions could be executed based on the pre-defined sequence until a failure occurs as information about the execution of the policy. The strategies could be defined and chosen according to the actual situations and the execution of the policy.
POLICY.MODEL. ATTR.006	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) to support the description of the mapping between the combinations of the policy actions and the combinations of the policy conditions.	Derived from representations in clause 5.1. This recommendation for a requirement allows a subset of conditions to trigger the execution of a specific subset of actions. It also allows for mapping in between the actions performed between different policies. This recommendation for a requirement is an extension of POLICY.MODEL.GEN.004. The information elements(s) should have these attribute(s).
POLICY.MODEL. ATTR.007	It is recommended that a requirement be specified for the NFV-MANO policy model attribute(s) to support intrinsic functions (such as wait, loop, date) for controlling the execution of the policy and perform operations on expressions related to events, conditions and actions.	Derived from representations in clause 5.1. This recommendation for a requirement allows a policy expression to contain intrinsic functions such as "wait", "loop", "date", "calculate" (basic arithmetic operations addition, subtraction, etc.) to help control the execution of the policy.

5.4.4 Other recommendations related to policy management

Table 5.4.4-1 provides recommendations for potential requirements related to NFV-MANO policy management.

Table 5.4.4-1: Recommendations for NFV-MANO policy management

Rec Number	Recommendation Description	Comments
POLICY.MODEL. OTH.001	It is recommended that a requirement be specified for the NFV-MANO policy management to indicate whether the policy is in its cool down period or not.	Derived from representations in clause 5.1. If the policy is in its cool down period, the policy should not be triggered. This recommendation concerns to potential enhancement of policy management interfaces as defined in ETSI GS NFV-IFA 013 [i.13] and stage 3 equivalent ETSI GS NFV-SOL012 [i.7].

5.4.5 Examples: potential attributes of the VNF auto-scaling policy

Based on recommendations from clause 5.4.3 and representations described in clause 5.1.1 of the present document, some potential policy model attributes are listed in table 5.4.5-1 as examples.

Table 5.4.5-1: Examples of VNF Auto-scaling policy attributes

Policy Target	Examples for events, conditions and actions	Reference to Recommendations
VNF auto-scaling	Types of events, such as alarm notifications, VNF resources performance (such as CPU occupancy, memory usage), etc.	POLICY.MODEL.ATTR.002/003.
VNF auto-scaling	Conditions such as VNF resources performance (CPU occupancy, memory usage, etc.), the "autoScalable" attribute of the "VnfConfigurableProperties", etc.	POLICY.MODEL.ATTR.005.
VNF auto-scaling	Actions such as the approaches to scale. For example, an approach could scale a single aspect of a VNF. Another approach could scale multiple aspects of a VNF simultaneously.	POLICY.MODEL.ATTR.005.

5.4.6 Examples: potential attributes of the NS self-healing policy and NS root cause analysis policy

Based on recommendations from clause 5.4.3 and representations described in clause 5.1.2 of the present document, some potential policy model attributes are listed in table 5.4.6-1 as examples.

Table 5.4.6-1: Examples of NS self-healing policy attributes

Policy Target	Examples for events, conditions and actions	Reference to Recommendations
NS Self-healing	Events to support the definition of the types of triggers needed by policy events, such as types of fault notifications, types of faults data, an explicit trigger by another policy, etc.	POLICY.MODEL.ATTR.002/003.
NS Self-healing	Conditions to support the definition of the variables. For fault sources, the variables could be only from VIM, only from VNFM, from VIM and VNFM both, etc. For faults details, the variables could be VM hardware faults, CPU occupancy rate fluctuation, VNF offline, etc. For faults correlations, the variables could be A's fault caused by B's fault, an independent fault, etc.	POLICY.MODEL.ATTR.003/003/004.
NS root cause analysis policy	Actions to support the definition of fault notifications sent from NFVO to OSS/BSS.	POLICY.MODEL.ATTR.005.
NS Self-healing	Actions to support the execution of the healing actions, such as re-connection of constituents in the NS, re-start of the VNF/VM, re-allocation of virtualized resources, an explicit trigger for another policy, etc.	POLICY.MODEL.ATTR.005/006.

5.4.7 Examples: potential attributes of the virtualized resource optimization policy

Based on recommendations from clause 5.4.3 and representations described in clause 5.1.3 of the present document, some potential policy model attributes are listed in table 5.4.7-1 as examples.

Table 5.4.7-1: Examples of virtualized resource optimization policy attributes

Policy Target	Examples for events, conditions and actions	Reference to Recommendations
Virtualized resource optimization	Events to support the definition of the types of triggers needed by policy events, such as VNF performance metrics, remaining capacity of the NFVI-PoPs, etc.	POLICY.MODEL.ATTR.003/004.
Virtualized resource optimization	Conditions to support the judgement of the sub-optimal NFVI-PoPs resource utilization, such as specific VNF performance metrics and their values, NS performance metrics and their values, etc.	POLICY.MODEL.ATTR.004/005/006.
Virtualized resource optimization	Actions to support the definition of the actions could be used to handle sub-optimal scenarios.	POLICY.MODEL.ATTR.006.
Virtualized resource optimization	Actions to support the selection and migration sequence of migrated VNFs.	POLICY.MODEL.ATTR.006/007.

6 Analysis of existing SDOs and open source community policy models

6.1 TMF policy model

6.1.1 Introduction to TMF policy model

TMF Information Framework (SID) Frameworx (also known as NGOSS) specification defines a structural framework that can be used to guide the development of distributed computing solutions, specifically by creating and reusing Frameworx artifacts.

According to TMF GB922 Frameworx Standard Information Common Business Entities-Policy [i.1], Frameworx contains four views, namely system view, implementation view, operational view and business view. TMF believes that policies can play an important role in all four views of Frameworx, that is, policies can be used to represent, define and manage all types of entities. In order to implement these functions of the policy and integrate the policy into the architecture, SID Frameworx defines the policy as a new Common Business Entities ABE(Aggregate Business Entity) to contain the information model of the policy.

As shown in figure 6.1.1-1 (cited from [i.1]), in order to express a policy, Policy ABE defines 4 core entity groups, namely Policy, Policy Specification, Policy Application and Policy Management.

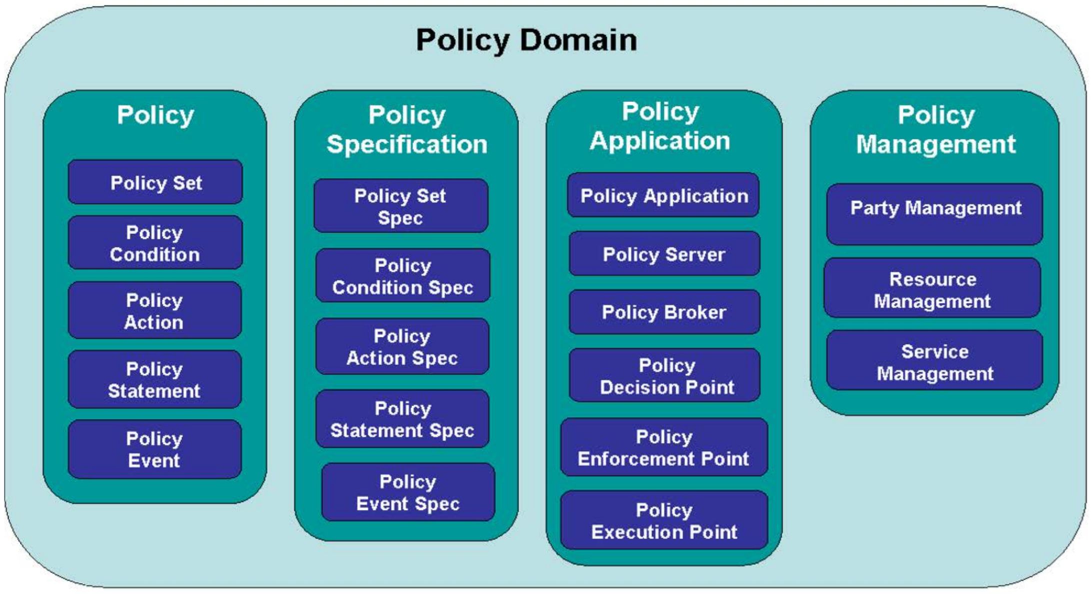

Figure 6.1.1-1: Policy Domain Level 1 and Level 2 ABEs

Policy Specification ABE provides the ability to create policy templates, and Policy ABE is used to represent instances of these templates. Policy Application and Policy Management ABEs use the capabilities of the above two level 1 ABEs to provide a standardized representation of policies.

In conclusion, the main basis for the design of the policy information model is Policy ABE and Policy Specification ABE. Clause 6.1.2 will mainly focus on the details of the TMF SID policy information model, and each class contained in the information model is determined by Policy ABE and Policy Specification ABE.

More information can be obtained from TMF GB922 Frameworx Standard Information Common Business Entities-Policy [i.1] and TMF TR234 [i.2].

6.1.2 Overview of TMF information model

6.1.2.1 TMF SID policy information model

6.1.2.1.0 Introduction

In 6.1.2.1 sub-clauses, details of TMF SID policy information model are presented. The classes and relationships introductions and analysis shown hereafter are based on TMF GB922 [i.1]. For more information details, please refer to [i.1].

6.1.2.1.1 TMF SID policy information model overview

As shown in figure 6.1.2.1.1-1, the TMF SID policy information model is composed of Level 1 Policy/Specification ABEs and Level 2 ABEs contained.

Figure 6.1.2.1.1-1 shows the overview of the TMF SID policy information model. This figure takes the class PolicyRuleBase as the center and describes the classes included in the Policy Event, Policy Condition, Policy Action and the relationships between these classes.

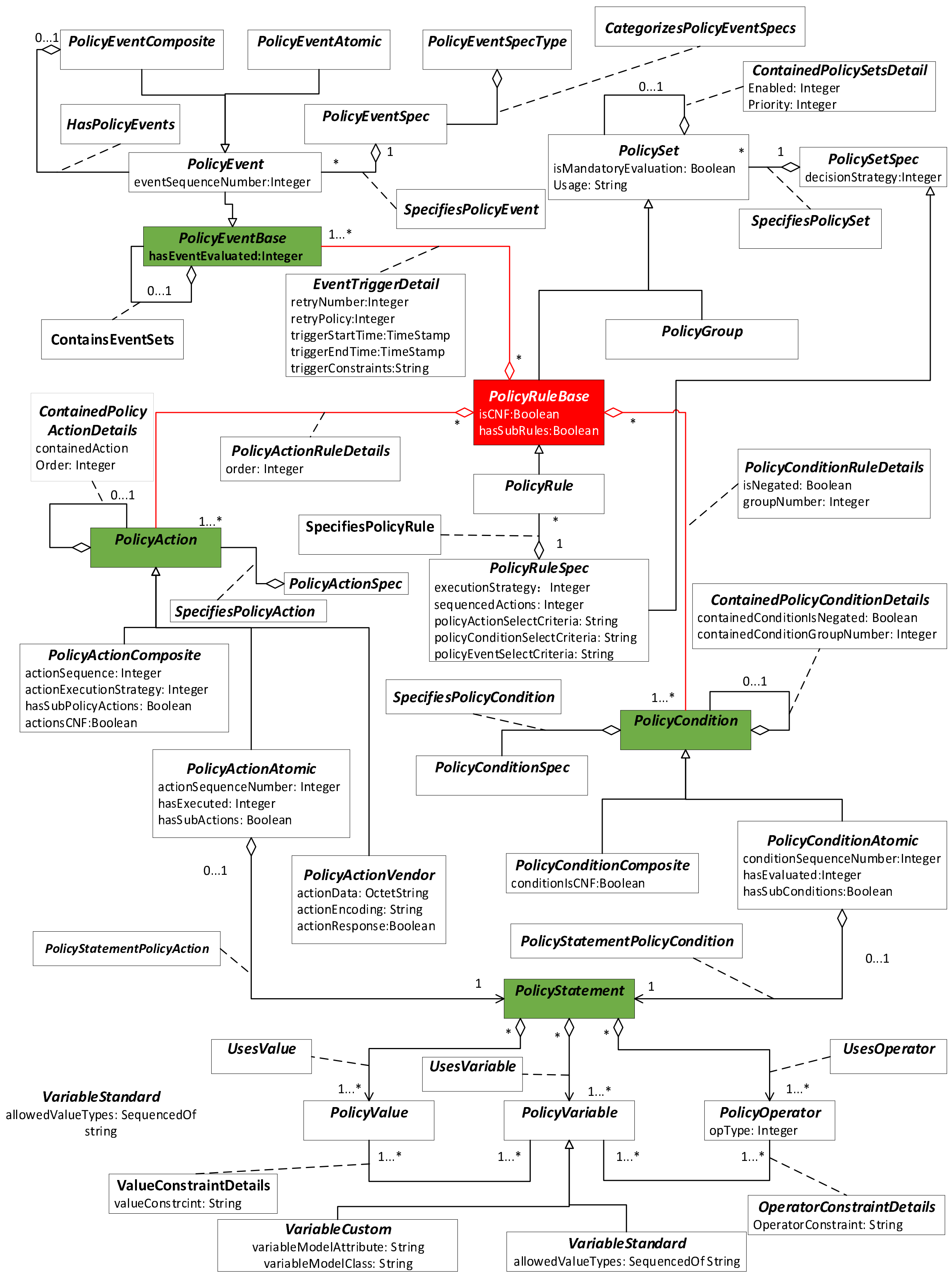

Figure 6.1.2.1.1-1: TMF SID Policy Information Model Overview

TMF SID policy information model supports the use of ECA imperative policy format. In figure 6.1.2.1.1-1, PolicyRuleBase is the center since the most significant feature of ECA policy is that a policy rule consists of three modules (Policy Event, Policy Condition and Policy Action).

In addition to the above four classes and relationships, this policy information model also considers more details:

- Use the PolicyGroup class as a container to allow multiple PolicyRule classes and PolicyGroup classes to be aggregated in the same container, so that multiple policy decisions/actions can be executed together. PolicyRule and PolicyGroup have common attributes, so PolicySet is introduced as the superclass of these two classes. This design method allows PolicyGroup and PolicyRule to share the same semantics, and also allows PolicyGroup and PolicyRule to nest themselves and each other, which more satisfies implementation considerations.

- There are some same features used in design patterns. Standard composite pattern is not used in PolicySet and PolicyGroup/PolicyRuleBase, PolicyCondition and PolicyConditionComposite/PolicyConditionAtomic. PolicySet, PolicyCondition and PolicyAction use self-aggregation to indicate the characteristics of self-nesting and mutual nesting between classes.

- According to the characteristics of the format, some variables, some values, and some operators of PolicyCondition and PolicyAction in the specific implementation process, the TMF SID policy information model aggregates the PolicyStatement class into the above two classes, which will allow TMF Policy Decision Point (PDP), TMF Policy Enforcement Point (PEP), TMF Policy Execution Point (PXP) which use these classes to share the same basic policy semantics and syntax.

6.1.2.1.2 PolicyRuleBase and related classes

As shown in figure 6.1.2.1.1-1, PolicyRuleBase is an intelligent container and has an association with PolicyEvent, PolicyCondition, and PolicyAction. This relationship and the corresponding cardinality indicate that PolicyRule should contain at least these three classes before it can be instantiated, and these three classes can exist and be stored in Policy Repository independently. This kind of design method provides a foundation for the reuse of policy components.

PolicyRuleBase contains two attributes, isCNF is used to describe the format of PolicyConditions, and hasSubRules is used to indicate whether there is a nested relationship.

PolicyRuleSpec provides specifications for the construction and execution of PolicyRule, that is, the excecutionStrategy and sequencedActions in PolicyRuleSpec guide the implementation of the policy; policyActionSelectCriteria, policyConditionSelectCriteria, and policyEventSelectCriteria provide guidance for the selection of the classes associated with PolicyRule.

6.1.2.1.3 PolicySet and related classes

PolicySet is the superclass of PolicyRuleBase and PolicyGroup. It is used to generalize PolicyRules and PolicyGroups. It is also a collection class like PolicyRuleBase and PolicyGroup. This also shows that they can form nested relationships, but they do not need to use the standard composite pattern.

PolicyGroup is a generalized aggregation container, which allows multiple aggregated PolicyRules to be conducted at the same time.

PolicySetSpec is to define the attributes, methods, relationships, constraints, etc. that different PolicySets can share together, aiming to match the needs of different applications. PolicySetSpec currently defines the attribute decisionStrategy, which aims to define the evaluation order of PolicyRuels.

6.1.2.1.4 PolicyEvent and related classes

PolicyEvents is a base class used to trigger the evaluation of PolicyRules. PolicyEvent uses composite pattern to establish PolicyEventAtomic and PolicyEventComposite.

From system viewpoint, HasPolicyEvents is a class that enables external applications, such as policy servers, to dynamically adjust event sets. EventTriggerDetails is an association class used to determine the semantics and set of events that can be used by PolicyRules.

PolicyEventSpec is an abstract base class, which defines stand-alone and aggregated PolicyEventSpecs using the composite pattern. The specific relationship is shown in figure 6.1.2.1.4-1.

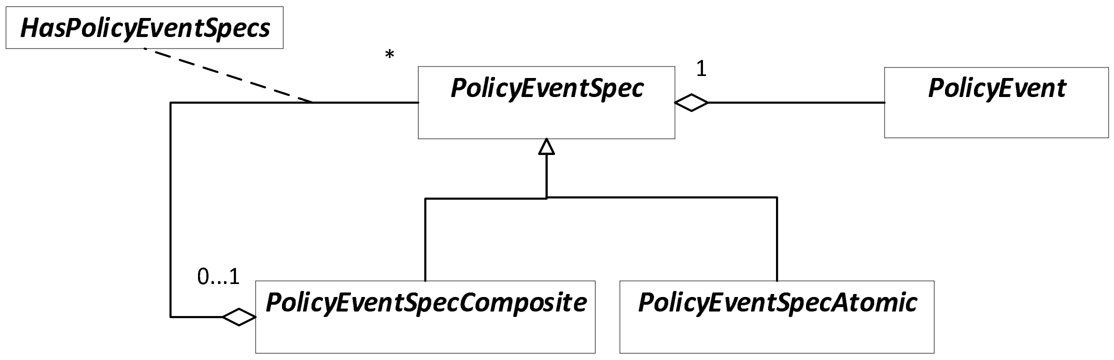

Figure 6.1.2.1.4-1: PolicyEventSpecifications

Considering that there are multiple types of PolicyEvents, the TMF SID policy information model uses PolicyEventSpecType to define the generic category of PolicyEventSpecs. This method allows PolicyEventSpecs with the same behaviour and other semantics to be grouped. Related information is shown in figure 6.1.2.1.1-1.

6.1.2.1.5 PolicyCondition and related classes

PolicyCondition is an abstract class used to represent rule-specific or reusable policy conditions. PolicyCondition does not explicitly use the composite pattern, but consists of two subclasses, PolicyConditionComposite and PolicyConditionAtomic. This design method has many characteristics:

- Enable rich semantics.
- PolicyConditionAtomic is the base class of all simple policy conditions. A simple Policy Condition consists of a single Boolean clause, which is used to test a single condition.
- PolicyConditionComposite is the base class of all complex policy conditions. It is used to contain one or more PolicyConditons at a higher level. It is designed to deal with more complex relationships between PolicyConditions, such as nested or a group of subordinate.

PolicyConditionSpec is a concrete base class used to define invariant characteristics (attributes, methods, relationships, constraints, etc.). Currently PolicyConditionSpec does not define any attributes.

6.1.2.1.6 PolicyAction and related classes

PolicyAction is an abstract class used to express how to compose the action clauses of PolicyRule. Same as the design of PolicyCondition, PolicyAction does not explicitly use the combined mode, but is composed of three subclasses, namely PolicyActionComposite, PolicyActionAtomic, and PolicyActionVendor.

This design method has many characteristics:

- Enable rich semantics.
- PolicyActionAtomic is the base class of all simple policy conditions. A simple PolicyAction consists of a single Boolean clause, which is used to perform a single action.
- PolicyActionComposite is the base class of all complex policy conditions, used to form more coarse action structures.
- PolicyActionVendor is a concrete base class used to define the content and format of vendor-specific actions. It is not recommended to consider this class in standardization work.

PolicyActionSpec is a concrete base class used to define invariant characteristics (attributes, methods, relationships, constraints, etc.). Currently PolicyConActionSpec does not define any attributes.

6.1.2.1.7 PolicyStatement and related classes

As shown in figure 6.1.2.1.1-1, PolicyConditionComposite and PolicyActionComposite have an aggregation relationship with PolicyStatement. The main reason is that the format of PolicyCondition and PolicyAction is {variable, operator, value}, and the main purpose of PolicyStatement is to enable a model to summarize a standardization PolicyCondition and PolicyAction format.

According to the common format {variable, operator, value}, PolicyStatement aggregates three subcategories:

- PolicyVariables: an abstract class used to model different types of variables. This abstract class is associated with VariableCustom and VariableStandard, which are used to define custom and standardized variable sets. The association classes VariableCustom and VariableStandard are shown in figure 6.1.2.1.1-1.
- VariableStandard: an abstract base class for defining a standard set of PolicyVariable objects that are common to most applications. This is the superclass for a standard set of PolicyVariable subclasses that are part of the model.(This concept is cited from [i.1]).
- VariableCustom: an abstract base class that defines a set of standardized policy variables for use in an application-specific manner. The term "custom" means that such variables are explicitly designed to work with attributes that are not in any of the VariableStandard subclasses. This means that the particular semantics (including any applicable constraints) are not known to SID model. (This concept is cited from [i.1], which can be referenced for more information).
- PolicyValues: an abstract base class used to model different types of values. This abstract class is associated with ValueCustom and ValueStandard, which are used to define custom and standardized value sets.
- PolicyOperators: a concrete class used to model different types of operators.

PolicyStatementSpec is a concrete base class used to define invariant characteristics (attributes, methods, relationships, constraints, etc.). Currently, PolicyStatementSpec does not define any attributes.

6.1.3 Comparison of recommendations with TMF SID policy model

6.1.3.1 Introduction

In this clause, the recommendations for policy model requirements which are described in clause 5.4 are compared with the TMF SID policy model. The results include 'Fully meets', 'Partially meets' and 'Does not meet':

- Fully meets: the recommendation could be satisfied by the TMF SID policy model.
- Partially meets: part of the recommendation could be satisfied by the TMF SID policy model or this recommendation is under the consideration but needs more enhancements.
- Does not meet: the recommendation is not considered by the TMF SID policy model at all.

6.1.3.2 Comparison with general recommendations for policy model

Table 6.1.3.2-1 provides the comparison results of the TMF SID policy model and the general recommendations listed in clause 5.4.1 of the present document.

Table 6.1.3.2-1: Comparison results of general recommendations.

Rec Number	Status	Comments
POLICY.MODEL. GEN.001	Fully meets	The TMF SID policy information model is explicitly used to support imperative policy paradigm. It has class PolicyStatement referenced by PolicyCondition and PolicyAction to represent an imperative policy paradigm.
POLICY.MODEL. GEN.002	Fully meets	There are classes PolicyEvent, PolicyCondition, PolicyAction and related classes to represent the constituents of an ECA policy expression form.
POLICY.MODEL. GEN.003	Fully meets	The class PolicyEvent use composite pattern to allow several atomic policy events to form an event of a complete policy. The classes PolicyCondition and PolicyAction follow the same pattern to fully meet this recommendation.
POLICY.MODEL. GEN.004	Fully meets	There are PolicyRuleSpec, PolicyEventSpec, PolicyConditionSpec, PolicyActionSpec to define the relationships between policy events, policy conditions, policy actions.
POLICY.MODEL. GEN.005	Does not meet	The TMF SID policy information model does not support the concept of context.

6.1.3.3 Comparison with recommendations for policy model information elements

Table 6.1.3.3-1 provides the comparison results of the TMF SID policy model and the recommendations for policy model information elements listed in clause 5.4.2 of the present document.

Table 6.1.3.3-1: Comparison results of recommendations for policy model information elements

Rec Number	Status	Comments
POLICY.MODEL. INFO.001	Fully meets	There are classes PolicyEvent, PolicyCondition, PolicyAction and other related classes to support this recommendation.
POLICY.MODEL. INFO.002	Fully meets	There are classes PolicyRuleBase, PolicySet, PolicyGroup and other related classes to support this recommendation.
POLICY.MODEL. INFO.003	Fully meets	There are specific classes to provide specifications to define the attributes, relationships, methods and constrains, such as PolicyEventSpec, PolicyConditionSpec, PolicyActionSpec, PolicyRuleSpec, PolicySetSpec.
POLICY.MODEL. INFO.004	Fully meets	There are classes PolicyStatement, PolicyValue, PolicyVariable, PolicyOperator to define the policy statement form.
POLICY.MODEL. CLASS.005	Does not meet	There is no class related to context.

6.1.3.4 Comparison with recommendations for policy model attributes

Table 6.1.3.4-1 provides the comparison results of the TMF SID policy model and the recommendations for policy model attributes listed in clause 5.4.3 of the present document.

Table 6.1.3.4-1: Comparison results of recommendations for policy model attributes

Number	Status	Comments
POLICY.MODEL. ATTR.001	Partially meets	There are classes PolicyEventSpecType and PolicyEventSpec to define attributes of a policy event. Therefore, the types of triggers attributes could be added in one of these classes. However, attribute(s) should be added to the classes above to satisfy this recommendation.
POLICY.MODEL. ATTR.002	Partially meets	There are classes PolicyEventSpecType and PolicyEventSpec to define attributes of a policy event. There are also similar classes to define attributes of a policy event. However, attribute(s) should be added to the classes above to satisfy this recommendation.
POLICY.MODEL. ATTR.003	Partially meets	There is an attribute hasSubCondtions in class PolicyConditionAtomic to represent if there are sub-conditions of a policy condition. There is an attribute conditionSequeceNumber in class PolicyConditionAtomic to represent the sequence of parsing. However, attribute(s) should be added to the classes above to satisfy this recommendation.

Number	Status	Comments
POLICY.MODEL.ATTR.004	Partially meets	There are classes PolicyValue, PolicyVariable and PolicyOperator to represent the policy statement form. However, attribute(s) should be added to the class above to satisfy this recommendation.
POLICY.MODEL.ATTR.005	Fully meets	There are attributes executionStrategy, sequecedActions, policyEventSelectCriteria, policyActionSelectCriteria and policyConditionSelectCriteria in class PolicyRuleSpec to support this recommendation.
POLICY.MODEL.ATTR.006	Does not meet	There is no attribute to support this recommendation.
POLICY.MODEL.ATTR.007	Partially meets	There are classes PolicyCondition, PolicyAction and other related classes that could be used to support this recommendation. However, attribute(s) should be added to the classes above to satisfy this recommendation.

6.1.4 Features comparison and gaps analysis

TMF GB922 Frameworx Standard Information Common Business Entities-Policy [i.1] defines an information model specific to SID Frameworx. The classes and relationships between different classes and design patterns in this information model have some unique features since this model is customized for SID Frameworx. However, these features and characteristics have the possibility to be used for reference to design policy models specific to NFV-MANO. In this clause, the features derived from [i.1] will be listed as a summary. Moreover, gap analysis will also be listed based on the analysis in clause 6.1.3.

Features of TMF SID policy model are as follows:

1) Support imperative policy paradigm directly, and ECA format are supported only.

2) Use a common statement method for unified modelling of conditions and actions, which may limit flexibility while providing reusability.

3) The Composite pattern is adopted to provide a common structure for a given object, especially for the design of Level2 Policy Specification ABE.

4) The Entity-EntitySpecification pattern is adopted to separate the characteristics and behaviours of common and invariant from multiple characteristics and behaviours. For example, the aggregation relationship design of PolicySet/PolicySetSpec, PolicyRule/PolicyRuleSpec use this design pattern.

Gap analysis based on comparison results in clause 6.1.3 are as follows:

1) Most of the general recommendations described in clause 5.4.1 are met. However, this policy model does not support the concept of context. It can cause the policy could not be context-aware policy. It could be a limitation to constrain the policy to make dynamic adaptions based on changes of context.

2) Most of the recommendations for policy model information elements described in clause 5.4.2 are met. However, this policy model does not have information elements related to context. Feasibility to extend this policy model to support context should be analysed and more corresponding complex work are expected since context has relationships with many information elements.

3) Most of the recommendations for policy model attributes described in clause 5.4.3 are met or considered. Since this policy model support ECA form directly, attributes related to policy events, conditions and actions are fully met or partially met. However, there is no attribute to support the description of the mapping between policy items. It can cause low flexibility to constrain policy designers to design a complex policy and lead to a large number of policies to be designed in order to finish a complex policy. This gap could increase the burden on the repository.

6.2 IETF policy model

6.2.1 Introduction to IETF policy models

IETF SUPA (Simplified Use of Policy Abstractions) working group defines a Generic Policy Information Model (GPIM) [i.4] for representing different types of policies. According to IETF RFC 8328 [i.5], GPIM uses a common extensible framework that is independent of language, protocol, repository and the level of abstraction of the content and meaning of a policy. This enables a common set of concepts defined in this information model to be mapped into different representations of policy (e.g. imperative, declarative). GPIM also supports different data models using different languages, protocols and repositories to optimize its usage The SUPA ECA Policy Rule Information Model (EPRIM) [i.4] extends the GPIM to represent policy rules that use the Event-Condition-Action (ECA) paradigm.

6.2.2 Overview of IETF information models

6.2.2.1 GPIM: Generic Policy Information Model

As illustrated in figure 6.2.2.1-1,GPIM defines the following concepts as the common elements independent of police type. GPIM provides a generic extensible framework as the common basis for inclusion of extensions for differenttypes of policies can be extended. Generic policy information model is proposed in [i.4] which can be referenced for more information.

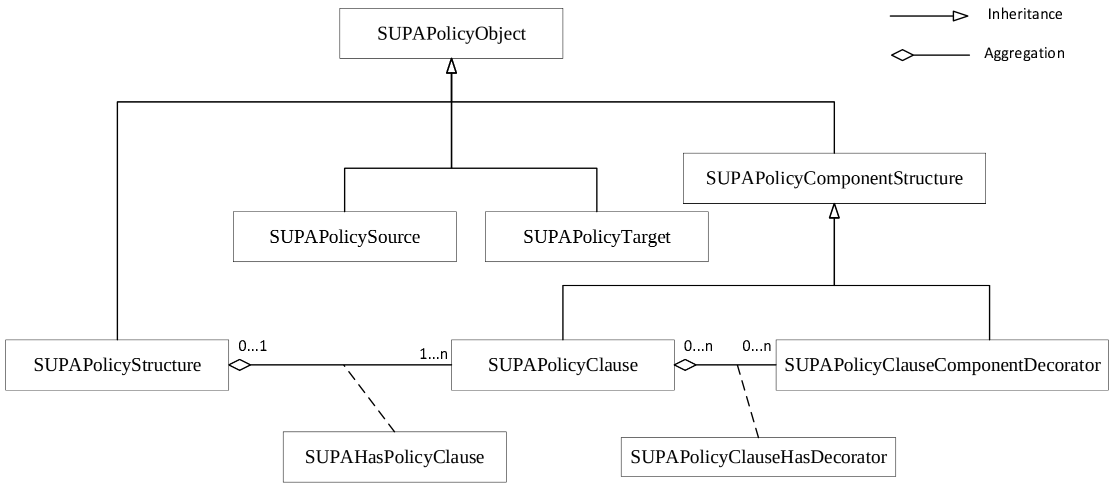

Figure 6.2.2.1-1: Functional View of the Top-Level GPIM

SUPAPolicyObject: A SUPAPolicyObject serves as a single root of the SUPA system except for the metadata objects. This simplifies the code generation and reusability. The subclasses of SUPAPolicyObject, SUPAPolicyTarget and SUPAPolicySource, are common structures for different types of policies. In order to enable GPIM to be flexibly extended to support different types of policies, SUPAPolicyStructure and SUPAPolicyComponentStructure are defined as abstract classes.

SUPAPolicyTarget: This class defines a set of managed entities that a SUPA policy is applied to.

SUPAPolicySource: This class defines a set of managed entities that authored or are otherwise responsible for this SUPA policy.

SUPAPolicyStructure: SUPAPolicyStructure is an abstract superclass that is the base class for defining different types of policies. The SUPA policy is defined as a policy container. The policy container specifies the structure, content and resources, targets and metadata of the SUPA policy (optional). This is implemented by a subclass of SUPAPolicyStructure. For example, the SUPAECAPolicyRule class in the EPRIM .

SUPAPolicyComponentStructure: SUPAPolicyComponentStructure is an abstract superclass for defining components of different types of policies. SUPAPolicyStructure subclasses define the structure of a policy, while SUPAPolicyComponentStructure subclasses define the content. For example, the events, conditions and action clauses contained in SUPAECAPolicyRule are filled in by the corresponding subclasses of SUPAPolicyComponentStructure.

SUPAPolicyClause: All policies derived from the GPIM are made up of one or more SUPAPolicyClauses, which define the content of the policy. SUPAPolicyClause is an abstract class and serves as a convenient aggregation point for assembling other objects that make up a SUPAPolicyClause.

SUPAPolicyClauseComponentDecorator: SUPAPolicyClauses are constructed using the decorator pattern. This is a design pattern that enables behavior to be selectively added to an individual object. The decorator pattern uses composition, instead of inheritance, to avoid class and relationship explosion. The subclass of SUPAPolicyClauseComponentDecorator can be used to form a SUPAPolicyClause.

In GPIM, the association relationship between classes is also defined as a class, called association class, which is used to describe the dependency information between classes. Aggregation is a kind of strong association, which represents a whole-part dependency relationship.

SUPAHasPolicyClause: The association relationship between SUPAPolicyStructure and SUPAPolicyClause is defined by SUPAHasPolicyClause, which is used to manage which set of SUPAPolicyClauses can be aggregated by which set of SUPAPolicyStructure.

SUPAPolicyClauseHasDecorator: SUPAPolicyClauses are constructed using the decorator pattern. The association relationship between SUPAPolicyClause and SUPAPolicyClauseComponentDecorator is defined by SUPAPolicyClauseHasDecorator class. SUPAPolicyClauseComponentDecorator is used to add behavior to SUPAPolicyClause.

6.2.2.2 EPRIM: ECA Policy Rule Information Model

As shown in figure 6.2.2.2-1, SUPA ECA Policy Rule Information Model (EPRIM) extends the GPIM to represent Event-Condition-Action (ECA) policy rule. ECA policy rule information model is proposed in [i.4] which can be referenced for more information.

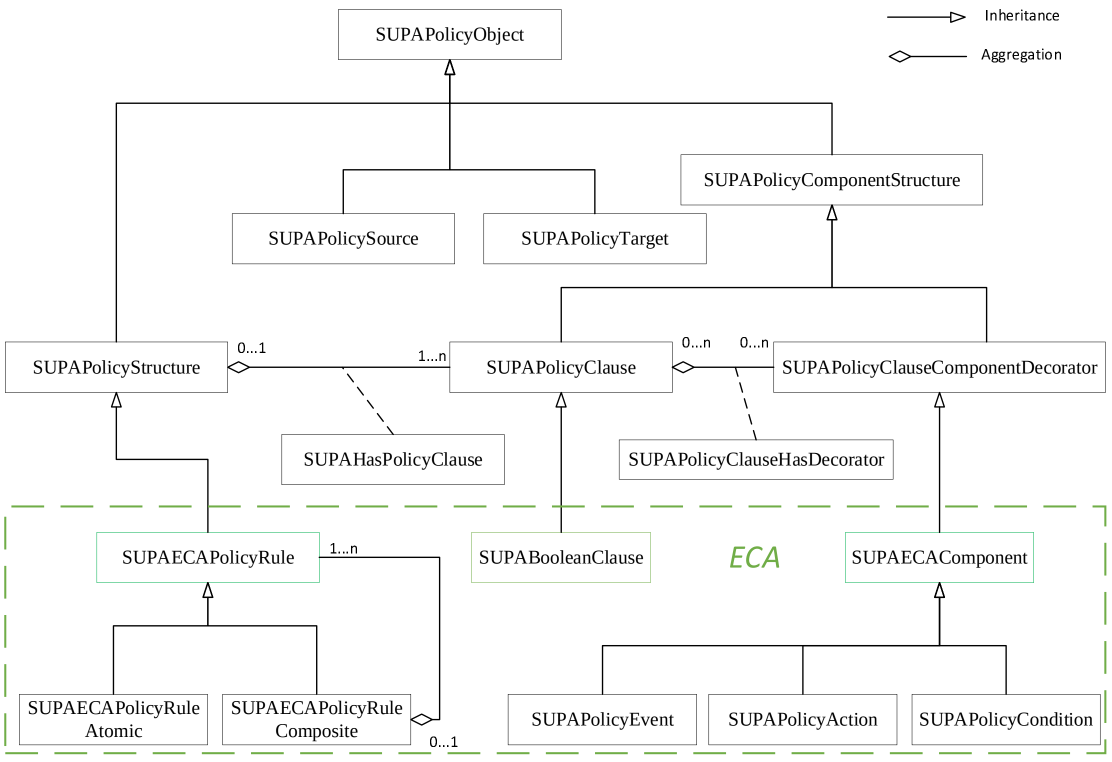

Figure 6.2.2.2-1: Functional View of the Top-Level EPRIM

EPRIM focuses on ECA policy rules and defines new classes and subclasses: SUPAECAPolicyRule, SUPAPolicyBooleanClause, SUPAECAComponent, which are inherited from the existing classes of GPIM.

SUPAECAPolicyRule: SUPAECAPolicyRule inherits from the SUPAPolicyStructure class in GPIM. It aggregates PolicyEvents, PolicyConditions, and PolicyActions into a type of policy rule known as an Event-Condition-Action (ECA) paradigm. EPRIM applies the composite pattern to the SUPAECAPolicyRule by creating two subclasses SUPAECAPolicyRuleAtomic and SUPAECAPolicyRuleComposite, which are used to model atomic ECA policy rule and composite ECA policy rules respectively.

SUPABooleanClause: A SUPABooleanClause specializes a SUPAPolicyClause and defines a Boolean expression consisting of a standard structure. SUPABooleanClause is used to construct Boolean clauses that can be used to determine events and conditions during the construction of ECA rules.

SUPAECAComponent: SUPAECAComponent inherits from the SUPAPolicyClauseComponentDecorator class in GPIM. A SUPAECAComponent defines three concrete subclasses, one each to represent the concepts of reusable events, conditions, and actions. They are called SUPAPolicyEvent, SUPAPolicyCondition, and SUPAPolicyAction, respectively.

6.2.2.3 A YANG data model for ECA policy management

6.2.2.3.1 Overview

In clauses 6.3.2.1 and 6.3.2.2, the GPIM and its extended ERPIM information model proposed by the IETF SUPA working group are introduced. It is worth noting that these two models have relatively high complexity and hard to be applied in practice. Therefore, the NETMOD working group of IETF refers to part of the ideas of the GPIM and EPRIM information models, and designs a YANG data model for ECA policy management in A YANG Data model for ECA Policy Management [i.12].

This YANG data model is designed to assist clients in transferring network management tasks to the server. The specific implementation is to enable the server to have the ability to control configuration, monitor status parameters, and take simple and timely actions when conditions are met.

This YANG data model adopts ECA policy rules. Therefore, the three key elements event, condition and action included in this policy rule are the key modules included in this data model. In addition, this YANG data model introduces key modules such as Policy Variable, ECA, and ECA XPath Function Library.

6.2.2.3.2 ECA policy variable

Policy Variables (PVs) are used to record and represent the intermediate variables required during the execution of the policy or the initial variables required by the events, conditions and actions of the policy. PVs are divided into global PVs and local PVs. Global PVs are used by multiple policy instances. Local PVs only be used by a single policy instance.

According to the different functions of PVs in the policy, it can be divided into PV-sources and PV-results. A PV-source is used to represent the result of XPATH, and is used to define the expression of intermediate variables needed in the process of determining whether an event or condition is met, or in the process of executing an action. A PV-result is used to record the evaluated result during the execution of the intermediate variable.

The structure of the Policy Variable model is as follows: (this structure is quoted from [i.16]):

```
+--rw policy-variables
|  +--rw policy-variable* [name]
|     +--rw name                         string
|     +--rw type                         identityref
|     +--rw (xpath-value-choice)?
|        +--:(policy-source)
|        |  +--rw (pv-source)
|        |     +--:(xpath-expr)
|        |     |  +--rw xpath-expr?        yang:xpath1.0
|        |     +--:(scalar-constant)
|        |     |  +--rw scalar-constant?    string
|        |     +--:(nodeset-constant)
|        |        +--rw nodeset-constant?   <anydata>
|        +--:(policy-result)
|           +--rw (pv-result)
|              +--:(scalar-value)
|              |  +--rw scalar-value?       string
|              +--:(nodeset-value)
|                 +--rw nodeset-value?      <anydata>
```

NOTE 1: For more information and details, please refer to clauses 3.1 and 3.3 of [i.16].

NOTE 2: Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

6.2.2.3.3 ECA event

The ECA Event is used to define the event of the strategy. The ECA Event is an event notification supported by any subscribed YANG module or an event stream pushed to the server through YANG Push subscription (a typical example is a netconf stream). ECA Event currently are divided into:

- server event
- datastore event
- timer event
- diagnostics event

The structure of the ECA Event model is as follows: (this structure is quoted from [i.16]):

```
+--rw events
|  +--rw event* [event-name]
|     +--rw event-name               string
|     +--rw event-type?              identityref
|     +--rw policy-variable*      -> /gncd/policy-variables/policy-variable/name
|     +--rw local-policy-variable*   -> /gncd/ecas/eca/policy-variable/name
|     +--rw (type-choice)?
|        +--:(server-event)
|        |  +--rw event-stream?      string
|        |  +--rw event-module?      string
|        |  +--rw event?             <anydata>
|        +--:(datastore-event)
|        |  +--rw datatore?          string
|        |  +--rw data-path?         string
|        |  +--rw data?              <anydata>
|        +--:(timer-event)
|        +--:(diagnostics-event)
```

NOTE 1: For more information and details, please refer to clause 3.2 of [i.16].

NOTE 2: Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

6.2.2.3.4 ECA condition

The ECA Condition is evaluated by the server when the event defined by ECA Event is triggered. The ECA Condition is a list that contains multiple ECA Condition entries. A single ECA Condition entry could be presented in the format of an Xpath expression. It is evaluated as either true or false.

The structure of the ECA Condition model is as follows: (this structure is quoted from [i.16]):

```
+--rw conditions
|  +--rw condition* [name]
|     +--rw name                    string
|     +--rw (expression-choice)?
|        +--:(xpath)
|           +--rw condition-xpath?   string
```

NOTE 1: For more information and details, please refer to clause 3.3 of [i.16].

NOTE 2: Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

6.2.2.3.5 ECA action

The ECA Action is executed when the ECA Condition is evaluated as true. The ECA Action is a list that contains multiple ECA Action entries. ECA Action could update or call the properties of the local managed object. The actions that ECA Action could perform are as follows:

- sending one-time notification

- (re-)configuration scheduling - scheduling one time or periodic (re-)configuration in the future
- stopping current ECA
- invoking the same ECA recursively

The structure of the ECA Action model is as follows: (this structure is quoted from [i.16]):

```
+--rw actions
|  +--rw time-schedule!
|  |  +--rw period?   centiseconds
|  +--rw action* [name]
|     +--rw name                   string
|     +--rw action-element* [name]
|     |  +--rw name                         string
|     |  +--rw action-type?                 identityref
|     |  +--rw (action-operation)?
|     |     +--:(action)
|     |     |  +--rw next-period          boolean
|     |     |  +--rw action-name?
|     |     |          -> /gnca/actions/action/name
|     |     +--:(function-call)
|     |     |  +--rw function-call
|     |     |     +--rw func-name       leafref
|     |     |     +--rw policy-source   leafref
|     |     |     +--rw policy-result   leafref
|     |     |  +--:(rpc-operation)
|     |     |  |  +--rw rpc-operation
|     |     |  |     +--rw rpc-name?          string
|     |     |  |     +--rw nc-action-xpath?   string
```

NOTE 1: For more information and details, please refer to clause 3.4 of [i.16].

NOTE 2: Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

6.2.2.3.6 ECAs

The ECAs is a key module of this YANG data model. It is a container that contains local PVs, global PVs and combinations of multiple events, conditions and actions. ECAs allows the creation of a CONDITION-ACTION list to allow multiple conditions to be combined to jointly trigger an action; different conditions could be combined to trigger different actions.

The structure of the ECAs model is as follows: (this structure is quoted from [i.16]):

```
+--rw ecas
|  +--rw eca* [name]
|     +--rw name                string
|     +--rw username            string
|     +--rw event-name          string
|     +--rw policy-variable* [name]
|     |  +--rw name                          leafref
|     |  +--rw is-static?                    boolean
|     +--rw condition-action* [name]
|     |  +--rw name         string
|     |  +--rw condition*   -> /gncd/conditions/condition/name
|     |  +--rw action?      -> /gncd/actions/action/name
|     +---x start
|     +---x stop
|     +---x next-action
```

NOTE 1: For more information and details, please refer to clause 3.5 of [i.16].

NOTE 2: Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

6.2.2.3.7 ECA XPath function library

The ECA XPath Function Library is a key module of this YANG data model. This module aims to provide standard or vendor-specific function library. The function library includes function calls, Remote Process Calls (RPC) and eca-names.

The structure of the ECA XPath Function Library model is as follows: (this structure is quoted from [i.16]):

```
+--rw eca-func-libs
   +--rw eca-function* [func-name]
   |   +--rw func-name    string
   +--rw eca-rpc* [rpc-name]
   |  +--rw rpc-name     string
   +--rw eca-name    -> /gncd/ecas/eca/name
```

NOTE 1: For more information and details, please refer to clause 3.6 of [i.16].

NOTE 2: Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

6.2.2.3.8 ECA information model derived from ECA YANG data model

According to the code provided by [i.18], the information model corresponding to this YANG data model could be derived by using UML conversion tool. The pyang(python) could be used to generate the .uml file of this data, and then the plantuml tool could be used to generate UML class diagrams. The informative conversion procedure and code is shown in Annex A.

NOTE: Other tools or methods could also be used for the conversion. The conversion methods are out of the scope of the present document.

6.2.3 Comparison of recommendations with EPRIM

6.2.3.1 Introduction

In this clause, the recommendations for policy model requirements which are described in clause 5.4 are compared with the EPRIM. The results could be 'Fully meets', 'Partially meets' and 'Does not meet'.

- Fully meets: the recommendation could be satisfied by the EPRIM.
- Partially meets: part of the recommendation could be satisfied by the EPRIM or this recommendation is under the consideration but needs more enhancements.
- Does not meet: the recommendation is not considered by the EPRIM at all.

6.2.3.2 Comparison with general recommendations for policy model

Table 6.2.3.2-1 provides the comparison results of the EPRIM and the general recommendations listed in clause 5.4.1 of the present document.

Table 6.2.3.2-1: Comparison results of general recommendations.

Rec Number	Status	Comments
POLICY.MODEL. GEN.001	Fully meets	The EPRIM uses ECA policy form to support the imperative policy paradigm. SUPAECAPolicyRule is an imperative policy rule and defines its structure.
POLICY.MODEL. GEN.002	Fully meets	The EPRIM uses SUPAPolicyStructure to support the ECA policy expression form.
POLICY.MODEL. GEN.003	Fully meets	The EPRIM has classes SUPAPolicyEvent, SUPAPolicyAction and SUPAPolicyCondition to support this recommendation.
POLICY.MODEL. GEN.004	Partially meets	The EPRIM has the class SUPAECAComponent to aggregate policy events, conditions and actions. However, there is currently no explicit description to support the different combinations of policy events, conditions and actions in a single policy to achieve the goals.
POLICY.MODEL. GEN.005	Does not meet	The EPRIM does not support the concept of context.

6.2.3.3 Comparison with recommendations for policy model information elements

Table 6.2.3.3-1 provides the comparison results of the EPRIM and the recommendations for policy model information elements listed in clause 5.4.2 of the present document.

Table 6.2.3.3-1: Comparison results of recommendations for policy model information elements

Rec Number	Status	Comments
POLICY.MODEL. INFO.001	Fully meets	There are classes SUPAPolicyEvent, SUPAPolicyAction and SUPAPolicyCondition to support this recommendation.
POLICY.MODEL. INFO.002	Fully meets	There are classes SUPAPolicyStructure and SUPAECAPolicyRule to support this recommendation. More details could be found in clauses 5.4, 6.4 of [i.4].
POLICY.MODEL. INFO.003	Partially meets	EPRIM and GRIM define relationships in classes and between classes completely and totally; define most of the attributes; emphasize the importance of methods and constraints, but give no details.
POLICY.MODEL. INFO.004	Fully meets	There are class SUPAPolicyTerm, SUPAPolicyVariable, SUPAPolicyOperator, SUPAPolicyValue to support this recommendation. The details are described in clauses 5.8, 5.9, 5.10, 5.11 of [i.4].
POLICY.MODEL. CLASS.005	Does not meet	The EPRIM does not support the concept of context.

6.2.3.4 Comparison with recommendations for policy model attributes

Table 6.2.3.4-1 provides the comparison results of the EPRIM and the recommendations for policy model attributes listed in clause 5.4.3 of the present document.

Table 6.2.3.4-1: Comparison results of recommendations for policy model attributes

Number	Status	Comments
POLICY.MODEL. ATTR.001	Does not meet	Policy event types are not considered by EPRIM.
POLICY.MODEL. ATTR.002	Partially meets	There is a class SUPAPolicySource to support this recommendation. However, no attributes are defined yet, but extension could be made to fully meet this recommendation. More details could be found in clause 5.14 of [i.4].
POLICY.MODEL. ATTR.003	Fully meets	Conjunctive normal form and disjunctive normal form are used to support this recommendation.
POLICY.MODEL. ATTR.004	Fully meets	There are attributes supaPolVarName, supaPolOpType, supaPolValContent and so on to support this recommendation. More details could be found in clauses 5.9, 5.10, 5.11 of [i.4].
POLICY.MODEL. ATTR.005	Fully meets	There are attributes 'supaPolExecStatus' and 'supaPolExecFailStrategy' in class 'SUPAPolicyStructure' to support this recommendation. More details could be found in clauses 5.9, 5.10, 5.11 of [i.4].
POLICY.MODEL. ATTR.006	Does not meet	The mapping relationships are not considered by EPRIM.
POLICY.MODEL.A TTR.007	Partially meets	There are classes SUPAPolicyCondition and SUPAPolicyAction to support this recommendation. However, no attributes are defined yet, but extension could be made to satisfy this recommendation.

6.2.4 Features comparison and gaps analysis

IETF SUPA has two information models GPIM and EPRIM. Moreover, EPRIM is the extension of GPIM, which means EPRIM inherits GPIM and has specific features and characteristics to support ECA policy form. In this clause, the features from [i.3] and [i.4] will be listed as a summary. Moreover, gap analysis will also be listed based on analysis in clause 6.2.3.

Some features of GPIM and EPRIM are as follows:

- SUPA defines a general policy information model GPIM. GPIM provides a general extensible framework to enable the different type of policies to share common model elements. By adding new subclasses and associations based on the GPIM model, the general information model could be extended to represent specific types of policy rules, such as imperative policy and declarative policy.
- Currently, based on GPIM, an EPRIM information model suitable for ECA form policy rules has been established. For other types of policy rules (such as declarative policies), no corresponding information model standards have been established.
- In the key places of the GPIM information model, the composite pattern and decorator pattern are adopted, which has high flexibility and extensible:
  - Composite pattern allows one to have composite entities which are made up of either other composites, or any of the atomic subclasses (e.g. SUPAECAPolicyRuleAtomic).
  - Decorator pattern enables behaviour to be selectively added to an individual object, either statically or dynamically, without affecting the behaviour of other objects from the same class. The decorator pattern uses composition, instead of inheritance, to avoid class and relationship explosion
- In the GPIM model, the association between classes is common. Therefore, the association is abstracted as a class, so that attributes, operations, and other characteristics could be added to the association between classes.

Gap analysis based on comparison results in clause 6.2.3 are as follows:

1) Most of the general recommendations described in clause 5.4.1 are met. However, there are some aspects to be considered:
   - This policy model does not support the different combinations of policy items to form a complete policy. It can cause that complex policies need to be separated into several simple policies if combinations of policy events, conditions and actions to achieve the same function of a policy are not supported.
   - This policy model does not support the concept of context. It can cause the policy could not be context-aware policy. It could be a limitation to constrain the policy to make dynamic adaptions based on changes of context.
2) Most of the recommendations for policy model information elements described in clause 5.4.2 are met. However, there are some aspects to be considered:
   - This policy model does not give too much information about attributes, relationships, methods and constraints of the EPRIM. It can make it difficult to reference or reuse this policy model since more adaptions and enhancements are expected.
   - There is no attribute to support the description of the mapping between policy items. It can cause low flexibility to constrain policy designers to design a complex policy and lead to a large number of policies to be designed in order to finish a complex policy. This gap could increase the burden on the repository.
3) Most of the recommendations for policy model attributes described in clause 5.4.3 are met or considered. However, since this policy information model is expected to provide extendibility, there are some issues to be considered:
   - There is no attribute to support the description of policy event types and the mapping relationships between policy items. In this situation, more information elements are needed to contain these attributes.
   - There is no attribute to show the source needed by the policy items. However, it could be enhanced easily since there is a class SUPAPolicySource which could be used to contain this attribute.

6.3 ONAP policy model

6.3.1 Introduction to ONAP policy model

The ONAP policy runtime architecture defined in [i.7] includes two components, PAP (Policy Administration Point, that administers and manages policies) and PDP (Policy Deployment Point, that executes one or more policy artifacts). ONAP supports the use of a unified PAP to connect and manage multiple PDPs, which are used to design and execute different types of policies.

PAP is mainly responsible for two aspects. One aspect is interacting with policy database and transferring policies to PDPs. The other aspect is the lifecycle management of policies, including but not limited to setting up different PDP groups; reading PDP metadata, policies and artifacts of policy sets; set the working mode and life cycle mode of the PDP.

The PDP is responsible for the execution of the policies issued by the PAP. ONAP currently supports 3 different types of PDPs, namely PDP-APEX (PDP-A), PDP-DROOLS (PDP-D), PDP-XACML (PDPX). The policy language, internal architecture, and policy implementation principles used by these three PDPs are not the same. Different types of ONAP policy requirements may be supported by these three PDPs, or only one or two PDPs.

The ONAP policy design uses the TOSCA template uniformly and generates yaml files for policy developers to choose and use. The selection is based on the specific requirements for policy implementation. The policy developer then fills in the content of the policy type according to actual needs, to generate the policy instance yaml or json file and send it to the PAP, and then the PAP sends the policy to different types of PDP to run according to the actual situation such as context.

A policy instance issued to PAP is created based on the content defined in the policy type yaml file. The developer fills in more details under the framework of the policy type, such as specific fields, objects, values, etc. and then generates a policy instance yaml or json file. Generally speaking, policy designers will generate policy yaml files for PDP-XACML and PDP-DROOLS, and policy json files for PDP-APEX.

A specific policy requirement corresponds to a specific policy type. A policy type is supported by at least one PDP type, and it may also be supported by multiple PDP types. If the latter situation occurs, the policy developer needs to determine which PDP type to support and fill in the details according to the selected PDP type. The policy type supported by the PDP is mainly determined by the characteristics of the PDP engine. Currently ONAP defines a variety of basic policy types and derived sub-policy types for policy developers to choose.

6.3.2 Architecture

6.3.2.0 Introductionto PDP-APEX (PDP-A)

The characteristics of the PDP-A policy model introduced in clause 6.3.2 are the flexibility of defining, deploying, and executing policies. APEX stands for Adaptive Policy EXecution. APEX supports almost all existing policy models, and supports the conversion of legacy policies into the execution format supported by APEX.

Contents and figures presented in clause 6.3.2 are based on websites [i.8] and [i.9]. Some descriptions and figures are cited directly from these references. Some descriptions and figures are summarized based on these references.

There are 3 types of PDPs, which are PDP-APEX (PDP-A), PDP-Drools (PDP-D) and PDP-XACML (PDP-X). Based on initial analysis of different types of PDPs supported in ONAP, PDP-APEX (PDP-A) supports relevant policy models that meet the policy aspects defined in ETSI NFV GR IFA-023 [i.6]. Hence, PDP-APEX (PDP-A) is analysed in detail in clause 6.3.2. PDP-Drools (PDP-D) and PDP-XACML (PDPX) are not considered in the present document.

6.3.2.1 ONAP policy execution engine

The APEX policy execution engine is a lightweight engine that divides the execution of the policy into three steps. The first step is to enter the policy trigger event into the engine. As shown in figure 6.3.2.1-1, the trigger event is defined as stimulus event. The second step is the execution of the policy. The policy consists of multiple states forming an n-states state chain. Each state chooses to execute different tasks through task selection logic. The third step is the policy execution response.

This type of design can more flexibly support different expression formats of policies. For example, the simplest policy has only one state, and the complex policy is represented by n-state (n takes any value). For example, the ECA policy is represented by the three-state model, and the MEDA (Match-Establish-Decide-Act) policy is represented by the four-state model.

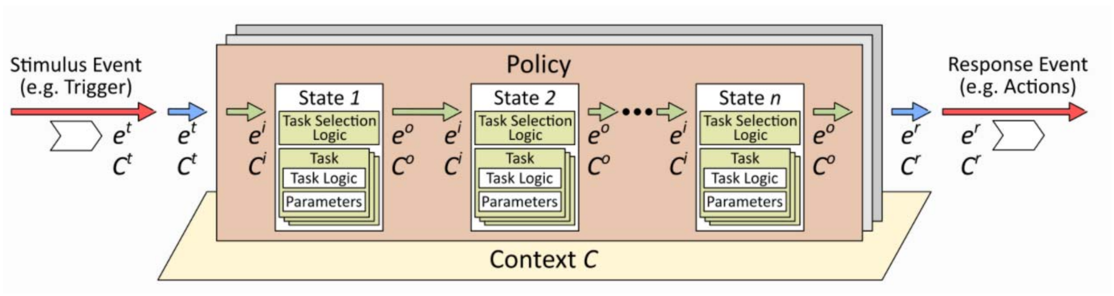

Figure 6.3.2.1-1: APEX States and Context

6.3.2.2 APEX policy design

6.3.2.2.0 APEX policy design parts

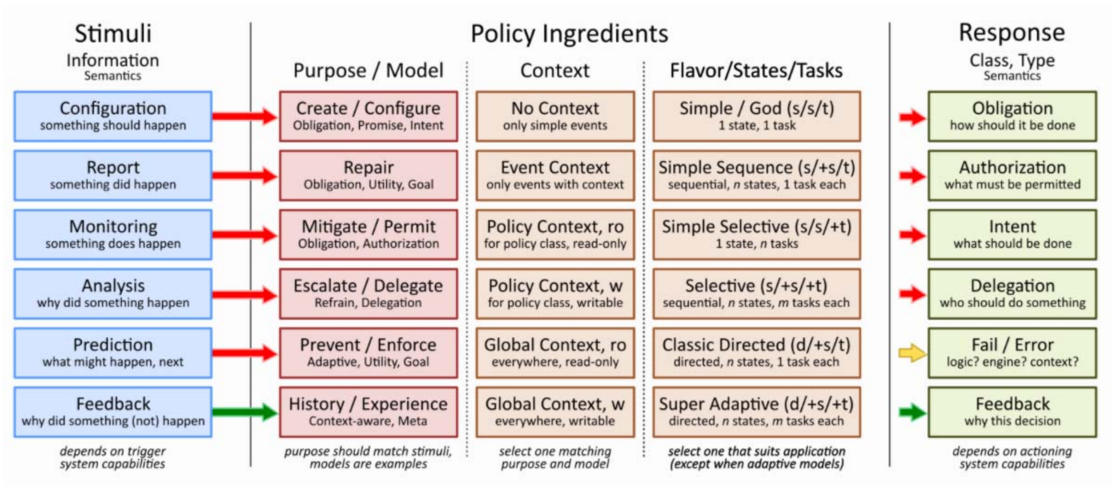

Figure 6.3.2.2.0-1: Policy Design Process

In the above abstract model, APEX policy design is divided into three parts:

- The first part 'Stimuli' defines the triggering conditions of the policy input.
- The second part 'Policy Ingredients' defines the goal, context and component structure of the internal analysis and processing of the policy.
- The third part 'Response' defines the policy feedback or output after execution.

APEX supports developers in the continuous development of policy components. At the same time, it supports more component combinations and the definition of new components. Figure 6.3.2.2.0-1 shows the common APEX policy design combinations currently supported by ONAP.

6.3.2.2.1 Policy input

APEX currently defines 6 types of Stimuli, including: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- Configuration, i.e. what should happen. An example is an event that states an intended network configuration and the policy should provide the detailed actions for it. The policy can be realized for instance as an obligation policy, a promise or an intent.

- Report, i.e. something did happen. An example is an event about an error or fault and the policy needs to repair that problem. The policy would usually be an obligation, utility function, or goal policy.
- Monitoring, i.e. something does happen. An example is a notification about certain network conditions, to which the policy might (or might not) react. The policy will mitigate the monitored events or permit (deny) related actions as an obligation or authorization.
- Analysis, i.e. why did something happen. An example is an analytic component sends insights of a situation requiring a policy to act on it. The policy can solve the problem, escalate it, or delegate it as a refrain or delegation policy.
- Prediction, i.e. what will happen next. An example are events that a policy uses to predict a future network condition. The policy can prevent or enforce the prediction as an adaptive policy, a utility function, or a goal.
- Feedback, i.e. why did something happen or not happen. Similar to analysis, but here the feedback will be in the input event and the policy needs to something with that information. Feedback can be related to history or experience, for instance a previous policy execution. The policy needs to be context-aware or be a meta-policy.

6.3.2.2.2 Policy context

The options supported by the policy context are as follows: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- No context: only trigger events, such as a string or number required.
- Event context: the input event, which provides all the information for the policy.
- Policy context (read only): policy can access other additional information related to it, but the information cannot be modified.
- Policy context (read and write): policy can access other additional information related to it, and the information can be modified.
- Global context (read only): policy can access any additional information, but the information cannot be modified.
- Global context (read and write): policy can access any additional information, and the information can be modified.

6.3.2.2.3 Policy combinations

In order to determine how the policy completes the task specified by the user, APXE elaborates the design and combination process of the states and tasks of the policy. The typical combination is as follows: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- Simple/God: A simple policy with 1 state and 1 task, such as determining configuration parameters or simple access control.
- Simple sequence: A simple policy with multiple states, each state has one task. For example, ECA policy will have 3 states (E, C, and A), each with 1 logic (1 task).
- Simple selective: A policy with one state but multiple tasks. The policy will select the appropriate task (and its logic) when executed.
- Selective: A complex policy with multiple states including multiple tasks. For example, an ECA policy has multiple tasks in E, C, and A.
- Classic directed: A policy that includes multiple states whose execution sequence is dynamically determined, and is suitable for implementing decision trees based on context information.
- Super Adaptive/Free Style: A policy that has no restrictions of states and tasks, and do calculation in the running state (executed by policy). This policy is very close to a general programming system and can solve very complex problems.

6.3.2.2.4 Policy response

The response represents the feedback sent to the policy user or other components that issue requirements or the corresponding component that processes the policy next: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- Responsibility (what should happen).
- Authorization (for example, rule-based access or authorization of other access control or security systems).
- Intent (the response is not to provide detailed operations, but an intent statement which will be processed by a system further).
- Designation (hand over the problem to other components, and possibly provide some information or instructions).
- Failure/error (the policy encounters a problem and reports it).
- Feedback (why should the policy make a specific decision).

6.3.2.3 APEX policy model

6.3.2.3.1 Basic concepts

As shown in figure 6.3.2.3.1-1, the APEX policy model defines a series of common operations and two types of reference methods for the base classes.

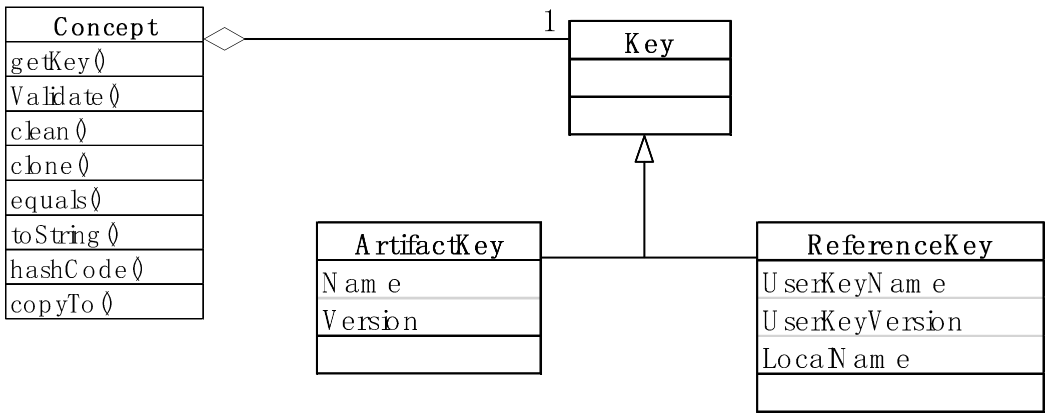

Figure 6.3.2.3.1-1: Concept and Keys in APEX Policy model

'Concept' is the parent class of all classes. These methods are inherited by all subclasses and applied to policy design and implementation. At the same time, each concept has a key which is responsible for uniquely defining the concept instance.

The parent class 'Concept' has 8 virtual methods:

- Getkey() - gets the unique key for this concept instance in the system.
- Validate() - validates the structure of this concept, its sub-concepts and its relationships.
- Clean() - carries out housekeeping on the concept such as trimming strings, remove any hanging references.
- Clone() - creates a deep copy of an instance of this concept.
- Equals() - checks if two instances of this concept are equal.
- toString() - returns a string representation of the concept.
- hashCode() - returns a hash code for the concept.

- copyTo() - carries out a deep copy of one instance of the concept to another instance, overwriting the target fields.

The parent class 'Concept' has 2 two types of keys:

- ArtifactKey: Independent concepts such as Policy, Task, Event, etc.
- ArtifactKey has two fields:
    - Name: The name of the concept, which is unique in a given policy model.
    - version: The version number of the concept, using major.minor.path scheme.
- ReferenceKey: sub-concepts included in other independent concepts, such as state, EventParameter, etc. has ReferenceKey.
- ReferenceKey has three fields:
    - UserKeyName: The name of the independent concept corresponding to this non-independent concept.
    - UserKeyVersion: The version number of the independent concept corresponding to this non-independent concept.
    - LocalName: The name of this non-independent concept, which only needs to be unique among all child concept names included in the parent concept.

6.3.2.3.2 APEX policy information model

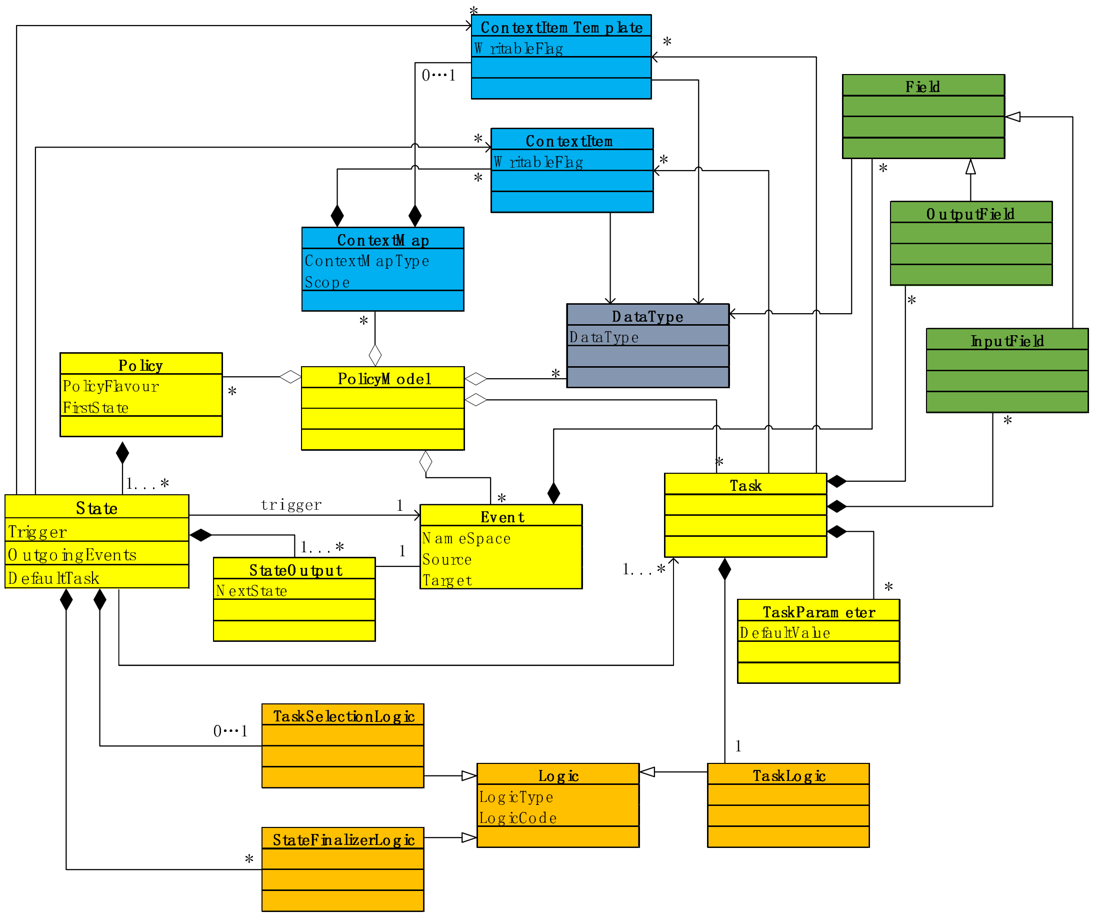

Figure 6.3.2.3.2-1: ONAP APEX Policy UML Class Diagram

Figure 6.3.2.3.2-1 shows the UML class diagram of the ONAP APEX policy model:

- General model perspective: the yellow part on the corresponding map is the core component of the APEX policy model.
- Logic model perspective: the orange-marked part on the corresponding figure is the logic-related classes in the APEX policy.
- Context model perspective: the blue part of the corresponding figure is the context-related classes (including datatype) in the APEX policy.
- Event and field model perspective: the green part of the corresponding figure is the event-related classes in the APEX policy, and they together represent the relevant information about the input and output of the policy.

6.3.2.3.3 General model

6.3.2.3.3.1 Task

Task is the smallest unit of a policy. A task is encapsulated into a logic unit and is designed as an indivisible execution unit that can be referenced by multiple policies at the same time.

A task sets multiple input fields and multiple output fields. Multiple tasks contained in the same State are executed sequentially through output fields and input fields. In other words, the output field output by the task are used as the input field of another task.

Tasks use or modify related context during execution. Task contains a set of ContextItem and ContextItemTemplate for task access (access rights and methods are determined by the WritableFlag fields of the two context-related concepts). Task in the running state only accesses the context items defined by the policy designer in the design state.

A task is configured by certain parameters. These parameters are defined by the TaskParameter concept.

6.3.2.3.3.2 TaskParameter

TaskParameter is responsible for representing the configuration parameters of the task. TaskParameter is a non-independent concept, so its instance is keyed with a referenceKey. The LocalName field of ReferenceKey contains the names of all parameters.

TaskParameter has two fields: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- DefaultValue: defines the default value that the task parameter is set to.
- TaskParameter: contains one or more task parameters, where each item contains the parameter key, value as well as the taskId to which it is associated. If the taskId is not specified, then the parameters are added to all tasks.

6.3.2.3.3.3 State

A set of states are used to form a policy. Each state has at least one task. If there are multiple optional tasks, each time it is executed, the state calls TaskSelectionLogic to select one of the tasks according to the context.

State has three fields: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- Trigger: holds the ArtifactKey of the event that triggers this state.
- OutgoingEvents: holds the ArtifactKey references of all possible events that may be output from the state.
- DefaultTask: holds a reference to the default task for the state, a task that is executed if no task selection logic is specified. If the state has only one task, that task is the default task.
- NextState: a field of a state holds the ReferenceKey key of the state in the policy to execute after this state.

6.3.2.3.3.4 Policy

Policy concept is one of the core concepts of APEX policy model. It is keyed with an artifactKey. A policy is composed of multiple states, and each policy has a different flavour, which is used to determine all the states included in the policy.

Policy has two fields: definitions below are cited from ONAP websites:

- PolicyFlavour: a policy paradigm with a specific state structure. Currently supports MEDA flavour, OODA (Observe, Orient, Decide, Act) flavour, ECA flavour, XACML flavour, FREESTYLE flavour, etc.
- FirstState: the first state of the state-chain of the policy. The trigger event is also the trigger event of the entire policy.

6.3.2.3.3.5 PolicyModel

PolicyModel is a container that contains definitions of all policies and their associated events, context maps, and tasks. PolicyModel has a map relationship with other concepts, specifically policy, event, context map, tasks. Each map is indexed by the above-mentioned concept keys. Because all policies contain at least one Input event, one output event and execute at least one task, at least one entry is included in the policy, event, and task map in the design process of the policy model.

PolicyModel is an independent concept. It is keyed with an artifactKey, which uses the validate method to verify the concepts, structure, and relationships of the entire policy model.

6.3.2.3.4 Logic model

6.3.2.3.4.1 Logic

As shown in the orange part of figure 6.3.2.3.2-1, the main function of Logic concept is to define programming task logic for tasks defined by Task concept or task selection logic defined by State concept.

Logic is a non-independent concept, so it is keyed with a ReferenceKey. The LocalName field is used to indicate the name of the logic.

Logic has two fields:

- LogicCode: a string that holds the program code that is to be executed at run time.
- LogicType: defines the language of the code.

6.3.2.3.4.2 TaskLogic

TaskLogic is responsible for decision-making in task.

TaskSelectionLogic is responsible for selecting a specific task from the state containing multiple tasks.

StateFinalizerLogic is responsible for calculating the final output event in the state and selecting the next appropriate state from the policy model.

6.3.2.3.5 Context model

6.3.2.3.5.1 DataType

In order to provide better consistency and facilitate the tracking of context changes, all contexts have a DataType concept: input and output contexts are represented by Field concepts. Other contexts are represented by ContextItem concepts.

Because DataType is an independent concept, its instance is keyed with an artifactKey.

The DataType field in the DataType concept is used to represent the Java class of the Datatype instance object.

6.3.2.3.5.2 ContextMap

APEX defines the context set involved in the policy model as a ContextMap instance, including a set of context instances and their templates.

ContextMap is an independent concept. It is keyed with an artifactKey. The ContexItem concept instance is a dependent concept subordinate to ContextMap.

ContextMap has two fields: (definitions below are cited from ONAP websites [i.8] and [i.9]):

- ContextMapType: defines the type of context map as BAG or SAMETYPE. Definitions below are cited from ONAP websites:
  - BAG: a context map with fixed content. Each possible context item in the context map is defined at design time and is held in the ContextMap context instance as ContextItem concept definitions and only the values of the context items in the context map can be changed at run time.
  - SAMETYPE: a context map used to represent a group of ContextItem instances of the same type. Unlike a BAG context map, the ContextItem concept instances of a SAMETYPE context map can be added, modified, and deleted at runtime.
- Scope: defines the scope of application of context map. Currently, APEX defines four types of scopes. Definitions below are cited from ONAP websites:
  - EPHEMERAL scope means that the context map is owned, used, and modified by a single application but the context map only exists while that application is running.
  - APPLICATION scope specifies that the context map is owned, used, and modified by a single application, the context map is persistent.
  - GLOBAL scope specifies that the context map is globally owned and is used and modified by any application, the context map is persistent.
  - EXTERNAL scope specifies that the context map is owned by an external system and may be used in a read-only manner by any application, the context map is persistent.

6.3.2.3.5.3 ContextItem

The ContextItem concept instance is keyed with a ReferenceKey, and the name of the context item is contained in the LocalName field of the ReferenceKey. ContextItem has a reference relationship to DataType, and this relationship defines the value of this context item at runtime.

ContextItem has one field:

- WritableFlag: Indicates whether the context item is read-only or can be read and written at the same time in the running state.

6.3.2.3.5.4 ContextTemplate

For SAMETYPE ContextMap that can adjust ContextItem in the running state, ContextItemTemplate provides a standardized template for its instances, that is, every ContextItem concept instance is created using this template.

Because ContextItemTemplate is an independent concept, its instance is keyed with a ReferenceKey.

ContextItem has one field:

- WritableFlag: Indicates whether the context item is read-only or can be read and written at the same time in the running state.

6.3.2.3.6 Event/Field model

6.3.2.3.6.1 Event

Event is an independent concept. It is keyed with an artifactKey. Event and Field have a combination relationship. All parameters of an event are the mapping of a Field concept instance.

Event has 3 fields:

- NameSpace: determine the application domain of an event.
- Source: determine the system that outputs the event.
- Target: determine the system that receives this event.

6.3.3 Comparison of recommendations with ONAP APEX policy model

6.3.3.1 Introduction

In this clause, the recommendations for policy model requirements which are described in clause 5.4 are compared with the ONAP APEX policy model. The results could be 'Fully meets', 'Partially meets' and 'Does not meet'.

- Fully meets: the recommendation could be satisfied by the ONAP APEX policy model.
- Partially meets: part of the recommendation could be satisfied by the ONAP APEX policy model or this recommendation is under the consideration but needs more enhancements.
- Does not meet: the recommendation is not considered by the ONAP APEX policy model at all.

6.3.3.2 Comparison with general recommendations for policy model

Table 6.3.3.2-1 provides the comparison results of the ONAP APEX policy model and the general recommendations listed in clause 5.4.1 of the present document.

Table 6.3.3.2-1: Comparison results of general recommendations

Rec Number	Status	Comments
POLICY.MODEL.GEN.001	Fully meets	The ONAP APEX policy model explicitly supports imperative policy paradigm. It has a class Policy which has the field PolicyFlavour to support many types of imperative policy paradigms such as MEDA, ECA, OODA, etc. More details are described in clause 6.3.2.3.3.4.
POLICY.MODEL.GEN.002	Fully meets	The ONAP APEX policy model support the ECA policy expression form by supporting ECA flavour. More details are described in clause 6.3.2.3.3.4.
POLICY.MODEL.GEN.003	Fully meets	The ONAP APEX policy model has classes Event, State, Task and other related classes to support this recommendation. More details are described in clause 6.3.2.3.2.
POLICY.MODEL.GEN.004	Fully meets	The ONAP APEX policy model has the classes TaskLogic, TaskSelectionLogic, StateFinalizerLogic to support this recommendation. More details are described in clause 6.3.2.3.4.2.
POLICY.MODEL.GEN.005	Fully meets	The ONAP APEX policy model has classes ContextMap, ContextItem, CotextTemplate, Datatype to support this recommendation. More details are described in clause 6.3.2.3.5.

6.3.3.3 Comparison with recommendations for policy model information elements

Table 6.3.3.3-1 provides the comparison results of the ONAP APEX policy model and the recommendations for policy model information elements listed in clause 5.4.2 of the present document.

Table 6.3.3.3-1: Comparison results of recommendations for policy model information elements

Rec Number	Status	Comments
POLICY.MODEL. INFO.001	Fully meets	There is a class Event to represent policy events. There are classes State, Task and other related classes to represent policy conditions and actions together.
POLICY.MODEL. INFO.002	Fully meets	There is a class PolicyModel to support this recommendation. More details are described in clause 6.3.2.3.3.5.
POLICY.MODEL. INFO.003	Fully meets	There is a class PolicyModel to support this recommendation. More details are described in clause 6.3.2.3.3.5.
POLICY.MODEL. INFO.004	Fully meets	To provide more flexibilities, the ONAP APEX policy model does not define the statement form explicitly. However, it has a class Logic and the field LogicType to define the language of the code. It means the statement form is defined by the chosen coding language. More details are described in clause 6.3.2.3.3.5.
POLICY.MODEL. CLASS.005	Fully meets	There are classes ContextMap, ContextItem and other related classes to support this recommendation. More details are described in clause 6.3.2.3.5.

6.3.3.4 Comparison with recommendations for policy model attributes

Table 6.3.3.4-1 provides the comparison results of the ONAP APEX policy model and the recommendations for policy model attributes listed in clause 5.3.3 of the present document.

Table 6.3.3.4-1: Comparison results of recommendations for policy model attributes

Number	Status	Comments
POLICY.MODEL. ATTR.001	Partially meets	There is a class Event used to represent the details of policy events. However, attribute(s) should be added to the class above to satisfy this recommendation. More details are described in clause 6.3.2.3.6.1.
POLICY.MODEL. ATTR.002	Fully meets	There is a field Source in the class Event to determine the system that outputs the event. More details are described in clause 6.3.2.3.6.1.
POLICY.MODEL. ATTR.003	Partially meets	There is a class PolicyModel used to represent the relationships between policy events, conditions and actions. However, attribute(s) should be added to the class above to satisfy this recommendation. More details are described in clause 6.3.2.3.3.5.
POLICY.MODEL. ATTR.004	Partially meets	The ONAP APEX policy model allows different coding languages to define their own policy description forms. This recommendation could be satisfied by the coding languages directly. More details are described in clause 6.3.2.3.4.1.
POLICY.MODEL. ATTR.005	Fully meets	There are fields DefaultTask, OutgoingEvents, NextState in the class State and the classes related to task selection to define the execution sequence and the strategies of the policy actions. More details are described in clause 6.3.2.3.3.3 and clause 6.3.2.3.4.2.
POLICY.MODEL. ATTR.006	Fully meets	There are fields DefaultTask, OutgoingEvents, NextState in the class State and the classes related to task selection to define the mapping relationships between policy conditions and actions. More details are described in clause 6.3.2.3.3.3 and clause 6.3.2.3.4.2.
POLICY.MODEL. ATTR.007	Partially meets	There are classes State and Task that could be used to support these functions. However, attribute(s) should be added to the classes above to satisfy this recommendation. More details are described in clause 6.3.2.3.3.3 and clause 6.3.2.3.3.1.

6.3.4 Features comparison and gaps analysis

Policy framework related to ONAP is defined in [i.9]. In this clause, the features derived from [i.9] will be listed as a summary. Moreover, gap analysis will also be listed based on analysis in clause 6.3.3.

Features of ONAP APEX policy model are as follows:

1) ETSI NFV-IFA GR 023 [i.6] defines a policy management and enforcement structure which is similar to that of ONAP. The two main components are PAP and PF.

2) The PAP defined in ETSI NFV-IFA GR 023 [i.6] and ONAP have the same part in function. Both of them are responsible for the life cycle management of policies. PAP in ETSI NFV-IFA GR 023 [i.6] is responsible for receiving the policy issued by the policy designer and sending it to the PF located in different MANO FBs. The difference is that there are no multiple types of PF, and PAP only needs to determine which FB the policy is applied to.

3) The PF defined in [i.6] is similar to ONAP PDP, and is responsible for the analysis, decision-making, and execution of the policy. When the triggering conditions of the policy are met, PF executes a series of actions of the policy.

4) ONAP supports the use of a unified PAP to connect and manage multiple PDPs, which are used to design and execute different types of policies.

5) The APEX policy engine uses components and parameters such as state, task, context to provide maximum flexibility in policy design and implementation in a manner close to general programming

6) The APEX policy information model supports the differentiation of policy intent/objectives/types through policy modelling.

7) In ONAP, the policy framework is deployment agnostic and model driven so that policy development, deployment, and execution is as flexible as possible. Policy API, Policy PAP, PDPs, Policy engine etc. can be deployed as micro services in suitable ONAP components.

Gap analysis based on comparison results in clause 6.1.3 are as follows:

1) All of the general recommendations described in clause 5.4.1 are met.

2) All of the recommendations for policy model information elements described in clause 5.4.2 are met.

3) All of the recommendations for policy model elements described in clause 5.4.3 are met or considered. However, there are two recommendations specific to NFV-MANO policy modelling which could not be met directly. Therefore, there are some issues to be considered:

   - There is no attribute to define the policy event types. However, it could be enhanced easily since there is a class Event which could be used to contain this attribute.

   - There is no attribute to define the statement form of policy conditions and actions. However, it could be achieved by another way: in class Logic, there is an attribute LogicType which allows different types of programming languages to define the statement form.

NOTE: The programming languages applicable to NFV-MANO policy modelling are out of the scope of the present document.

7 Analysis and recommendations

7.1 Summary of gaps analysis between NFV-MANO and other SDOs policy models

7.1.1 Perspective#1: Degree of fulfilment of recommendations

Table 7.1.1-1 analyses the degree of fulfilment of the TMF SID, IETF EPRIM and ONAP APEX policy models to the recommendations to NFV-MANO policy model requirements listed in clause 5.4 of the present document.

Table 7.1.1-1: Statistics on the degree of fulfilment of recommendations

	Number of 'Fully meets'	Number of 'Partially meets'	Number of 'Does not meet'
TMF SID	9	5	3
IETF EPRIM	9	4	4
ONAP APEX	13	5	0

TMF SID: three recommendations related to context-aware policies are not satisfied; more enhancements are needed for five recommendations with mark of "Partially meets".

IETF EPRIM: three recommendations related to context-aware policies and one recommendation to policy types are not satisfied; more enhancements are needed for four recommendations with mark of "Partially meets".

ONAP APEX: all recommendations are satisfied or considered; more enhancements are needed for four recommendations with mark of "Partially meets".

7.1.2 Perspective#2: Possibility in alignment and enhancement

Table 7.1.2-1 analyses the support of other SDO's policy models for NFV-MANO policy paradigms and the collaboration possibilities between ETSI ISG NFV and respective SDO.

Table 7.1.2-1: Support for policy paradigms and the collaboration possibility

	Support for imperative policy paradigm	Support for declarative policy paradigm	Collaboration possibility	Implementations or frameworks
TMF SID	Only ECA	No, and corresponding enhancements are impossible	Medium, last update is in October, 2018	Framework
IETF EPRIM	Only ECA	No, and corresponding enhancements are impossible	Low, the working group is closed	No
ONAP APEX	ECA, OODA, MEDA, FREESTYLE	No, but corresponding enhancements are possible	High, work on policy is ongoing	APEX PDP Engine; ONAP policy framework architecture

TMF SID: only supports ECA imperative policy paradigm since classes are specific to ECA form; impossible to support declarative policy paradigm since classes are specific to ECA form; medium collaboration possibility with ISG NFV on policy model design since the lase update is in October, 2018; TMF SID Frameworx references for future use of the TMF SID policy model.

IETF EPRIM: only supports ECA imperative policy paradigm since classes are specific to ECA form; impossible to support declarative policy paradigm since classes are specific to ECA form; GPIM claims to support declarative policy paradigm, but concrete specification is not available yet; low collaboration possibility with ISG NFV on policy model design since the corresponding working group is closed by IETF; there are no known implementations or frameworks which use IETF SUPA EPRIM.

ONAP APEX: supports ECA imperative policy paradigm and other imperative policy paradigms, such as OODA, MEDA, FREESTYLE since the classes are not specific to ECA form; does not support declarative policy paradigm, but is possible to support it by enhancements since the flexibility of the design; high collaboration possibility with ISG NFV on policy model design since the ONAP Policy Project is active; there is an APEX PDP engine which uses ONAP APEX policy model and more details could be found in [i.9]; there is the Policy Framework used for ONAP APEX policy and more details could be found in [i.9] and [i.14].

As described in clause 6.2.2 of ETSI GR NFV-IFA 041 [i.12], there is a relationship between intent and policy. On one hand, intent can be regarded as a kind of declarative policy. On the other hand, the interpretation of an intent can take advantage of polices to be applied to NFV-MANO as part of intent realization.

Based on analysis in clause 7.1.2 of the present document, no policy models can be used to describe declarative policy (e.g. intent). However, there is a possibility to enhance ONAP APEX policy model to support the description of a declarative policy.

The policies derived from the Intent Management interpretation and the logic to handle intents in NFV-MANO is out of the scope of the present document and will be determined in the normative work of ETSI GR NFV-IFA 041 [i.12].

7.1.3 Perspective#3: Use of policies to support Intent

As described in clause 6.2.2 of ETSI GR NFV-IFA 041 [i.12], there is a relationship between intent and policy. On one hand, intent can be regarded as a kind of declarative policy. On the other hand, the interpretation of an intent can derive polices to be applied to NFV-MANO as part of intent realization.

Based on analysis in clause 7.1.2 of the present document, no policy models can be used to describe declarative policy (e.g. intent). However, there is a possibility to enhance ONAP APEX policy model to support the description of a declarative policy.

The policies derived from the Intent Management interpretation and the logic to handle intents in NFV-MANO is out of the scope of the present document and will be determined in the normative work of ETSI GR NFV-IFA 041 [i.12].

7.1.4 Summary

Based on analysis in clauses 7.1.1, 7.1.2 and 7.1.3, recommendations for policy model requirements in clause 5.4 are NFV-MANO specific and can not be completely satisfied by any existing policy model from other SDOs described in clause 6. Among the three SDOs, ONAP APEX policy model is the one with minimum gaps to the recommendations in clause 5.4, which can satisfy most of the recommendations and some further enhancements to ONAP APEX policy model are needed.

7.2 Recommendations for NFV-MANO policy models

7.2.1 Introduction

Based on the introduction of policy type classifications in clause 5.2, policy model requirement recommendations in clause 5.4, the introduction of the industry's existing policy models in clause 6 and gap analysis in clause 7.1, recommendations for future work on policy model design are listed in clause 7.2.

7.2.2 Recommendations for future work on policy model architecture design

Table 7.2.2-1 provides the recommendations for future work related to policy model architecture.

Table 7.2.2-1: Recommendations on policy model architecture

Identifier	Recommendation description	Comment/Traceability
FUTURE.POL.MODEL.Arc.001	It is recommended to consider the classifications of policy types described in clause 5.2 to introduce them into the design of the policy model applicable to NFV-MANO.	Based on the introduction in clause 5.2.
FUTURE.POL.MODEL.ARC.002	It is recommended to consider the support of the policies for the intent handling as described in clause 6.2.2 of ETSI GR NFV-IFA 041 [i.12] (see note 2).	Based on the analysis in clause 7.1.3.
FUTURE.POL.MODEL.ARC.003	It is recommended to consider the policy model general recommendations listed in clause 5.4.1.	
FUTURE.POL.MODEL.ARC.004	It is recommended to consider the relationship between classes and the definitions of attributes described in clause 5.4 into the design of the policy model applicable to NFV-MANO.	Based on the analysis in clause 7.1.
FUTURE.POL.MODEL.ARC.005	It is recommended to consider the flexibility described in clause 5.4 into the design of the policy model applicable to NFV-MANO to support other types of policy paradigms, such as MEDA, OODA (see note 1).	Based on the analysis in clause 7.1 and the introduction of ONAP policy model in clause 6.3.2.3.3.4.
NOTE 1: Only ECA form is introduced in the present document. Other types of forms are out of the scope of the present document. NOTE 2: Intent is out of the scope of the present document. The present document only recommends to consider policy model used for intent handling when the policy model is designed.		

7.2.3 Recommendations for future work on policy model information elements design

Table 7.2.3-1 provides the recommendations for future work related to policy model information elements.

Table 7.2.3-1: Recommendations on policy model information elements

Identifier	Recommendation description	Comment/Traceability
FUTURE.POL.MODEL.INF.001	It is recommended to consider the support for corresponding information elements to describe the combinations and aggregations of policy items described in clause 5.4 and clause 6 to assure policy items could be combined and aggregated to be a complete complex policy.	Derived from the analysis in clause 7.1.
FUTURE.POL.MODEL.INF.002	It is recommended to consider the support for corresponding information elements to describe context-aware policy described in clause 5.3, clause 5.4 and clause 6 to allow the policy to be aware of the context.	Derived from the analysis in clause 7.1.
FUTURE.POL.MODEL.INF.003	It is recommended to consider the support for corresponding information elements to define the mapping relationships between policy items and to support different policy paradigms (MEDA, OODA, etc.) described in clause 6.	Derived from the analysis in clause 7.1.
FUTURE.POL.MODEL.INF.004	It is recommended to consider the recommendations for policy model information elements listed in clause 5.4.2.	

7.2.4 Recommendations for future work on policy model attributes design

Table 7.2.4-1 provides the recommendations for future work related to policy model attributes.

Table 7.2.4-1: Recommendations on policy model attributes

Identifier	Recommendation description	Comment/Traceability
FUTURE.POL.MODEL.ATT.001	It is recommended to consider the support for corresponding attributes to describe the context to allow the policy to be aware of the context.	Derived from the analysis in clause 7.1.
FUTURE.POL.MODEL.ATT.002	It is recommended to consider the support for corresponding attributes to describe the mapping relationships between policy items to allow policy items in different paradigms (MEDA, OODA, etc.) to be executed correctively.	Derived from the analysis in clause 7.1.
FUTURE.POL.MODEL.ATT.003	It is recommended to consider the support for corresponding attributes to support different types of policy paradigms	Derived from the analysis in clause 7.1.
FUTURE.POL.MODEL.ATT.004	It is recommended to consider the recommendations for policy model attributes listed in clause 5.4.3.	

Annex A: Conversion procedure and code

1) Install pyang tool.
2) Copy the yang file provided by [i.18] to a special directory, for example, ietf-eca.yang.
3) Login a terminal and execute commands to the directory defined in step 2.
4) Execute a command to generate the .uml file: pyang -f uml ietf-eca.yang -o ietf-eca.uml.
5) Copy the plantuml.jar to the same directory.
6) Execute a command to generate the UML class diagram: java -jar plantuml.jar ietf-eca.uml.
7) The diagram will be saved as a .png file in the corresponding directory.

NOTE: For the download of pyang tool in step 1 and plantuml.jar in step 5, please refer to [i.17] and [i.18].

Annex B: Change History

Date	Version	Information about changes
August 2020	0.0.1	First draft, implementing contributions: NFVIFA(20)000553r3 approved in IFA#205
February 2021	0.0.2	Implementations of approved contributions: NFVIFA(21)000012 accepted in IFA#222 NFVIFA(21)000003r3 accepted in IFA#225 NFVIFA(21)000004r3 accepted in IFA#225 NFVIFA(21)000005r3 accepted in IFA#225 NFVIFA(21)000006r4 accepted in IFA#225 NFVIFA(21)000009r1 accepted in IFA#222 NFVIFA(21)000010r3 accepted in IFA#225 NFVIFA(20)000615r2 accepted in IFA#210 NFVIFA(20)000616r6 accepted in IFA#210 NFVIFA(20)000617r1 accepted in IFA#210 NFVIFA(20)000618r2 accepted in IFA#210 NFVIFA(20)000619r2 accepted in IFA#210 NFVIFA(20)000620r1 accepted in IFA#210
April 2021	0.0.3	Implementations of approved contributions from IFA#226: NFVIFA(21)000304 NFVIFA(21)000305r2 NFVIFA(21)000306r3 NFVIFA(21)000307r2 NFVIFA(21)000308r3 NFVIFA(21)000309r4 NFVIFA(21)000310r3
May 2021	0.0.4	Implementations of approved contributions from IFA#238: NFVIFA(21)000310r3_IFA042_5_3_1_General_Recommendations NFVIFA(21)000311r2_IFA042_5_3_2_Recommendations_for_Policy_Model_Information_El NFVIFA(21)000312r2_IFA042_5_3_3_Recommendations_for_Policy_Model_Attributes NFVIFA(21)000313r2_IFA042_5_4_1_Examples__Potential_Attributes_of_the_VNF_Auto- NFVIFA(21)000314r2_IFA042_5_4_2_Examples__Potential_Attributes_of_the_NS_Self-h NFVIFA(21)000315r3_IFA042_5_4_3_Examples__Potential_Attributes_of_the_Virtualiz NFVIFA(21)000316r2_IFA042_6_3_2_3_A_YANG_Data_Model_for_ECA_Policy_Management
May 2021	0.0.5	Implementations of approved contributions from IFA#238: NFVIFA(21)000407r1 NFVIFA(21)000408 NFVIFA(21)000409r1
June 2021	0.0.6	Implementations of approved contributions from IFA#241: NFVIFA(21)000410r2 NFVIFA(21)000411r2 NFVIFA(21)000424r2 NFVIFA(21)000439r1 NFVIFA(21)000440r1 NFVIFA(21)000441r1 NFVIFA(21)000443r1 NFVIFA(21)000444r1 NFVIFA(21)000454r1 Implementations of approved contributions from IFA#242: NFVIFA(21)000442r1 NFVIFA(21)000445r1 NFVIFA(21)000446r1 NFVIFA(21)000487r1 NFVIFA(21)000488r1 NFVIFA(21)000489r1 NFVIFA(21)000503r1 NFVIFA(21)000520r1 NFVIFA(21)000530r1

Date	Version	Information about changes
July 2021	0.1.0	Implementations of approved contributions from IFA#243: NFVIFA(21)000538 Implementations of approved contributions from IFA#246: NFVIFA(21)000558r1 NFVIFA(21)000559r1 NFVIFA(21)000560r1 NFVIFA(21)000561r1 NFVIFA(21)000562 NFVIFA(21)000563r1 NFVIFA(21)000617r2 NFVIFA(21)000618r2
August 2021	0.2.0	Implementations of approved contributions from IFA#250: NFVIFA(21)000674r1 NFVIFA(21)000675r1 NFVIFA(21)000699 NFVIFA(21)000701r2
August 2021	0.3.0	Implementations of approved contributions from IFA#250: NFVIFA(21)000700r1 Implementations of approved contributions from IFA#251: NFVIFA(21)000727 NFVIFA(21)000728 NFVIFA(21)000729r1 NFVIFA(21)000730r1

History

Document history		
V4.1.1	November 2021	Publication